Bloque 10: Presentacion WEB

Paso 73: Web de presentacion

¡Excelente idea! Crear una página web para presentar el proyecto es el broche de oro. Le dará una apariencia profesional y te permitirá mostrar tu trabajo de una forma interactiva y ordenada.

He preparado una guía completa con todo el código necesario para que construyas una web sencilla pero funcional usando **Flask**, un micro-framework de Python. Esta solución te permitirá tener una web que se ejecuta en tu propio ordenador y que puede "llamar" a tu script de inferencia de Python para procesar imágenes en tiempo real.

Plan de Acción: Tu Web de Presentación en 5 Pasos

Vamos a organizar tu proyecto web de la siguiente manera:

Paso 1: Organizar la Estructura de Carpetas

Primero, crea una carpeta principal para tu web, por ejemplo, `proyecto_web`. Dentro de ella, crea la siguiente estructura. Es importante mantener estos nombres para que el código funcione directamente.

```
/proyecto_web/
|
|-- /static/
|   |-- style.css       # ← Nuestro archivo de estilos (CSS)
|   |-- script.js       # ← Nuestro archivo de lógica del navegador (JavaScript)
|
|-- /templates/
|   |-- base.html       # ← Una plantilla base para no repetir código
|   |-- inferencia.html  # ← La página para probar el modelo
|   |-- fases.html      # ← La página para mostrar tus HTML
|
|-- /fases_html/         # ← ¡AQUÍ DENTRO PEGARÁS TUS HTML!
|   |-- menu.html
|   |-- fase1.html
|   |-- etc...
```

```
|
|-- /uploads/          # ← Carpeta para guardar imágenes subidas (se creará
|
|-- app.py             # ← El "corazón" de nuestra web (el servidor Flask)
|
|-- best.pt            # ← Copia aquí tu mejor modelo
```

Acciones:

1. Crea esta estructura de carpetas.
2. **Copia tu archivo** `best.pt` (`Modelo_Hibrido_v1/weights/best.pt`) dentro de la carpeta `proyecto_web`.
3. **Pega todos tus archivos HTML** (el menú y los de las fases) dentro de la carpeta `fases_html`.

Paso 2: Instalar Flask

Flask es la librería que nos permitirá crear el servidor web con Python.

1. Abre tu terminal y **activa tu entorno virtual** (`.venv`).
2. Instala Flask:

```
pip install Flask
```

Paso 3: El Código del "Corazón" de la Web (`app.py`)

Este es el script de Python que actuará como servidor. Se encargará de mostrar las páginas y de llamar a tu modelo YOLO cuando un usuario suba una imagen.

