Bloque 15: Scripts

app.py

```
# app.py (Versión final que usa el motor de inferencia externo)
import os
import uuid

from flask import Flask, jsonify, render_template, request, send_from_directory, url_for
from werkzeug.utils import secure_filename

# --- ¡IMPORTAMOS NUESTRO MOTOR DE INFERENCIA! ---
from motor_inferencia import realizar_inferencia

# --- 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_EXTENSIONS


# --- DEFINICIÓN DE LAS PÁGINAS (RUTAS) ---
@app.route("/")
@app.route("/inferencia")
def inferencia_page():
    """
    Esta única función ahora maneja tanto la página de inicio ('/')
    como la de inferencia ('/inferencia'), haciendo que 'url_for' la encuentre.
    """
    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

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

        result_filename = f"result_{filename}"
        result_path_on_disk = os.path.join(STATIC_FOLDER, result_filename)

        # --- LLAMADA A NUESTRO MOTOR DE IA ---
        # Ahora devuelve también el contexto para que lo podamos mostrar
        detecciones, contexto = realizar_inferencia(upload_path, result_path_on_disk)
        # -------------------------------------------------

        result_url = url_for("static", filename=result_filename)

        # Devolvemos la imagen, las detecciones Y el contexto
        return jsonify(
            {"result_image": result_url, "detections": detecciones, "context": contexto}
        )

    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)
```

detect_balls/train_yolo_model.py

```
import os

from ultralytics import YOLO

# --- 1. Definir rutas y parámetros ---

# data_yaml_file = "custom_data.yaml"
data_yaml_file = "custom_data_shared_labels.yaml"

base_project_dir = os.path.join(".", "detect_balls")
data_yaml_path = os.path.join(base_project_dir, data_yaml_file)
execution_name = "Modelo_Hibrido_v1"
```

```
execution_epochs = 150
execution_image_resize = 640
execution_batch = 24
early_stop = 30
tasa_aprendizaje = 0.01

# Directorio donde se guardarán los resultados del entrenamiento (pesos, logs, etc.)
runs_dir = os.path.join(base_project_dir, "runs")
os.makedirs(runs_dir, exist_ok=True)  # Crea el directorio si no existe

# --- 2. Cargar un modelo pre-entrenado (ej. YOLOv8n) ---
# 'n' es por nano, la versión más pequeña y rápida para empezar.

# print("Cargando modelo YOLOv8n pre-entrenado...")
# model = YOLO(os.path.join(base_project_dir, "yolo8", "yolov8n.pt"))

# print("Cargando modelo YOLO11n pre-entrenado...")
# model = YOLO(os.path.join(base_project_dir, "yolo11", "yolo11n.pt"))

# print("Cargando modelo YOLO11m pre-entrenado...")
model = YOLO(os.path.join(base_project_dir, "models_yolo11", "yolo11m.pt"))

# print("Cargando modelo BEST classic pool balls: pool_classic.pt...")
# model = YOLO(os.path.join(base_project_dir, "models_custom", "pool_classic.pt"))

# print("Cargando modelo best.pt del ultimo entrenamiento...")
# model = YOLO(os.path.join(runs_dir, execution_name, "weights", "last.pt"))


# print("Cargando modelo best.pt del ultimo entrenamiento...")
# model = YOLO(os.path.join(base_project_dir, "poolballs46.pt"))
# model = YOLO(os.path.join(runs_dir, execution_name, "weights", "best.pt"))

# --- 3. Entrenar el modelo ---
# Documentación de los argumentos de entrenamiento: https://docs.ultralytics.com/usage/train/

# ========================================================================
# FASE 1: ENTRENAR SOLO LA "CABEZA" (CONGELANDO EL CONOCIMIENTO BASE)
# ========================================================================
"""
print("--- INICIANDO FASE 1: ENTRENAMIENTO DE LA CABEZA (BACKBONE CONGELADO) ---")

# Cargar el modelo experto en bolas clásicas
model_fase1 = YOLO(os.path.join(base_project_dir, "models_custom", "pool_classic.pt"))

# El parámetro clave aquí es 'freeze=11'. Congela las primeras 11 capas del modelo.
# Usamos menos épocas y paciencia porque solo queremos estabilizar la nueva capa de clasificación.
results_fase1 = model_fase1.train(
    data=data_yaml_path,
    epochs=50,
    patience=15,
    batch=24,
    imgsz=640,
```

```
    name="Supermodelo_Fase1_Head",
    project=runs_dir,
    freeze=11,  # ¡Parámetro clave! Congela las capas del backbone.
)

print("\n--- FASE 1 COMPLETADA ---")

# Obtenemos la ruta al mejor modelo de la Fase 1
# La variable results.save_dir contiene la ruta a la carpeta de la ejecución
path_fase1_best = os.path.join(results_fase1.save_dir, "weights/best.pt")
print(f"Mejor modelo de la Fase 1 guardado en: {path_fase1_best}")
"""

# ================================================================================
# FASE 2: AJUSTE FINO DE TODO EL MODELO (CON LEARNING RATE BAJO)
# ================================================================================
"""
print("\n--- INICIANDO FASE 2: AJUSTE FINO COMPLETO (LEARNING RATE BAJO) ---")

# Cargar el modelo resultante de la Fase 1
#model_fase2 = YOLO(os.path.join(runs_dir, "Supermodelo_Fase1_Head2", "weights", "best.pt"))

# Ahora entrenamos todo el modelo (sin 'freeze') pero con una tasa de aprendizaje muy baja
results_fase2 = model_fase2.train(
    data=data_yaml_path,
    epochs=150,
    patience=30,
    batch=24,
    imgsz=640,
    name="Supermodelo_Fase2_Final",
    project=runs_dir,
    augment=True,  # Mantenemos las aumentaciones que ya tenías
    mixup=0.1,
    hsv_s=0.9,
    lr0=0.0005,  # ¡Parámetro clave! Tasa de aprendizaje extremadamente baja.
    optimizer="AdamW",  # ¡NUEVO! Forzamos el optimizador y desactivamos el modo 'auto'
)

print("\n--- ¡ENTRENAMIENTO POR FASES COMPLETADO! ---")
path_fase2_best = os.path.join(results_fase2.save_dir, "weights/best.pt")
print(f"El 'Supermodelo' final está listo en: {path_fase2_best}")
"""

# ================================================================================
# ENTRENAMIENTO COMPLETO DESDE VARIABLES
# ================================================================================
print("\nIniciando entrenamiento del modelo...")

results = model.train(
    data=data_yaml_path,  # Ruta a tu archivo de configuración del dataset
    epochs=execution_epochs,  # Número de épocas de entrenamiento (ajustable)
    imgsz=execution_image_resize,  # Tamaño de la imagen de entrada (640×640 es un buen inicio)
    batch=execution_batch,  # Tamaño del batch (ajustable, depende de tu RAM/VRAM)
```

```
    name=execution_name,  # Nombre para esta ejecución de entrenamiento
    project=runs_dir,  # Directorio raíz para guardar los resultados
    # workers=os.cpu_count() // 2   # Descomentar para usar la mitad de los núcleos de CPU para carga de da
    # --- Control del Entrenamiento ---
    patience=early_stop,  # Parada temprana si no mejora en 25 épocas
    # --- Parámetros de Aumentación ---
    augment=True,  # Nos aseguramos de que la aumentación general esté activa
    # degrees=10.0,  # Rotación de hasta 10 grados
    # translate=0.2,  # Traslación de hasta un 20%
    # scale=0.8,  # Escalado de hasta un 80%
    # shear=5.0,  # Inclinación de hasta 5 grados
    mixup=0.1,  # Activar MixUp con un 10% de probabilidad
    hsv_s=0.9,  # Aumentar la variación de saturación
    # fliplr=0.5,  # Mantener el volteo horizontal
    # lr0=tasa_aprendizaje,
    # freeze=11,  # ¡Parámetro clave! Congela las capas del backbone.
)

print("\n¡Entrenamiento completado!")
print(f"Los resultados se guardaron en: {os.path.join(runs_dir, execution_name)}")
print("Puedes revisar los gráficos de entrenamiento y las métricas allí.")