Crea el archivo `app.py` dentro de `proyecto_web` y pega el siguiente código:

```
# app.py
import os
from flask import Flask, render_template, request, jsonify
from ultralytics import YOLO
from werkzeug.utils import secure_filename
import uuid

# --- CONFIGURACIÓN ---
```

```
UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

# Creamos la aplicación Flask
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Cargamos el modelo YOLO una sola vez al iniciar la aplicación
print("Cargando modelo YOLO...")
model = YOLO('best.pt')
print("¡Modelo cargado!")

# Lista de clases (debe coincidir con tu modelo híbrido)
CLASES_HIBRIDAS = [
    'black_8', 'blue_10', 'blue_2', 'dred_15', 'dred_7', 'green_14', 'green_6',
    'orange_13', 'orange_5', 'purple_12', 'purple_4', 'red_11', 'red_3',
    'white', 'yellow_1', 'yellow_9', 'be_blue_10', 'be_dred_15',
    'be_green_14', 'be_purple_13', 'be_purple_5', 'be_pink_4', 'be_pink_12',
    'be_red_11', 'be_yellow_9'
]

# --- LÓGICA DE POST-PROCESAMIENTO ---
# (La misma lógica que ya validamos)
CLASES_COMPARTIDAS = {'black_8', 'blue_2', 'dred_7', 'green_6', 'red_3', 'whi
DELATORES_BE = {'be_blue_10', 'be_dred_15', 'be_green_14', 'be_purple_13', 'b
DELATORES_CLASSIC = {'blue_10', 'dred_15', 'green_14', 'orange_13', 'orange_5

def post_procesar_con_razonamiento(detecciones_yolo):
    puntuacion_be = sum(d['conf'] for d in detecciones_yolo if d['clase'] in DEL
    puntuacion_classic = sum(d['conf'] for d in detecciones_yolo if d['clase'] in

    contexto = 'black_edition' if puntuacion_be > puntuacion_classic else 'class

    detecciones_refinadas = []
    for deteccion in detecciones_yolo:
        etiqueta_original = deteccion['clase']
        etiqueta_final = etiqueta_original
```

```
        if contexto == 'black_edition' and etiqueta_original in CLASES_COMPART
            etiqueta_final = 'be_' + etiqueta_original

        deteccion['etiqueta_final'] = etiqueta_final
        detecciones_refinadas.append(deteccion)

    return detecciones_refinadas

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENS

# --- DEFINICIÓN DE LAS PÁGINAS (RUTAS) ---

@app.route('/')
def home():
    return render_template('base.html')

@app.route('/inferencia')
def inferencia_page():
    return render_template('inferencia.html')

@app.route('/fases')
def fases_page():
    return render_template('fases.html')

# --- API PARA LA PREDICCIÓN ---

@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No se encontró el archivo'}), 400
    file = request.files['file']
    if file.filename == '' or not allowed_file(file.filename):
        return jsonify({'error': 'Archivo no válido'}), 400

    # Guardar el archivo subido
    filename = secure_filename(file.filename)
    upload_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
```

```
    file.save(upload_path)

    # Realizar la predicción
    results = model(upload_path, verbose=False)
    result = results[0]

    # Pre-procesar para la lógica contextual
    detecciones_yolo = [{'clase': CLASES_HIBRIDAS[int(box.cls[0])], 'conf': floa

    # Aplicar el "cerebro" contextual
    detecciones_finales = post_procesar_con_razonamiento(detecciones_yolo)

    # Dibujar resultados y guardar
    img = result.plot(labels=False, line_width=2) # Dibujar cajas sin etiquetas

    # Añadir nuestras etiquetas finales corregidas
    for deteccion in detecciones_finales:
        x1, y1, x2, y2 = map(int, deteccion['box_xyxy'])
        label = f"{deteccion['etiqueta_final']} {deteccion['conf']:.2f}"
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(img, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (

    # Guardar la imagen resultante
    result_filename = f"result_{uuid.uuid4()}.jpg"
    result_path = os.path.join('static', result_filename)
    cv2.imwrite(result_path, img)

    return jsonify({'result_image': result_path})

# --- Iniciar la aplicación ---
if __name__ == '__main__':
    os.makedirs(UPLOAD_FOLDER, exist_ok=True)
    app.run(debug=True)
```

Paso 4: El Código de las Plantillas HTML y JavaScript

Ahora crearemos los archivos que van en las carpetas `templates` y `static`.

`templates/base.html` (La plantilla principal)

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Proyecto IA - Billar</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <header>
    <h1>Proyecto Final: IA aplicada al Billar</h1>
    <nav>
      <a href="{{ url_for('inferencia_page') }}">Probar Modelo</a>
      <a href="{{ url_for('fases_page') }}">Fases del Proyecto</a>
    </nav>
  </header>
  <main>
    {% block content %}{% endblock %}
  </main>
</body>
</html>
```

templates/inferencia.html (La página de prueba)

```
{% extends "base.html" %}
{% block content %}
<div class="container">
  <h2>Prueba de Inferencia en Tiempo Real</h2>
  <p>Sube una imagen de una mesa de billar para ver las detecciones del mo
  <div class="inference-box">
    <input type="file" id="imageUpload" accept="image/*">
    <button id="predictBtn">Detectar Bolas</button>
    <div id="spinner" class="spinner" style="display:none;"></div>
  </div>
  <div class="results-container">
    <div class="image-box">
      <h3>Imagen Original</h3>
      <img id="imagePreview" src="" alt="Vista previa de la imagen">
```

```
        </div>
        <div class="image-box">
            <h3>Resultado de la Detección</h3>
            <img id="resultImage" src="" alt="Resultado del modelo">
        </div>
    </div>
</div>
<script src="{{ url_for('static', filename='script.js') }}"></script>
{% endblock %}
```

`templates/fases.html` (La página para tus documentos)

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h2>Fases y Documentación del Proyecto</h2>
    <p>Usa el menú para navegar por los diferentes documentos HTML del pro
    <div class="iframe-container">
        <iframe name="iframe_contenido" src="../fases_html/menu.html"></ifrar
    </div>
</div>
{% endblock %}
```

IMPORTANTE: Para que funcione el menú en la página de fases, abre tu archivo `menu.html` y asegúrate de que todos los enlaces (`<a>`) tengan el atributo `target="iframe_contenido"`. Por ejemplo: `<a href="fase1.html" target="iframe_contenido">Ver Fase 1</a>`.

`static/style.css` (Para que se vea bien)

```
body { font-family: sans-serif; margin: 0; background-color: #f4f4f9; color: #3
header { background-color: #003366; color: white; padding: 1rem; text-align:
nav { margin-top: 1rem; }
nav a { color: white; margin: 0 15px; text-decoration: none; font-weight: bold;
main { padding: 2rem; }
.container { max-width: 1200px; margin: auto; background: white; padding: 2r
.inference-box { text-align: center; margin-bottom: 2rem; }
.results-container { display: flex; justify-content: space-around; gap: 2rem; }
.image-box { width: 45%; border: 1px solid #ddd; padding: 1rem; }
```

```
.image-box h3 { text-align: center; }
img { max-width: 100%; height: auto; }
.iframe-container { width: 100%; height: 80vh; border: 1px solid #ddd; }
iframe { width: 100%; height: 100%; border: none; }
.spinner { border: 8px solid #f3f3f3; border-top: 8px solid #003366; border-ra
@keyframes spin { 0% { transform: rotate(0deg); } 100% { transform: rotate(3
```

`static/script.js` (La lógica del navegador)

```
document.addEventListener('DOMContentLoaded', () => {
    const imageUpload = document.getElementById('imageUpload');
    const imagePreview = document.getElementById('imagePreview');
    const predictBtn = document.getElementById('predictBtn');
    const resultImage = document.getElementById('resultImage');
    const spinner = document.getElementById('spinner');

    imageUpload.addEventListener('change', () => {
        const file = imageUpload.files[0];
        if (file) {
            imagePreview.src = URL.createObjectURL(file);
            resultImage.src = "";
        }
    });

    predictBtn.addEventListener('click', async () => {
        const file = imageUpload.files[0];
        if (!file) {
            alert("Por favor, selecciona una imagen primero.");
            return;
        }

        spinner.style.display = 'block';
        resultImage.src = "";

        const formData = new FormData();
        formData.append('file', file);

        try {
```

```
            const response = await fetch('/predict', {
                method: 'POST',
                body: formData
            });

            if (response.ok) {
                const data = await response.json();
                // Añadimos un timestamp para evitar problemas de caché del naveg
                resultImage.src = data.result_image + '?t=' + new Date().getTime();
            } else {
                const error = await response.json();
                alert(`Error: ${error.error}`);
            }
        } catch (error) {
            alert(`Ocurrió un error de red: ${error}`);
        } finally {
            spinner.style.display = 'none';
        }
    });
});
```

Paso 5: ¡Lanzar la Web!

1. Abre tu terminal.
2. Navega hasta la carpeta raíz de tu proyecto web (`cd ruta/a/proyecto_web`).
3. **Activa tu entorno virtual** `.venv`.
4. Ejecuta el servidor Flask:

   ```
   python app.py
   ```

5. Verás un mensaje que dice algo como `Running on http://127.0.0.1:5000`.
6. **Abre tu navegador web** y ve a esa dirección: `http://127.0.0.1:5000`.

¡Y ya está! Tendrás tu propia web local funcionando, con las dos pestañas que pediste, y lista para mostrar tu increíble trabajo.

¡Excelente idea! Añadir los informes en PDF y los audios explicativos enriquecerá enormemente la presentación de tu proyecto. Lo convertirá de una simple web a un portfolio multimedia completo.

La buena noticia es que, gracias a cómo configuramos el servidor Flask en el paso anterior, añadir estos archivos es **extremadamente sencillo** y no requiere tocar el archivo `app.py`.

Plan de Acción: Añadir PDFs y Audios

Paso 1: Organizar tus Archivos Multimedia

Para mantener el proyecto ordenado, vamos a crear carpetas específicas para tus PDFs y audios dentro de la carpeta `fases_html`.