```

detect_balls/encontrar_mejor_epoca.py

```
# encontrar_mejor_epoca.py
import pandas as pd

# --- CONFIGURACIÓN ---
# Ruta al archivo results.csv de tu último entrenamiento
RUTA_RESULTS_CSV = "./detect_balls/runs/Modelo_Hibrido_v1/results.csv"
# --- FIN DE LA CONFIGURACIÓN ---


def encontrar_mejor_epoca():
    try:
        # Cargar los resultados en un DataFrame de pandas
        df = pd.read_csv(RUTA_RESULTS_CSV)

        # Limpiar los nombres de las columnas por si tienen espacios extra
        df.columns = df.columns.str.strip()

        # Nombre de la métrica que define el "mejor" modelo
        metrica_clave = "metrics/mAP50-95(B)"

        # Encontrar el índice de la fila con el valor máximo en esa métrica
        indice_mejor_epoca = df[metrica_clave].idxmax()

        # Obtener toda la información de esa fila (de esa época)
        mejor_epoca_info = df.loc[indice_mejor_epoca]

        # El número de la época es el índice + 1
        # mejor_epoca_num = int(mejor_epoca_info["epoch"]) + 1
```

```
        mejor_epoca_num = int(mejor_epoca_info["epoch"])
        mejor_map50_95 = mejor_epoca_info[metrica_clave]
        mejor_map50 = mejor_epoca_info["metrics/mAP50(B)"]

        print("\n--- Análisis del Mejor Modelo (`best.pt`) ---")
        print(f"El mejor rendimiento se alcanzó en la ÉPOCA: {mejor_epoca_num}")
        print(f"  - mAP50-95 (estricto): {mejor_map50_95:.4f}")
        print(f"  - mAP50 (estándar):   {mejor_map50:.4f}")
        print("\nEl archivo 'best.pt' corresponde a los pesos guardados en esa época.")

    except FileNotFoundError:
        print(f"Error: No se encontró el archivo en la ruta '{RUTA_RESULTS_CSV}'.")
        print("Asegúrate de que la ruta a tu carpeta de 'runs' es correcta.")
    except KeyError:
        print(f"Error: No se encontró la columna '{metrica_clave}' en el archivo .csv.")
        print("Asegúrate de que el nombre de la métrica es correcto.")


if __name__ == "__main__":
    encontrar_mejor_epoca()
```

detect_balls/test_model.py

```
import os

import cv2
import matplotlib.patches as patches
import matplotlib.pyplot as plt
from ultralytics import YOLO

# --- 1. Definir rutas y parámetros ---
base_project_dir = os.path.join(".", "detect_balls")

train_version = "Supermodelo_Fase2_Final"
# train_version = "detect_balls_v"
version_run = ""
nombre_epoch = "Supermodelo_Fase2_Final"
path_runs = "runs"
path_test_results = "test_results"

# Ruta al modelo entrenado
model_path = os.path.join(
    base_project_dir,
    path_runs,
    train_version + version_run,
    "weights",
    "best.pt",
)

print("modelo: ", model_path)
```

```
"""
model_path = os.path.join(
    base_project_dir,
    "models_custom",
    "supermodel.pt",
)
"""


# nombre_imagen_test_1 = "87_jpg.rf.d7116def1f52d243e1296a65c37b0c4c.jpg"
# nombre_imagen_test_1 = "412_jpg.rf.b505e28c8901b5bb4ad7c848f2a5d68b.jpg"
# nombre_imagen_test_1 = "833_jpg.rf.10091a238702573667ad5d0ce98610c8.jpg"
nombre_imagen_test_1 = "50_jpg.rf.8af48c542670796a51fe342700a543ab.jpg"
# nombre_imagen_test_1 = "148_jpg.rf.dace67876bd0a52dd413ea5225637e57.jpg"
# nombre_imagen_test_1 = "745_jpg.rf.6e4a996c29238c014b0bb9eb4b4e82b5.jpg"
# nombre_imagen_test_1 = "2025-06-15 20-03-49_aug_11.jpg"

"""
test_image_path_1 = os.path.join(
    base_project_dir,
    "data",
    "test",
    "images",
    nombre_imagen_test_1,
)
"""

test_image_path_1 = os.path.join(
    base_project_dir,
    "dataset_unificado_aumentado",
    "images",
    "valid",
    nombre_imagen_test_1,
)


nombre_imagen_test_2 = "test_pool_table_1.png"
nombre_imagen_test_2 = "test_pool_table_2.png"
nombre_imagen_test_2 = "test_pool_table_3.png"
nombre_imagen_test_2 = "test_pool_table_4.jpg"
# nombre_imagen_test_2 = "test_pool_table_5.jpg"

test_image_path_2 = os.path.join(
    base_project_dir,
    "tests",
    nombre_imagen_test_2,
)

nombre_imagen_test = nombre_imagen_test_1
test_image_path = test_image_path_1

# Clases de tu dataset (debe coincidir con la lista usada en custom_data.yaml y en prepare_yolo_dataset.py)
```

```
classes = [
    "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_black_8",
    "be_blue_10",
    "be_blue_2",
    "be_dred_15",
    "be_dred_7",
    "be_green_14",
    "be_green_6",
    "be_purple_13",
    "be_purple_5",
    "be_pink_4",
    "be_pink_12",
    "be_red_11",
    "be_red_3",
    "be_white",
    "be_yellow_1",
    "be_yellow_9",
]

# --- 2. Cargar el modelo ---
try:
    model = YOLO(model_path)
    print(f"Modelo cargado desde: {model_path}")
except Exception as e:
    print(f"Error al cargar el modelo: {e}")
    print(
        "Asegúrate de que la ruta al archivo 'last.pt' sea correcta y que el entrenamiento se haya guardado."
    )
    exit()

# --- 3. Realizar la inferencia ---
print(f"Realizando inferencia en: {test_image_path}")
# La función predict devuelve un objeto Results.
# verbose=False para no imprimir los detalles de inferencia en consola,
# conf=0.25 es el umbral de confianza para mostrar detecciones (ajústalo si detecta demasiado o muy poco)
results = model.predict(
    source=test_image_path, save=False, imgsz=1024, conf=0.4, verbose=True
```

```
)

# --- 4. Procesar y visualizar los resultados ---
if results:
    # Solo tomamos el primer resultado (ya que estamos procesando una sola imagen)
    result = results[0]

    # Cargar la imagen original para visualización
    image = cv2.imread(test_image_path)
    image = cv2.cvtColor(
        image, cv2.COLOR_BGR2RGB
    )  # OpenCV carga BGR, matplotlib espera RGB

    fig, ax = plt.subplots(1, figsize=(12, 8))
    ax.imshow(image)
    ax.set_title("Detecciones del Modelo YOLO")

    # Extraer bounding boxes y clases
    for box in result.boxes:
        # box.xyxy: coordenadas en formato [x1, y1, x2, y2]
        x1, y1, x2, y2 = map(int, box.xyxy[0])

        # box.conf: confianza de la detección
        confidence = box.conf[0]

        # box.cls: ID de la clase detectada
        class_id = int(box.cls[0])
        class_name = (
            classes[class_id]
            if class_id < len(classes)
            else f"Unknown_Class_{class_id}"
        )

        # Dibujar el bounding box
        rect = patches.Rectangle(
            (x1, y1), x2 - x1, y2 - y1, linewidth=2, edgecolor="red", facecolor="none"
        )
        ax.add_patch(rect)

        # Poner la etiqueta de la clase y la confianza
        # Usamos coordenadas relativas para el texto para que no se salga de la imagen
        text_x = x1
        text_y = (
            y1 - 10 if y1 - 10 > 0 else y1 + 20
        )  # Ajusta la posición del texto para que sea visible
        ax.text(
            text_x,
            text_y,
            f"{class_name} {confidence:.2f}",
            color="red",
            fontsize=10,
            bbox=dict(facecolor="white", alpha=0.7, edgecolor="none", pad=1),
        )
```

```
        # Imprimir coordenadas y clase por consola
        print(
            f"Detectado: {class_name} (Confianza: {confidence:.2f}) - Coordenadas: xmin={x1}, ymin={y1}, xmax
        )
        print(f"   Centro: ({int((x1+x2)/2)}, {int((y1+y2)/2)})")

    plt.axis("off")  # Ocultar ejes

    output_image_path = os.path.join(
        base_project_dir, path_test_results, nombre_epoch + nombre_imagen_test
    )  # O el nombre que prefieras

    plt.savefig(output_image_path, bbox_inches="tight", pad_inches=0)
    print(f"\nImagen con detecciones guardada en: {output_image_path}")

    plt.show()

    # Si quisieras las coordenadas normalizadas (0 a 1) para un uso posterior:
    # Puedes usar result.boxes.xyxyn para obtenerlas directamente.
    # Por ejemplo:
    # normalized_boxes = result.boxes.xyxyn
    # for i, bbox_norm in enumerate(normalized_boxes):
    #     cls_id = int(result.boxes.cls[i])
    #     conf = result.boxes.conf[i]
    #     center_x_norm = (bbox_norm[0] + bbox_norm[2]) / 2
    #     center_y_norm = (bbox_norm[1] + bbox_norm[3]) / 2
    #     width_norm = bbox_norm[2] - bbox_norm[0]
    #     height_norm = bbox_norm[3] - bbox_norm[1]
    #     print(f"Normalizado: {classes[cls_id]} (Conf: {conf:.2f}) - Centro({center_x_norm:.4f}, {center_y_norm:

else:
    print("No se detectaron objetos en la imagen de prueba.")