1. Dentro de tu carpeta `proyecto_web/fases_html/`, crea dos nuevas carpetas:
   - `pdf`
   - `audio`
2. **Copia tus archivos:**
   - Mueve todos tus informes `.pdf` a la nueva carpeta `fases_html/pdf/`.
   - Mueve todos tus archivos de audio `.mp3` a la nueva carpeta `fases_html/audio/`.

Tu estructura de carpetas ahora se verá así:

```
/proyecto_web/
|
|-- /fases_html/
|   |-- /pdf/
|   |   |-- informe_fase1.pdf
|   |   |-- informe_fase2.pdf
|   |
|   |-- /audio/
|   |   |-- audio_fase1.mp3
|   |   |-- audio_fase2.mp3
|   |
|   |-- menu.html
```

```
|   |-- fase1.html
|   |-- fase2.html
|   |-- ...
|
|-- ... (resto de carpetas y archivos)
```

Paso 2: Añadir los Enlaces en tus Archivos HTML

Ahora, simplemente tienes que editar cada uno de tus archivos `faseX.html` para añadir los enlaces al PDF y el reproductor de audio correspondientes.

Cómo Añadir el Enlace al PDF

Usa la etiqueta `<a>` estándar. Te recomiendo añadir `target="_blank"` para que el PDF se abra en una nueva pestaña del navegador, lo cual es una mejor experiencia para el usuario.

```
<a href="pdf/informe_fase1.pdf" target="_blank">Ver/Descargar Informe Com
```

Cómo Añadir el Reproductor de Audio MP3

Usa la etiqueta `<audio>` de HTML5. El atributo `controls` le dice al navegador que muestre los controles de reproducción (play, pausa, volumen).

```
<h4>Explicación en Audio:</h4>
<audio controls>
  <source src="audio/audio_fase1.mp3" type="audio/mpeg">
  Tu navegador no soporta el elemento de audio.
</audio>
```

Ejemplo Completo para `fase1.html`

Aquí tienes un ejemplo de cómo podrías editar tu archivo `fase1.html` para incluir ambos elementos.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
```

```
    <title>Fase 1 del Proyecto</title>
    <style>
        body {
            font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto,
            margin: 2em;
        }
        h1 {
            color: #003366;
        }
        .recurso {
            margin-top: 25px;
            padding: 15px;
            border: 1px solid #ddd;
            border-radius: 5px;
            background-color: #f9f9f9;
        }
    </style>
</head>
<body>

    <h1>Fase 1: Recolección y Análisis de Datos</h1>

    <p>
        Aquí va toda tu explicación sobre la fase 1, los desafíos que encontraste,
        el análisis de los datasets, etc...
    </p>

    <div class="recurso">
        <h3>Informe Detallado</h3>
        <p>Para un análisis más profundo, consulta el documento completo en P
        <a href="pdf/informe_fase1.pdf" target="_blank">Ver/Descargar Informe
    </div>

    <div class="recurso">
        <h3>Resumen en Audio</h3>
        <p>Escucha un resumen de los puntos clave de esta fase.</p>
        <audio controls style="width: 100%;">
            <source src="audio/audio_fase1.mp3" type="audio/mpeg">
```

```
            Tu navegador no soporta el elemento de audio.
        </audio>
    </div>