```

detect_balls/inferencia_con_postproceso.py

```
# inferencia_con_razonamiento_final.py
import os

import cv2
import matplotlib.patches as patches
import matplotlib.pyplot as plt
from ultralytics import YOLO

# --- 1. CONFIGURACIÓN ---
base_project_dir = os.path.join(".", "detect_balls")
train_version = "Modelo_Hibrido_v1"
runs_dir = os.path.join(base_project_dir, "runs")
model_path = os.path.join(runs_dir, train_version, "weights", "best.pt")
```

```
TEST_IMAGES_DIR = os.path.join(base_project_dir, "tests", "Classic2")
output_dir = os.path.join(
    base_project_dir, "tests", "results_razonamiento_final_Classic2"
)

CONFIDENCE_THRESHOLD = 0.4

# Lista de clases del modelo HÍBRIDO (25 clases)
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",
]

# --- BASE DE CONOCIMIENTO PARA EL RAZONAMIENTO ---

DELATORES_BE = {
    "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_CLASSIC = {
    "blue_10",
    "dred_15",
    "green_14",
```

```
    "orange_13",
    "orange_5",
    # "purple_12",
    # "purple_4",
    "red_11",
    "yellow_9",
}

# Tu mapa de corrección, la "inteligencia experta"
MAPA_CORRECCION_CONTEXTUAL = {
    # Si el contexto es CLASSIC pero el modelo predice una bola BE...
    "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",
    # Si el contexto es BLACK_EDITION pero el modelo predice una bola Classic...
    "purple_4": "be_purple_5",
    "purple_12": "be_purple_13",
    "yellow_9": "be_yellow_9",
}

# --- FIN DE LA CONFIGURACIÓN ---


def post_procesar_con_razonamiento(detecciones_yolo):
    puntuacion_be = sum(
        d["conf"] for d in detecciones_yolo if d["clase"] in DELATORES_BE
    )
    puntuacion_classic = sum(
        d["conf"] for d in detecciones_yolo if d["clase"] in DELATORES_CLASSIC
    )

    contexto = "black_edition" if puntuacion_be > puntuacion_classic else "classic"
    print(
        f"\n  > Puntuación 'classic': {puntuacion_classic:.2f} | Puntuación 'be': {puntuacion_be:.2f} → Contexto
    )

    detecciones_finales = []
    for deteccion in detecciones_yolo:
        etiqueta_yolo = deteccion["clase"]
        etiqueta_corregida = etiqueta_yolo

        # 1. Aplicar corrección contextual si es necesario
        if contexto == "classic" and etiqueta_yolo in DELATORES_BE:
            etiqueta_corregida = MAPA_CORRECCION_CONTEXTUAL.get(
                etiqueta_yolo, etiqueta_yolo
            )
            print(
```

```
                f"    - CORRECCIÓN (Contexto Classic): YOLO dijo '{etiqueta_yolo}', se corrige a '{etiqueta_corregid
            )
        elif contexto == "black_edition" and etiqueta_yolo in DELATORES_CLASSIC:
            etiqueta_corregida = MAPA_CORRECCION_CONTEXTUAL.get(
                etiqueta_yolo, etiqueta_yolo
            )
            print(
                f"    - CORRECCIÓN (Contexto BE): YOLO dijo '{etiqueta_yolo}', se corrige a '{etiqueta_corregida}'."
            )

        # 2. Limpiar el prefijo 'be_' para la etiqueta final
        etiqueta_simple = etiqueta_corregida.replace("be_", "")

        deteccion["etiqueta_final"] = etiqueta_simple
        detecciones_finales.append(deteccion)

    return detecciones_finales


def main():
    os.makedirs(output_dir, exist_ok=True)
    model = YOLO(model_path)
    print(f"Modelo cargado desde: {model_path}")

    image_files = [
        f
        for f in os.listdir(TEST_IMAGES_DIR)
        if f.lower().endswith((".png", ".jpg", ".jpeg"))
    ]
    print(f"\nIniciando inferencia en {len(image_files)} imágenes...")

    for image_filename in image_files:
        test_image_path = os.path.join(TEST_IMAGES_DIR, image_filename)
        print(f"\n--- Procesando: {image_filename} ---")
        results = model.predict(
            source=test_image_path,
            save=False,
            imgsz=1024,
            conf=CONFIDENCE_THRESHOLD,
            verbose=False,
        )
        result = results[0]

        if not result.boxes:
            continue

        detecciones_yolo = [
            {
                "clase": CLASES_HIBRIDAS[int(box.cls[0])],
                "conf": box.conf[0],
                "box_xyxy": box.xyxy[0],
            }
            for box in result.boxes
```

```
        ]

        detecciones_finales = post_procesar_con_razonamiento(detecciones_yolo)

        print("  > Resultados Finales para la Imagen:")
        image = cv2.imread(test_image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        fig, ax = plt.subplots(1, figsize=(12, 8))
        ax.imshow(image)
        ax.set_title(f"Detecciones Corregidas en {image_filename}")

        for deteccion in detecciones_finales:
            x1, y1, x2, y2 = map(int, deteccion["box_xyxy"])
            etiqueta_final = deteccion["etiqueta_final"]
            confianza = deteccion["conf"]

            print(f"    - Bola: {etiqueta_final} (Confianza original: {confianza:.2f})")

            rect = patches.Rectangle(
                (x1, y1),
                x2 - x1,
                y2 - y1,
                linewidth=2,
                edgecolor="cyan",
                facecolor="none",
            )
            ax.add_patch(rect)
            text_y = y1 - 10 if y1 > 10 else y1 + 20
            ax.text(
                x1,
                text_y,
                f"{etiqueta_final} {confianza:.2f}",
                color="black",
                fontsize=10,
                bbox=dict(facecolor="cyan", alpha=0.8),
            )

        plt.axis("off")
        output_image_path = os.path.join(output_dir, f"resultado_{image_filename}")
        plt.savefig(output_image_path, bbox_inches="tight", pad_inches=0)
        plt.close(fig)

    print("\n\n--- Proceso de inferencia final completado. ---")


if __name__ == "__main__":
    main()
```

detect_balls/generate_graphs.py

```
# prompt: vuelve a generar las graficas regogiendo los datos del archivo "data.csv", creando una grafica dife

#!pip install pandas matplotlib seaborn

import io
import os

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Read the CSV file into a pandas DataFrame
base_project_dir = os.path.join(".", "detect_balls")
csv_file = os.path.join(base_project_dir, "runs", "Modelo_Hibrido_v1", "results.csv")
df = pd.read_csv(csv_file)

# Plotting training losses
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x="epoch", y="train/box_loss", label="Train Box Loss")
sns.lineplot(data=df, x="epoch", y="train/cls_loss", label="Train Class Loss")
sns.lineplot(data=df, x="epoch", y="train/dfl_loss", label="Train DFL Loss")
plt.title("Training Losses vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss Value")
plt.legend()
# plt.show()
# Guarda la figura en un archivo PNG en la misma carpeta del script
plt.savefig(os.path.join(base_project_dir, "graph_results", "graph_train.png"))
# Es una buena práctica cerrar la figura para liberar memoria
plt.close()

# Plotting metrics
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x="epoch", y="metrics/precision(B)", label="Precision (B)")
sns.lineplot(data=df, x="epoch", y="metrics/recall(B)", label="Recall (B)")
sns.lineplot(data=df, x="epoch", y="metrics/mAP50(B)", label="mAP50 (B)")
sns.lineplot(data=df, x="epoch", y="metrics/mAP50-95(B)", label="mAP50-95 (B)")
plt.title("Metrics vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Metric Value")
plt.legend()
# plt.show()
# Guarda la figura en un archivo PNG en la misma carpeta del script
plt.savefig(os.path.join(base_project_dir, "graph_results", "graph_metrics.png"))