</body>
</html>
```

Tu Tarea Final:

1. Crea las carpetas `pdf` y `audio` dentro de `fases_html`.
2. Copia tus archivos correspondientes a estas nuevas carpetas.
3. Edita cada uno de tus archivos `faseX.html` para añadir los fragmentos de código `<a>` y `<audio>`, asegurándote de que los nombres de los archivos (`informe_fase1.pdf`, `audio_fase1.mp3`, etc.) coincidan.

No necesitas hacer ningún otro cambio. Gracias a la ruta que creamos en `app.py`, el servidor ya sabe cómo encontrar y servir estos archivos. Simplemente guarda los cambios en tus HTML y refresca la página en el navegador.

¡Con esto, tu web de presentación se convertirá en un portfolio multimedia completo y muy profesional!

¡Excelente idea para mejorar la interfaz! Añadir una función de zoom o "lightbox" para ver las imágenes en detalle es un toque muy profesional y mejora mucho la experiencia de usuario al presentar los resultados.

Para lograrlo, necesitamos hacer tres cosas:

1. **HTML:** Añadir una "ventana modal" oculta a nuestra página de inferencia.
2. **CSS:** Darle estilo a esa ventana para que aparezca como una superposición que ocupe toda la pantalla.
3. **JavaScript:** Añadir la lógica para que la ventana se muestre al hacer clic en una imagen y se oculte al cerrarla.

He preparado el código para que solo tengas que copiar y pegar.

Paso 1: Modificar el HTML (`templates/inferencia.html`)

Vamos a añadir dos cosas a este archivo:

- La clase `zoomable` a tus imágenes para que el cursor cambie y sepamos que son clicables.
- El `div` de la ventana modal, que estará oculto por defecto.

Reemplaza el contenido de `templates/inferencia.html` con este código actualizado:

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <h2>Prueba de Inferencia en Tiempo Real</h2>
    <p>Sube una imagen de una mesa de billar para ver las detecciones del mo
    <div class="inference-box">
        <input type="file" id="imageUpload" accept="image/*">
        <button id="predictBtn">Detectar Bolas</button>
        <div id="spinner" class="spinner" style="display:none;"></div>
    </div>
    <div class="results-container">
        <div class="image-box">
            <h3>Imagen Original</h3>
            <img id="imagePreview" src="" alt="Vista previa de la imagen" class=
        </div>
        <div class="image-box">
            <h3>Resultado de la Detección</h3>
            <img id="resultImage" src="" alt="Resultado del modelo" class="zoom
        </div>
    </div>
</div>

<div id="imageModal" class="modal">
    <span class="close-btn">&times;</span>
    <img class="modal-content" id="modalImage">
</div>

<script src="{{ url_for('static', filename='script.js') }}"></script>
{% endblock %}
```

Paso 2: Añadir Estilos para la Ventana Modal (`static/style.css`)

Abre tu archivo `static/style.css` y **añade el siguiente código al final de todo**. No borres lo que ya tenías.

```
/* --- Estilos para la Imagen Ampliable y la Ventana Modal --- */

/* Estilo para que el cursor cambie al pasar sobre las imágenes */
.zoomable {
    cursor: pointer;
    transition: transform 0.2s ease-in-out;
}

.zoomable:hover {
    transform: scale(1.03); /* Efecto de zoom sutil al pasar el ratón */
}

/* El fondo de la ventana modal (la superposición oscura) */
.modal {
    display: none; /* Oculto por defecto */
    position: fixed; /* Se queda fijo en la pantalla */
    z-index: 1000; /* Se asegura de que esté por encima de todo */
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    overflow: auto; /* Permite scroll si la imagen es muy grande */
    background-color: rgba(0, 0, 0, 0.9); /* Fondo negro semitransparente */
}

/* El contenido de la modal (la imagen ampliada) */
.modal-content {
    margin: auto;
    display: block;
    max-width: 80%;
    max-height: 80vh; /* Ocupa como máximo el 80% de la altura de la ventana
    /* Animación de zoom al aparecer */
    animation-name: zoom;
    animation-duration: 0.4s;
```

```
}

@keyframes zoom {
    from {transform: scale(0)}
    to {transform: scale(1)}
}

/* El botón de cerrar (la 'X') */
.close-btn {
    position: absolute;
    top: 15px;
    right: 35px;
    color: #f1f1f1;
    font-size: 40px;
    font-weight: bold;
    transition: 0.3s;
}

.close-btn:hover,
.close-btn:focus {
    color: #bbb;
    text-decoration: none;
    cursor: pointer;
}
```

Paso 3: Actualizar la Lógica (static/script.js)

Finalmente, reemplaza todo el contenido de tu archivo `static/script.js` con esta nueva versión, que incluye la lógica para abrir y cerrar la ventana modal.

```
// static/script.js (Versión con Modal)
document.addEventListener('DOMContentLoaded', () => {
    // Elementos de la inferencia
    const imageUpload = document.getElementById('imageUpload');
    const imagePreview = document.getElementById('imagePreview');
    const predictBtn = document.getElementById('predictBtn');
    const resultImage = document.getElementById('resultImage');
    const spinner = document.getElementById('spinner');
```

```
// --- NUEVO: Elementos de la ventana modal ---
const modal = document.getElementById('imageModal');
const modalImage = document.getElementById('modalImage');
const closeBtn = document.querySelector('.close-btn');

// Función para abrir la modal
function openModal(imgElement) {
    if (imgElement && imgElement.src && !imgElement.src.endsWith('/')) {
        modal.style.display = "block";
        modalImage.src = imgElement.src;
    }
}

// Añadir evento de clic a las imágenes "zoomables"
imagePreview.addEventListener('click', () => openModal(imagePreview));
resultImage.addEventListener('click', () => openModal(resultImage));

// Añadir evento para cerrar la modal con el botón 'X'
closeBtn.addEventListener('click', () => {
    modal.style.display = "none";
});

// Cerrar la modal también al hacer clic en el fondo oscuro
modal.addEventListener('click', (event) => {
    if (event.target === modal) {
        modal.style.display = "none";
    }
});

// Lógica para la subida y previsualización de la imagen (sin cambios)
imageUpload.addEventListener('change', () => {
    const file = imageUpload.files[0];
    if (file) {
        const reader = new FileReader();
        reader.onload = (e) => {
            imagePreview.src = e.target.result;
        }
```

```
        reader.readAsDataURL(file);
        resultImage.src = "";
    }
});

// Lógica para la predicción (sin cambios)
predictBtn.addEventListener('click', async () => {
    const file = imageUpload.files[0];
    if (!file) {
        alert("Por favor, selecciona una imagen primero.");
        return;
    }

    spinner.style.display = 'block';
    resultImage.src = "";

    const formData = new FormData();
    formData.append('file', file);

    try {
        const response = await fetch('/predict', {
            method: 'POST',
            body: formData
        });

        if (response.ok) {
            const data = await response.json();
            resultImage.src = data.result_image + '?t=' + new Date().getTime();
        } else {
            const error = await response.json();
            alert(`Error: ${error.error}`);
        }
    } catch (error) {
        alert(`Ocurrió un error de red: ${error}`);
    } finally {
        spinner.style.display = 'none';
    }
```

```
    });
});
```

Resumen de los Pasos

1. Reemplaza el contenido de `templates/inferencia.html`.
2. Añade el nuevo código CSS al final de `static/style.css`.
3. Reemplaza todo el contenido de `static/script.js`.
4. Refresca la página de tu web en el navegador (`Ctrl+Shift+R` o `Cmd+Shift+R` para asegurar que carga los archivos nuevos).

Con estos cambios, ahora podrás hacer clic tanto en la imagen original como en la de resultados para ver una versión ampliada y poder inspeccionar los detalles de las detecciones cómodamente. ¡Es el toque final para una presentación de proyecto de primera!

¡Excelente! Entiendo perfectamente lo que necesitas. Quieres que tu web no sea solo una "demo", sino que utilice el **potente motor de inferencia y post-procesamiento** que hemos construido y depurado juntos.

Este es un paso crucial y muy profesional. Vamos a reestructurar el código para que tu servidor web (`app.py`) pueda "llamar" a tu script de inferencia como si fuera un módulo. Esto se conoce como **refactorización** y es una práctica de ingeniería de software excelente.

He preparado todo el código que necesitas. Solo tendrás que reemplazar dos archivos.

Diagnóstico del Problema: `matplotlib` en un Servidor Web

Antes de darte el código, es importante que entiendas por qué no podemos usar tu script de inferencia tal cual. Tu script actual usa la librería `matplotlib.pyplot` (`plt`) para dibujar las imágenes. `matplotlib` es fantástica para generar gráficos y para análisis en un entorno de escritorio, pero **no está diseñada para funcionar en un servidor web multiusuario.** No es "thread-safe", lo que significa que si dos personas intentaran subir una imagen a tu web al mismo tiempo, el servidor se colgaría o se cerraría.

La solución estándar y profesional es usar la librería `cv2` **(OpenCV)** para todas las tareas de manipulación de imágenes en el backend, ya que es extremadamente rápida y segura para estos entornos.

Paso 1: Crear el "Motor" de Inferencia (`motor_inferencia.py`)

Vamos a tomar toda la lógica de tu script de inferencia y la empaquetaremos en un nuevo archivo. Este archivo será nuestro "motor" reutilizable.