# Es una buena práctica cerrar la figura para liberar memoria
plt.close()

# Plotting validation losses
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x="epoch", y="val/box_loss", label="Validation Box Loss")
sns.lineplot(data=df, x="epoch", y="val/cls_loss", label="Validation Class Loss")
sns.lineplot(data=df, x="epoch", y="val/dfl_loss", label="Validation DFL Loss")
plt.title("Validation Losses vs. Epoch")
```

```
plt.xlabel("Epoch")
plt.ylabel("Loss Value")
plt.legend()
# plt.show()
# Guarda la figura en un archivo PNG en la misma carpeta del script
plt.savefig(os.path.join(base_project_dir, "graph_results", "graph_validation.png"))
# Es una buena práctica cerrar la figura para liberar memoria
plt.close()

# Plotting learning rates
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x="epoch", y="lr/pg0", label="LR pg0")
sns.lineplot(data=df, x="epoch", y="lr/pg1", label="LR pg1")
sns.lineplot(data=df, x="epoch", y="lr/pg2", label="LR pg2")
plt.title("Learning Rate vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Learning Rate")
plt.legend()
# plt.show()
# Guarda la figura en un archivo PNG en la misma carpeta del script
plt.savefig(os.path.join(base_project_dir, "graph_results", "graph_learning_rates.png"))
# Es una buena práctica cerrar la figura para liberar memoria
plt.close()
```

detect_balls/pre-labeling.py

```
# pre_etiquetado_definitivo.py
import os

from ultralytics import YOLO

# --- 1. CONFIGURACIÓN ---

# Ruta a tu mejor modelo entrenado hasta ahora
MODELO_PATH = "./detect_balls/runs/detect_balls_v12/weights/best.pt"

# Carpeta donde tienes TODAS tus nuevas imágenes "black edition" sin etiquetar
IMAGENES_NUEVAS_DIR = "./detect_balls/datasets/black_edition_raw/images/"

# Carpeta donde se guardarán las etiquetas generadas por la IA
ETIQUETAS_GENERADAS_DIR = "./detect_balls/datasets/black_edition_raw/labels/"

# Umbral de confianza
UMBRAL_CONFIANZA = 0.25

# ¡¡CRUCIAL!! Esta lista y su orden deben ser IDÉNTICOS a los del `data.yaml`
# con el que se entrenó el modelo que estás usando en MODELO_PATH.
CLASSES = [
    "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",
]

CLASES_MAESTRA = [
    # --- Set Original (IDs 0-15) ---
    "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",
    # --- Set Black Edition (IDs 16-31) ---
    "be_black_8",
    "be_blue_10",
    "be_blue_2",
    "be_dred_15",
    "be_dred_7",
    "be_green_14",
    "be_green_6",
    "be_purple_13",
    "be_purple_5",
    "be_pink_12",
    "be_pink_4",
    "be_red_11",
    "be_red_3",
    "be_white",
    "be_yellow_1",
    "be_yellow_9",
]
```

```
# --- FIN DE LA CONFIGURACIÓN ---

os.makedirs(ETIQUETAS_GENERADAS_DIR, exist_ok=True)

try:
    model = YOLO(MODELO_PATH)
    print(f"Modelo cargado exitosamente desde: {MODELO_PATH}")
except Exception as e:
    print(f"Error al cargar el modelo: {e}")
    exit()

try:
    lista_imagenes = [
        f
        for f in os.listdir(IMAGENES_NUEVAS_DIR)
        if f.lower().endswith((".png", ".jpg", ".jpeg"))
    ]
    if not lista_imagenes:
        print(f"No se encontraron imágenes en: {IMAGENES_NUEVAS_DIR}")
        exit()
    print(f"Se encontraron {len(lista_imagenes)} imágenes para pre-etiquetar.")
except FileNotFoundError:
    print(f"Error: El directorio de imágenes no existe: {IMAGENES_NUEVAS_DIR}")
    exit()

# Procesar cada imagen
for nombre_imagen in lista_imagenes:
    ruta_imagen = os.path.join(IMAGENES_NUEVAS_DIR, nombre_imagen)
    print(f"\nProcesando imagen: {nombre_imagen}...")

    results = model.predict(ruta_imagen, conf=UMBRAL_CONFIANZA, verbose=False)
    result = results[0]

    nombre_base = os.path.splitext(nombre_imagen)[0]
    ruta_etiqueta = os.path.join(ETIQUETAS_GENERADAS_DIR, f"{nombre_base}.txt")

    with open(ruta_etiqueta, "w") as f_etiqueta:
        if result.boxes:
            print(f"  > Se detectaron {len(result.boxes)} objetos:")
            for box in result.boxes:
                coords_xywhn = box.xywhn[0]
                id_clase = int(box.cls[0])
                confianza = box.conf[0]

                nombre_clase = (
                    CLASSES[id_clase]
                    if id_clase < len(CLASSES)
                    else f"ID_DESCONOCIDO_{id_clase}"
                )
                print(
                    f"    - Propuesta: {nombre_clase} (ID: {id_clase}, Conf: {confianza:.2f})"
                )
```

```
            linea = f"{id_clase} {coords_xywhn[0]:.6f} {coords_xywhn[1]:.6f} {coords_xywhn[2]:.6f} {coords_xy
            f_etiqueta.write(linea)
    else:
        print("  > No se detectó ningún objeto con la confianza suficiente.")

print("\n¡Proceso de pre-etiquetado completado!")
print(
    f"Las etiquetas generadas (con los IDs correctos) están en: {ETIQUETAS_GENERADAS_DIR}"
)
```

detect_balls/traducir_labels_a_schema_hibrido.py

```
# traducir_labels_a_schema_hibrido.py
import os
import shutil

# --- CONFIGURACIÓN ---

# Directorio con las etiquetas del dataset unificado (el de 32 clases)
LABELS_ORIGINALES_DIR = "detect_balls/dataset_unificado_aumentado/labels/"
# Directorio de salida para las nuevas etiquetas "híbridas"
LABELS_HIBRIDAS_DIR = "detect_balls/dataset_hibrido/labels/"

# Tu lista MAESTRA ORIGINAL con la que se generaron las etiquetas (32 clases)
CLASES_MAESTRA_ORIGINAL = [
    "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_black_8",
    "be_blue_10",
    "be_blue_2",
    "be_dred_15",
    "be_dred_7",
    "be_green_14",
    "be_green_6",
    "be_purple_13",
    "be_purple_5",
```

```
    "be_pink_4",
    "be_pink_12",
    "be_red_11",
    "be_red_3",
    "be_white",
    "be_yellow_1",
    "be_yellow_9",
]

# Tu NUEVA lista HÍBRIDA de 25 clases
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",
]
# --- FIN DE LA CONFIGURACIÓN ---


def traducir_etiquetas_hibridas():
    id_a_nombre_original = {i: name for i, name in enumerate(CLASES_MAESTRA_ORIGINAL)}
    nombre_a_id_hibrido = {name: i for i, name in enumerate(CLASES_HIBRIDAS)}

    # Mapeo de lógica para unificar clases
    # La clave es el nombre original, el valor es el nombre en el nuevo esquema
    mapa_traduccion = {
        # Bolas que son únicas en el set BE
        "be_blue_10": "be_blue_10",
        "be_dred_15": "be_dred_15",
        "be_green_14": "be_green_14",
        "be_purple_13": "be_purple_13",
        "be_purple_5": "be_purple_5",
```

```
    "be_pink_4": "be_pink_4",
    "be_pink_12": "be_pink_12",
    "be_red_11": "be_red_11",
    "be_yellow_9": "be_yellow_9",
    # Bolas que son únicas en el set Clásico (las rayadas blancas)
    "blue_10": "blue_10",
    "dred_15": "dred_15",
    "green_14": "green_14",
    "orange_13": "orange_13",
    "purple_12": "purple_12",
    "red_11": "red_11",
    "yellow_9": "yellow_9",
    # Bolas sólidas que se unifican (el prefijo 'be_' desaparece)
    "black_8": "black_8",
    "be_black_8": "black_8",
    "blue_2": "blue_2",
    "be_blue_2": "blue_2",
    "dred_7": "dred_7",
    "be_dred_7": "dred_7",
    "green_6": "green_6",
    "be_green_6": "green_6",
    "orange_5": "orange_5",  # No hay 'be_orange_5', pero lo mantenemos por coherencia
    "purple_4": "purple_4",  # No hay 'be_purple_4' (es rosa), pero lo mantenemos
    "red_3": "red_3",
    "be_red_3": "red_3",
    "white": "white",
    "be_white": "white",
    "yellow_1": "yellow_1",
    "be_yellow_1": "yellow_1",
}

for split in ["train", "valid", "test"]:
    print(f"Traduciendo etiquetas de: {split}...")
    dir_origen = os.path.join(LABELS_ORIGINALES_DIR, split)
    dir_destino = os.path.join(LABELS_HIBRIDAS_DIR, split)

    for filename in os.listdir(dir_origen):
        with open(os.path.join(dir_origen, filename), "r") as f_in, open(
            os.path.join(dir_destino, filename), "w"
        ) as f_out:
            for line in f_in:
                parts = line.strip().split()
                # id_original = int(parts[0])
                id_original = int(float(parts[0]))
                coords = " ".join(parts[1:])

                nombre_original = id_a_nombre_original[id_original]
                nombre_hibrido = mapa_traduccion.get(nombre_original)

                if nombre_hibrido and nombre_hibrido in nombre_a_id_hibrido:
                    id_hibrido = nombre_a_id_hibrido[nombre_hibrido]
                    f_out.write(f"{id_hibrido} {coords}\n")
                else:
```

```
                print(
                    f"ADVERTENCIA: No se pudo traducir la clase '{nombre_original}'"
                )

    print("\n¡Traducción de etiquetas al esquema híbrido completada!")


if __name__ == "__main__":
    traducir_etiquetas_hibridas()
```

detect_balls/inferencia_con_postproceso_debug.py

```
# inferencia_con_postproceso_debug.py
import os

import cv2
import matplotlib.patches as patches
import matplotlib.pyplot as plt
from ultralytics import YOLO

# --- 1. CONFIGURACIÓN (Mantén la tuya) ---
base_project_dir = os.path.join(".", "detect_balls")
train_version = "Modelo_Hibrido_v1"
runs_dir = os.path.join(base_project_dir, "runs")
model_path = os.path.join(runs_dir, train_version, "weights", "best.pt")

# TEST_IMAGES_DIR = os.path.join(base_project_dir, "tests_classic_set")
TEST_IMAGES_DIR = os.path.join(base_project_dir, "tests", "Mix")
# output_dir = os.path.join(base_project_dir, "results_final_con_postproceso")
output_dir = os.path.join(base_project_dir, "tests", "results_final_con_postproceso")

CONFIDENCE_THRESHOLD = 0.4

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",
]
CLASES_COMPARTIDAS = [
    "black_8",
    "blue_2",
    "dred_7",
    "green_6",
    "red_3",
    "white",
    "yellow_1",
]
DELATORES_BE = [
    "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",
]

# --- FIN DE LA CONFIGURACIÓN ---


def post_procesar_detecciones(detecciones_yolo):
    nombres_clases_detectadas = [d["clase"] for d in detecciones_yolo]

    # 1. Detectar el contexto
    contexto = "classic"
    delator_encontrado = None
    for delator in DELATORES_BE:
        if delator in nombres_clases_detectadas:
            contexto = "black_edition"
            delator_encontrado = delator  # Guardamos qué delator activó el cambio
            break

    print(f"\n  > Iniciando Post-Procesamiento...")
    if delator_encontrado:
        print(
            f"  > Contexto detectado: '{contexto}' (activado por la detección de '{delator_encontrado}')"
        )
    else:
        print(f"  > Contexto detectado: '{contexto}' (no se encontraron 'delatores')")
```

```
    # 2. Refinar etiquetas
    detecciones_refinadas = []
    for deteccion in detecciones_yolo:
        etiqueta_original = deteccion["clase"]
        etiqueta_final = etiqueta_original

        if contexto == "black_edition" and etiqueta_original in CLASES_COMPARTIDAS:
            etiqueta_final = "be_" + etiqueta_original

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

        # --- SALIDA DE DEPURACIÓN ---
        print(
            f"    - Predicción YOLO: '{etiqueta_original}' → Etiqueta Final: '{etiqueta_final}'"
        )

    return detecciones_refinadas


def inferencia_final_debug():
    os.makedirs(output_dir, exist_ok=True)
    model = YOLO(model_path)
    print(f"Modelo cargado desde: {model_path}")

    image_files = [
        f
        for f in os.listdir(TEST_IMAGES_DIR)
        if f.lower().endswith((".png", ".jpg", ".jpeg"))
    ]
    print(f"\nIniciando inferencia en {len(image_files)} imágenes...")

    for image_filename in image_files:
        test_image_path = os.path.join(TEST_IMAGES_DIR, image_filename)
        print(f"\n--- Procesando: {image_filename} ---")

        results = model.predict(
            source=test_image_path,
            save=False,
            imgsz=1024,
            conf=CONFIDENCE_THRESHOLD,
            verbose=False,
        )
        result = results[0]

        if not result.boxes:
            print("No se detectaron objetos en esta imagen.")
            continue

        detecciones_yolo = []
        print("  > Detecciones del Modelo (salida en bruto):")
        for box in result.boxes:
```

```
            class_id = int(box.cls[0])
            clase_detectada = CLASES_HIBRIDAS[class_id]
            print(
                f"    - Objeto detectado como: '{clase_detectada}' (Conf: {box.conf[0]:.2f})"
            )
            detecciones_yolo.append(
                {"clase": clase_detectada, "conf": box.conf[0], "box_xyxy": box.xyxy[0]}
            )

        detecciones_finales = post_procesar_detecciones(detecciones_yolo)

        # --- Visualización con las etiquetas finales (sin cambios aquí) ---
        image = cv2.imread(test_image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        fig, ax = plt.subplots(1, figsize=(12, 8))
        ax.imshow(image)
        ax.set_title(f"Detecciones Finales en {image_filename}")

        for deteccion in detecciones_finales:
            x1, y1, x2, y2 = map(int, deteccion["box_xyxy"])
            etiqueta_a_mostrar = deteccion["etiqueta_final"]
            confianza = deteccion["conf"]

            rect = patches.Rectangle(
                (x1, y1),
                x2 - x1,
                y2 - y1,
                linewidth=2,
                edgecolor="lime",
                facecolor="none",
            )
            ax.add_patch(rect)

            text_y = y1 - 10 if y1 > 10 else y1 + 20
            ax.text(
                x1,
                text_y,
                f"{etiqueta_a_mostrar} {confianza:.2f}",
                color="black",
                fontsize=10,
                bbox=dict(facecolor="lime", alpha=0.8),
            )

        plt.axis("off")
        output_image_path = os.path.join(output_dir, f"resultado_{image_filename}")
        plt.savefig(output_image_path, bbox_inches="tight", pad_inches=0)
        plt.close(fig)

    print("\n\n--- Proceso de inferencia final completado. ---")


if __name__ == "__main__":
    inferencia_final_debug()
```

detect_balls/test_model_folder.py

```
# test_modelo_bucle.py
import os

import cv2
import matplotlib.patches as patches
import matplotlib.pyplot as plt
from ultralytics import YOLO

# --- 1. CONFIGURACIÓN ---
# Mantén todas las variables que puedas querer cambiar aquí arriba

# --- Rutas del Proyecto ---
base_project_dir = os.path.join(".", "detect_balls")
train_version = "Supermodelo_Fase2_Final"
train_version = "Modelo_Hibrido_v1"

runs_dir = os.path.join(base_project_dir, "runs")