1. En la raíz de tu proyecto web (`/proyecto_web/`), crea un nuevo archivo llamado `motor_inferencia.py` .
2. Pega el siguiente código en él. He adaptado tu script, convirtiéndolo en una única función `procesar_imagen` y reemplazando `matplotlib` por `cv2` para dibujar.

`motor_inferencia.py` (Tu nuevo motor)

```
# motor_inferencia.py
import os
import cv2
from ultralytics import YOLO

# --- CONFIGURACIÓN Y LISTAS DE CLASES ---
# (Las mismas que ya validamos)
CLASES_HIBRIDAS = [
    'black_8', 'blue_10', 'blue_2', 'dred_15', 'dred_7', 'green_14', 'green_6',
    'orange_13', 'orange_5', 'purple_12', 'purple_4', 'red_11', 'red_3',
    'white', 'yellow_1', 'yellow_9', 'be_blue_10', 'be_dred_15',
    'be_green_14', 'be_purple_13', 'be_purple_5', 'be_pink_4', 'be_pink_12',
    'be_red_11', 'be_yellow_9'
]
DELATORES_BE = {'be_blue_10', 'be_dred_15', 'be_green_14', 'be_purple_13', 'b
DELATORES_CLASSIC = {'blue_10', 'dred_15', 'green_14', 'orange_13', 'orange_5
MAPA_CORRECCION_CONTEXTUAL = {
    'be_pink_4': 'red_3', 'be_pink_12': 'red_11', 'be_purple_5': 'purple_4',
    'be_purple_13': 'purple_12', 'be_yellow_9': 'yellow_9', 'be_dred_15': 'dred_15',
    'be_blue_10': 'blue_10', 'be_green_14': 'green_14', 'be_red_11': 'red_11',
    'purple_4': 'be_pink_4', 'purple_12': 'be_pink_12', 'orange_5': 'be_purple_5',
    'orange_13': 'be_purple_13', 'yellow_9': 'be_yellow_9'
}
```

```
# --- CARGA DEL MODELO (una sola vez) ---
try:
    print("Cargando modelo YOLO para el motor de inferencia...")
    model = YOLO('best.pt')
    print("¡Motor de inferencia listo!")
except Exception as e:
    print(f"Error fatal al cargar el modelo en el motor: {e}")
    model = None

# --- LÓGICA DE POST-PROCESAMIENTO ---
def post_procesar_con_razonamiento(detecciones_yolo):
    puntuacion_be = sum(d['conf'] for d in detecciones_yolo if d['clase'] in DEL
    puntuacion_classic = sum(d['conf'] for d in detecciones_yolo if d['clase'] in

    contexto = 'black_edition' if puntuacion_be > puntuacion_classic else 'class

    detecciones_corregidas = []
    for deteccion in detecciones_yolo:
        etiqueta_yolo = deteccion['clase']
        etiqueta_corregida = etiqueta_yolo

        if contexto == 'classic' and etiqueta_yolo in DELATORES_BE:
            etiqueta_corregida = MAPA_CORRECCION_CONTEXTUAL.get(etiqueta_
        elif contexto == 'black_edition' and etiqueta_yolo in DELATORES_CLASSI
            etiqueta_corregida = MAPA_CORRECCION_CONTEXTUAL.get(etiqueta_

        etiqueta_simple = etiqueta_corregida.replace('be_', '')
        deteccion['etiqueta_final'] = etiqueta_simple
        detecciones_corregidas.append(deteccion)

    return detecciones_corregidas

# --- FUNCIÓN PRINCIPAL DEL MOTOR ---
def procesar_imagen(input_path, output_path):
    """
    Función principal que toma una imagen, la procesa y guarda el resultado.
    Devuelve la lista de detecciones finales.
    """
```

```
    if not model:
        raise Exception("El modelo YOLO no se ha cargado correctamente.")

    # 1. Realizar predicción
    results = model.predict(source=input_path, save=False, imgsz=1024, conf=
    result = results[0]

    if not result.boxes:
        # Si no hay detecciones, simplemente copiamos la imagen original como
        img = cv2.imread(input_path)
        cv2.imwrite(output_path, img)
        return []

    # 2. Estructurar y post-procesar detecciones
    detecciones_yolo = [{'clase': CLASES_HIBRIDAS[int(box.cls[0])], 'conf': floa
    detecciones_finales = post_procesar_con_razonamiento(detecciones_yolo)

    # 3. Dibujar resultados con OpenCV (cv2)
    img = cv2.imread(input_path)
    for deteccion in detecciones_finales:
        x1, y1, x2, y2 = map(int, deteccion['box_xyxy'])
        label = f"{deteccion['etiqueta_final']} {deteccion['conf']:.2f}"
        color = (0, 255, 0) # Verde en BGR

        cv2.rectangle(img, (x1, y1), (x2, y2), color, 2)

        # Fondo para el texto
        (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
        cv2.rectangle(img, (x1, y1 - 20), (x1 + w, y1), color, -1)
        cv2.putText(img, label, (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0

    # 4. Guardar la imagen final
    cv2.imwrite(output_path, img)

    return detecciones_finales
```