# --- ¡NUEVO! Define aquí la carpeta con tus imágenes de prueba ---
# Debes crear esta carpeta y colocar dentro todas las imágenes que quieras probar.
TEST_IMAGES_DIR = os.path.join(base_project_dir, "tests", "Classic")
TEST_IMAGES_DIR = os.path.join(base_project_dir, "tests", "Black")
TEST_IMAGES_DIR = os.path.join(base_project_dir, "tests", "Mix")
TEST_IMAGES_DIR = os.path.join(base_project_dir, "tests", "Classic2")

# Carpeta donde se guardarán los resultados de este script
output_dir = os.path.join(base_project_dir, "tests", "results_classic")
output_dir = os.path.join(base_project_dir, "tests", "results_black")
output_dir = os.path.join(base_project_dir, "tests", "results_mix")
output_dir = os.path.join(base_project_dir, "tests", "results_classic2")

# --- Ruta al Modelo ---
# Apunta al mejor modelo de tu entrenamiento final
model_path = os.path.join(
    runs_dir,
    train_version,
    "weights",
    "best.pt",
)

# --- Parámetros de Inferencia ---
CONFIDENCE_THRESHOLD = 0.4  # Umbral de confianza (ajústalo según necesites)

# --- Lista de Clases Maestra ---
# Asegúrate de que esta lista sea la definitiva y correcta para tu modelo
classes = [
    "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_black_8",
    "be_blue_10",
    "be_blue_2",
    "be_dred_15",
    "be_dred_7",
    "be_green_14",
    "be_green_6",
    "be_purple_13",
    "be_purple_5",
    "be_pink_4",
    "be_pink_12",
    "be_red_11",
    "be_red_3",
    "be_white",
    "be_yellow_1",
    "be_yellow_9",
]

classes = [
    "black_8",  # solida negra 8, valida para 'black_8' y 'be_black_8'
    "blue_10",  # rayada azul y blanca 10, valida para 'blue_10'
    "blue_2",  # solida azul 2, valida para 'blue_2' y 'be_blue_2'
    "dred_15",  # rayada marron y blanca 15, valida para 'dred_15'
    "dred_7",  # solida marron 7, valida para 'dred_7' y 'be_dred_7'
    "green_14",  # rayada verde y blanca 14, valida para 'green_14'
    "green_6",  # solida verde 6, valida para 'green_6' y 'be_green_6'
    "orange_13",  # rayada naranja y blanca 13, valida para 'orange_13'
    "orange_5",  # solida naranja 5, valida para 'orange_5'
    "purple_12",  # rayada morada y blanca 12, valida para 'purple_12'
    "purple_4",  # solida morada 4, valida para 'purple_4'
    "red_11",  # rayada roja y blanca 11, valida para 'red_11'
    "red_3",  # solida roja 3, valida para 'red_3' y 'be_red_3'
    "white",  # solida blanca, valida para 'white' y 'be_white'
    "yellow_1",  # solida amarilla 1, valida para 'yellow_1' y 'be_yellow_1'
    "yellow_9",  # rayada amarilla y blanca 9, valida para 'yellow_9'
    "be_blue_10",  # rayada azul y negra 10, valida para 'be_blue_10'
    "be_dred_15",  # rayada morada y negra 15, valida para 'be_dred_15'
    "be_green_14",  # rayada verde y negra 14, valida para 'be_green_14'
    "be_purple_13",  # rayada morada y negra 13, valida para 'be_purple_13'
    "be_purple_5",  # solida morada 5, valida para 'be_purple_5'
```

```
    "be_pink_4",  # solida rosa 4, valida para 'be_pink_4'
    "be_pink_12",  # rayada rosa y negra 12, valida para 'be_pink_12'
    "be_red_11",  # rayada roja y negra 11, valida para 'be_red_11'
    "be_yellow_9",  # rayada amarilla y negra 9, valida para 'be_yellow_9'
]
# --- FIN DE LA CONFIGURACIÓN ---


def inferencia_en_lote():
    # Crear el directorio de salida si no existe
    os.makedirs(output_dir, exist_ok=True)

    # --- 2. Cargar el modelo (una sola vez) ---
    try:
        model = YOLO(model_path)
        print(f"Modelo cargado desde: {model_path}")
    except Exception as e:
        print(f"Error al cargar el modelo: {e}")
        return

    # --- 3. Bucle para procesar todas las imágenes ---
    image_files = [
        f
        for f in os.listdir(TEST_IMAGES_DIR)
        if f.lower().endswith((".png", ".jpg", ".jpeg"))
    ]

    if not image_files:
        print(f"No se encontraron imágenes en la carpeta: {TEST_IMAGES_DIR}")
        return

    print(f"\nIniciando inferencia en {len(image_files)} imágenes...")

    for image_filename in image_files:
        test_image_path = os.path.join(TEST_IMAGES_DIR, image_filename)
        print(f"\n--- Procesando: {image_filename} ---")

        # Realizar la inferencia en la imagen actual
        results = model.predict(
            source=test_image_path,
            save=False,
            imgsz=1024,
            conf=CONFIDENCE_THRESHOLD,
            verbose=False,
        )

        # --- 4. Procesar y visualizar los resultados (para cada imagen) ---
        result = results[0]
        if not result.boxes:
            print("No se detectaron objetos en esta imagen.")
            continue

        image = cv2.imread(test_image_path)
```

```
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        fig, ax = plt.subplots(1, figsize=(12, 8))
        ax.imshow(image)
        ax.set_title(f"Detecciones en {image_filename}")

        for box in result.boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            confidence = box.conf[0]
            class_id = int(box.cls[0])
            class_name = (
                classes[class_id]
                if class_id < len(classes)
                else f"ID_Desconocido_{class_id}"
            )

            # Dibujar caja
            rect = patches.Rectangle(
                (x1, y1),
                x2 - x1,
                y2 - y1,
                linewidth=2,
                edgecolor="cyan",
                facecolor="none",
            )
            ax.add_patch(rect)

            # Poner etiqueta
            text_y = y1 - 10 if y1 > 10 else y1 + 20
            ax.text(
                x1,
                text_y,
                f"{class_name} {confidence:.2f}",
                color="white",
                fontsize=10,
                bbox=dict(facecolor="black", alpha=0.7),
            )

            print(f"  Detectado: {class_name} (Confianza: {confidence:.2f})")

        plt.axis("off")

        # Guardar la imagen con un nombre único
        output_filename = f"resultado_{image_filename}"
        output_image_path = os.path.join(output_dir, output_filename)
        plt.savefig(output_image_path, bbox_inches="tight", pad_inches=0)
        print(f"  Imagen con detecciones guardada en: {output_image_path}")

        # Cerrar la figura para liberar memoria y evitar que se muestren todas al final
        plt.close(fig)

    print("\n\n--- Proceso de inferencia en bucle completado. ---")
```

```
if __name__ == "__main__":
    inferencia_en_lote()
```

detect_balls/prepare_yolo_dataset.py

```
import os
import shutil

import pandas as pd

# --- 1. Definir rutas y nombres de archivos ---
# Ruta base donde se encuentra la carpeta 'data'
base_project_dir = "."

# Directorios de origen de las imágenes y anotaciones CSV
data_base_dir = os.path.join(base_project_dir, "data", "BallsDataset")
train_src_dir = os.path.join(data_base_dir, "train")
test_src_dir = os.path.join(
    data_base_dir, "test"
)  # Usaremos test como 'val' si no hay una carpeta 'valid'
val_src_dir = os.path.join(
    data_base_dir, "valid"
)  # Si existe una carpeta 'valid' separada

# Directorios de destino para el formato YOLO
output_images_dir = os.path.join(base_project_dir, "data", "images")
output_labels_dir = os.path.join(base_project_dir, "data", "labels")

output_train_images_dir = os.path.join(output_images_dir, "train")
output_train_labels_dir = os.path.join(output_labels_dir, "train")
output_val_images_dir = os.path.join(output_images_dir, "val")
output_val_labels_dir = os.path.join(output_labels_dir, "val")
output_test_images_dir = os.path.join(
    output_images_dir, "test"
)  # Para el conjunto de test
output_test_labels_dir = os.path.join(
    output_labels_dir, "test"
)  # Para el conjunto de test

# Clases del dataset (ordenadas, esto define el class_id)
# Es CRÍTICO que el orden de estas clases sea CONSISTENTE y completo
classes = [
    "white",
    "yellow_1",
    "blue_2",
    "red_3",
    "purple_4",
    "orange_5",
    "green_6",
```

```
    "dred_7",
    "black_8",
    "yellow_9",
    "blue_10",
    "red_11",
    "purple_12",
    "orange_13",
    "green_14",
    "dred_15",
]

# Crear el mapeo de nombres de clase a IDs
class_to_id = {name: i for i, name in enumerate(classes)}


# --- 2. Función para procesar un conjunto de datos (train, val, test) ---
def process_dataset_split(src_dir, dest_images_dir, dest_labels_dir):
    """
    Procesa un directorio de origen de imágenes y CSV de anotaciones,
    copiando imágenes y generando archivos de anotación YOLO.
    """
    print(f"\nProcesando directorio: {src_dir}")
    # Crear directorios de destino si no existen
    os.makedirs(dest_images_dir, exist_ok=True)
    os.makedirs(dest_labels_dir, exist_ok=True)

    annotations_file_path = os.path.join(src_dir, "_annotations.csv")

    if not os.path.exists(annotations_file_path):
        print(
            f"Advertencia: No se encontró _annotations.csv en {src_dir}. Saltando este directorio."
        )
        return

    df_annotations = pd.read_csv(annotations_file_path)

    # Limpiar archivos .txt antiguos en el directorio de labels de destino
    for f in os.listdir(dest_labels_dir):
        if f.endswith(".txt"):
            os.remove(os.path.join(dest_labels_dir, f))

    for index, row in df_annotations.iterrows():
        img_filename = row["filename"]
        img_width = row["width"]
        img_height = row["height"]
        obj_class_name = row["class"]
        xmin = row["xmin"]
        ymin = row["ymin"]
        xmax = row["xmax"]
        ymax = row["ymax"]

        # Verificar si la clase existe en nuestro mapeo
        if obj_class_name not in class_to_id:
```

```
            print(
                f"Advertencia: Clase '{obj_class_name}' no encontrada en la lista de clases definidas. Saltando ano
            )
            continue

        # Copiar la imagen al directorio correspondiente (si no está ya allí)
        source_img_path = os.path.join(src_dir, img_filename)
        destination_img_path = os.path.join(dest_images_dir, img_filename)
        if not os.path.exists(destination_img_path):
            try:
                shutil.copy(source_img_path, destination_img_path)
            except FileNotFoundError:
                print(
                    f"Error: Imagen original no encontrada en {source_img_path}. Asegúrate de que las imágenes es
                )
                continue  # Si la imagen no existe, no podemos procesar su anotación.

        # Convertir coordenadas a formato YOLO
        obj_class_id = class_to_id[obj_class_name]
        center_x = (xmin + xmax) / 2 / img_width
        center_y = (ymin + ymax) / 2 / img_height
        bbox_width = (xmax - xmin) / img_width
        bbox_height = (ymax - ymin) / img_height

        # Abrir el archivo .txt de la etiqueta en modo 'a' (append) para añadir todas las anotaciones de la image
        label_filename = os.path.splitext(img_filename)[0] + ".txt"
        label_filepath = os.path.join(dest_labels_dir, label_filename)

        with open(label_filepath, "a") as f:
            f.write(
                f"{obj_class_id} {center_x:.6f} {center_y:.6f} {bbox_width:.6f} {bbox_height:.6f}\n"
            )

    print(f"Procesamiento de {src_dir} completado.")


# --- 3. Procesar cada conjunto de datos ---
process_dataset_split(train_src_dir, output_train_images_dir, output_train_labels_dir)
process_dataset_split(val_src_dir, output_val_images_dir, output_val_labels_dir)
process_dataset_split(test_src_dir, output_test_images_dir, output_test_labels_dir)

print("\nTodos los conjuntos de datos han sido procesados.")

# --- 4. Crear el archivo custom_data.yaml (importante para YOLO) ---
# Se asume que la estructura de salida es data/images/train, data/images/val, data/images/test
data_yaml_content = f"""
path: {os.path.abspath(os.path.join(base_project_dir, 'data'))} # Ruta absoluta a la carpeta 'data'
train: images/train # Ruta relativa a las imágenes de entrenamiento
val: images/val   # Ruta relativa a las imágenes de validación
test: images/test # Ruta relativa a las imágenes de prueba

# Número de clases
nc: {len(classes)}
```

```
# Nombres de las clases
names: {classes}
"""

with open(os.path.join(base_project_dir, "custom_data.yaml"), "w") as f:
    f.write(data_yaml_content)

print(
    f"Archivo custom_data.yaml generado en {os.path.join(base_project_dir, 'custom_data.yaml')}"
)

print("\n¡Preparación del dataset completada!")
print("Puedes continuar con el entrenamiento de tu modelo YOLO.")
```

detect_table/test_model_capture_coords.py

```
import cv2
import numpy as np
from ultralytics import YOLO

# ------------------------------------------------------------------------------
# PASO 0: CONFIGURACIÓN INICIAL
# ------------------------------------------------------------------------------

# Tu MATRIZ DE HOMOGRAFÍA (Calculada en el paso anterior)
# Asegúrate de que esta es la matriz correcta que calculaste para la imagen
# que vas a procesar o para una configuración de cámara similar.
H_matrix = np.array(
    [
        [9.01494445e-01, 1.69172272e-17, -8.56419723e01],
        [-8.24671937e-04, 9.16210522e-01, -8.23806031e01],
        [1.62899797e-06, 6.57494777e-06, 1.00000000e00],
    ]
)

# Ruta a tu modelo YOLO entrenado (el archivo .pt)
MODEL_PATH = "poolballs36.pt"  # Cambia esto
# Ejemplo: MODEL_PATH = 'runs/detect/train/weights/best.pt'

# Nombres de las clases: Deben estar en el MISMO ORDEN que usó tu modelo para entrenar.
# Generalmente este orden viene del archivo .yaml que usaste para el entrenamiento.
CLASS_NAMES = [
    "white",
    "blue_10",
    "dred_15",
    "black_8",
    "purple_12",
    "dred_7",
    "orange_13",
```

```
    "blue_2",
    "red_3",
    "green_6",
    "green_14",
    "red_11",
    "yellow_1",
    "orange_5",
    "yellow_9",
]  # ¡¡AJUSTA ESTA LISTA A TUS CLASES Y SU ORDEN CORRECTO!!

# Ruta a la imagen que quieres procesar
IMAGE_PATH = "tests/test_pool_table_1.png"  # Cambia esto
# Idealmente, usa la misma imagen para la cual seleccionaste las esquinas manualmente,
# o una tomada desde una perspectiva muy similar.

# ------------------------------------------------------------------------------
# PASO 1: CARGAR MODELO Y REALIZAR INFERENCIA
# ------------------------------------------------------------------------------
try:
    model = YOLO(MODEL_PATH)
except Exception as e:
    print(f"Error al cargar el modelo YOLO: {e}")
    exit()

# Cargar la imagen original con OpenCV
img_original_cv = cv2.imread(IMAGE_PATH)
if img_original_cv is None:
    print(f"Error: No se pudo cargar la imagen {IMAGE_PATH}")
    exit()

# Realizar la inferencia con el modelo YOLO
print(f"Realizando inferencia en {IMAGE_PATH}...")
results = model(img_original_cv)  # La inferencia devuelve una lista de objetos Results

# Lista para guardar la información de las bolas con coordenadas transformadas
detected_balls_on_table_plane = []

# ------------------------------------------------------------------------------
# PASO 2: PROCESAR DETECCIONES Y TRANSFORMAR COORDENADAS
# ------------------------------------------------------------------------------
# 'results' es una lista, generalmente con un solo elemento si procesas una imagen.
if results and len(results) > 0:
    result = results[0]  # Tomamos el primer (y único) resultado
    boxes = (
        result.boxes
    )  # Accedemos al atributo 'boxes' que contiene la información de detección

    print(f"Se detectaron {len(boxes)} objetos.")