Paso 2: Modificar `app.py` para Usar el Motor

Ahora, vamos a "vaciar" el `app.py` de la lógica de IA y le enseñaremos a importar y usar la función `procesar_imagen` que acabamos de crear.

Reemplaza TODO el contenido de tu archivo `app.py` con este nuevo código:

```
# app.py (Versión final que usa el motor de inferencia)
import os
import uuid
from flask import Flask, render_template, request, jsonify, url_for, send_from_d
from werkzeug.utils import secure_filename

# --- IMPORTAMOS NUESTRO MOTOR DE INFERENCIA ---
from motor_inferencia import procesar_imagen

# --- CONFIGURACIÓN DE FLASK ---
UPLOAD_FOLDER = 'uploads'
STATIC_FOLDER = 'static'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENS

# --- DEFINICIÓN DE LAS PÁGINAS (RUTAS) ---
@app.route('/')
def home():
    return render_template('inferencia.html')

@app.route('/inferencia')
def inferencia_page():
    return render_template('inferencia.html')

@app.route('/fases')
def fases_page():
    return render_template('fases.html')

@app.route('/fases_html/<path:filename>')
```

```
def serve_fases_html(filename):
    return send_from_directory('fases_html', filename)

# --- API PARA LA PREDICCIÓN ---
@app.route('/predict', methods=['POST'])
def predict():
    try:
        if 'file' not in request.files:
            return jsonify({'error': 'No se encontró el archivo'}), 400
        file = request.files['file']
        if file.filename == '' or not allowed_file(file.filename):
            return jsonify({'error': 'Archivo no válido'}), 400

        # Guardar el archivo subido
        filename = secure_filename(str(uuid.uuid4()) + "_" + file.filename)
        upload_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(upload_path)

        # Definir la ruta de salida para la imagen con resultados
        result_filename = f"result_{filename}"
        result_path_on_disk = os.path.join(STATIC_FOLDER, result_filename)

        # --- LLAMADA A NUESTRO MOTOR DE INFERENCIA ---
        # Pasamos la ruta de entrada y la de salida
        detecciones = procesar_imagen(upload_path, result_path_on_disk)
        # ---------------------------------------------

        # Devolvemos la URL para que el navegador pueda mostrar la imagen
        result_url = url_for('static', filename=result_filename)

        # También podemos devolver las detecciones por si queremos mostrarla
        return jsonify({'result_image': result_url, 'detections': detecciones})

    except Exception as e:
        print(f"ERROR en /predict: {e}")
        return jsonify({'error': f'Ocurrió un error en el servidor: {e}'}), 500

# --- Iniciar la aplicación ---
```

```
if __name__ == '__main__':
    os.makedirs(UPLOAD_FOLDER, exist_ok=True)
    os.makedirs(STATIC_FOLDER, exist_ok=True)
    app.run(debug=True)
```

Plan de Ejecución Final

1. **Crea** `motor_inferencia.py` con el primer bloque de código.
2. **Reemplaza** el contenido de tu `app.py` con el segundo bloque de código.
3. **Detén tu servidor Flask** si estaba en marcha (`Ctrl+C`).
4. **Vuelve a lanzarlo:** `python app.py` .

¡Y ya está! Tu aplicación web ahora tiene una arquitectura mucho más limpia y profesional. El servidor web se encarga de las tareas web, y tu "motor de IA" se encarga del trabajo pesado. Y lo más importante, es una solución robusta y preparada para funcionar sin los problemas que `matplotlib` podría causar.

Pruébalo y verás que funciona exactamente igual, pero ahora está construido de la manera correcta.