    for i in range(len(boxes)):
        # Obtener coordenadas del bounding box (formato xyxy)
        xyxy = boxes.xyxy[i].cpu().numpy()  # [xmin, ymin, xmax, ymax]
```

```
        # Obtener confianza y ID de clase
        confidence = boxes.conf[i].cpu().numpy()
        class_id = int(boxes.cls[i].cpu().numpy())

        # Obtener el nombre de la clase
        if class_id < len(CLASS_NAMES):
            class_name = CLASS_NAMES[class_id]
        else:
            class_name = f"ClaseDesconocida_{class_id}"

        # A. Calcular el centro del bounding box en coordenadas de la IMAGEN
        xmin, ymin, xmax, ymax = xyxy
        center_x_image = (xmin + xmax) / 2
        center_y_image = (ymin + ymax) / 2

        # B. Preparar el punto para la transformación de perspectiva
        # El formato debe ser (1, 1, 2) para un solo punto, y de tipo float32
        point_in_image_to_transform = np.array(
            [[[center_x_image, center_y_image]]], dtype="float32"
        )

        # C. Aplicar la transformación de perspectiva usando la matriz H
        transformed_point_on_table = cv2.perspectiveTransform(
            point_in_image_to_transform, H_matrix
        )

        if transformed_point_on_table is not None:
            # Extraer las coordenadas transformadas (x_mesa, y_mesa)
            center_x_table = transformed_point_on_table[0][0][0]
            center_y_table = transformed_point_on_table[0][0][1]

            print(f"\n  Bola: {class_name} (Confianza: {confidence:.2f})")
            print(
                f"    Centro en Imagen (px): ({center_x_image:.1f}, {center_y_image:.1f})"
            )
            print(
                f"    Centro en Mesa (units): ({center_x_table:.1f}, {center_y_table:.1f})"
            )

            # Guardar la información
            detected_balls_on_table_plane.append(
                {
                    "clase": class_name,
                    "confianza": float(confidence),
                    "centro_imagen_px": (
                        round(center_x_image, 1),
                        round(center_y_image, 1),
                    ),
                    "centro_mesa_units": (
                        round(center_x_table, 1),
                        round(center_y_table, 1),
                    ),
                }
```

```
            )
        else:
            print(f"Error al transformar el punto para la detección de {class_name}.")
else:
    print("No se obtuvieron resultados de la inferencia.")

print("\n--- Proceso de Detección y Transformación Completado ---")
print(f"Bolas procesadas y transformadas: {len(detected_balls_on_table_plane)}")
for ball_data in detected_balls_on_table_plane:
    print(ball_data)

# ------------------------------------------------------------------------------
# OPCIONAL: MOSTRAR LA IMAGEN CON LAS DETECCIONES ORIGINALES (DE YOLO)
# ------------------------------------------------------------------------------
# Puedes descomentar estas líneas para ver los bounding boxes que YOLO dibuja
# en la imagen original. Esto es útil para verificar que YOLO está detectando
# las bolas correctamente antes de la transformación.

annotated_frame = results[0].plot()  # El método plot() dibuja las detecciones
cv2.imshow("Detecciones YOLO en Imagen Original", annotated_frame)
cv2.waitKey(0)  # Espera a que se presione una tecla
cv2.destroyAllWindows()
```

detect_table/balls_coord_transform.py

```
import cv2
import numpy as np

# Tu Matriz de Homografía calculada
H_matrix = np.array(
    [
        [9.01494445e-01, 1.69172272e-17, -8.56419723e01],
        [-8.24671937e-04, 9.16210522e-01, -8.23806031e01],
        [1.62899797e-06, 6.57494777e-06, 1.00000000e00],
    ]
)

# Coordenadas de ejemplo de los centros de las bolas en la IMAGEN ORIGINAL (en píxeles)
# (Más adelante, estos vendrán de la salida de tu modelo YOLO)
image_ball_centers = np.array(
    [
        [[300.0, 350.0]],  # Centro de la Bola Blanca (ejemplo)
        [[800.0, 400.0]],  # Centro de la Bola Roja (ejemplo)
    ],
    dtype="float32",
)

# Asegúrate de que el array de puntos tiene la forma correcta (N, 1, 2) o (1, N, 2)
# Si tienes una lista simple de tuplas [(x1,y1), (x2,y2)], puedes hacer:
# points_to_transform = np.array([[(x1,y1)], [(x2,y2)]], dtype="float32")
```

```
if image_ball_centers.ndim == 2:  # Si es (N,2)
    # Necesita ser (N,1,2) para perspectiveTransform
    image_ball_centers = image_ball_centers.reshape(-1, 1, 2)

# Aplicar la transformación de perspectiva
table_ball_centers = cv2.perspectiveTransform(image_ball_centers, H_matrix)

if table_ball_centers is not None:
    print("Coordenadas de las bolas en la imagen original:")
    for i, point in enumerate(image_ball_centers):
        print(f"  Bola {i+1}: ({point[0][0]:.1f}, {point[0][1]:.1f})")

    print("\nCoordenadas de las bolas transformadas (en el plano de la mesa ideal):")
    for i, point in enumerate(table_ball_centers):
        print(f"  Bola {i+1}: ({point[0][0]:.1f}, {point[0][1]:.1f})")
        # Estos son los puntos (x_mesa, y_mesa) que usarías para Pygame, por ejemplo.
        # Recuerda que estarán en el rango de 0 a TABLE_VIEW_WIDTH-1 y 0 a TABLE_VIEW_HEIGHT-1.
else:
    print("Error durante la transformación de perspectiva.")
```

detect_table/select_table_corners.py

```
import cv2
import numpy as np

# Lista para almacenar los puntos de las esquinas seleccionadas
corner_points = []
current_image = None


def select_corners_mouse_callback(event, x, y, flags, param):
    global corner_points, current_image

    if event == cv2.EVENT_LBUTTONDOWN:
        if len(corner_points) < 4:
            # Dibuja un círculo en el punto seleccionado
            cv2.circle(current_image, (x, y), 5, (0, 255, 0), -1)
            cv2.imshow("Selecciona Esquinas", current_image)

            corner_points.append((x, y))
            print(f"Punto {len(corner_points)} seleccionado: ({x}, {y})")
            if len(corner_points) == 4:
                print(
                    "¡4 esquinas seleccionadas! Presiona cualquier tecla para continuar."
                )
        else:
            print("Ya has seleccionado 4 esquinas. Presiona una tecla.")
```

```
def get_manual_corners(image_path):
    global corner_points, current_image
    corner_points = []  # Resetea los puntos para una nueva imagen

    original_image = cv2.imread(image_path)
    if original_image is None:
        print(f"Error: No se pudo cargar la imagen {image_path}")
        return None

    current_image = (
        original_image.copy()
    )  # Trabajamos sobre una copia para no alterar la original

    cv2.namedWindow("Selecciona Esquinas")
    cv2.setMouseCallback("Selecciona Esquinas", select_corners_mouse_callback)

    print(
        "Por favor, haz clic en las 4 esquinas de la mesa en la imagen (ej: superior-izquierda, superior-derecha,
    )
    cv2.imshow("Selecciona Esquinas", current_image)

    # Espera hasta que se presione una tecla y se hayan seleccionado 4 puntos
    while not (len(corner_points) == 4 and cv2.waitKey(0) != -1):
        if cv2.waitKey(1) & 0xFF == 27:  # Si se presiona ESC
            print("Selección cancelada.")
            cv2.destroyAllWindows()
            return None
        if (
            cv2.getWindowProperty("Selecciona Esquinas", cv2.WND_PROP_VISIBLE) < 1
        ):  # Si se cierra la ventana
            print("Ventana cerrada. Selección cancelada.")
            return None

    cv2.destroyAllWindows()

    if len(corner_points) == 4:
        print("Esquinas seleccionadas:", corner_points)
        return np.array(corner_points, dtype="float32")
    else:
        print("No se seleccionaron 4 esquinas.")
        return None


# --- Uso ---
if __name__ == "__main__":
    # Reemplaza 'ruta/a/tu/imagen_de_billar.jpg' con la ruta a una de tus imágenes
    image_file = "tests/test_pool_table_1.png"

    # Para probar, puedes usar una de las imágenes de ejemplo que vienen con el modelo YOLO
    # o una que tengas de una mesa de billar.
    # Por ejemplo, si tienes una imagen llamada "mesa1.jpg" en la misma carpeta:
    # image_file = 'mesa1.jpg'
```

```
    selected_corners = get_manual_corners(image_file)

    if selected_corners is not None:
        # Aquí 'selected_corners' es un array de NumPy con los 4 puntos.
        # Estos son tus "Puntos de Origen" para la homografía.
        print("\nCoordenadas de las esquinas en la imagen (Puntos de Origen):")
        print(selected_corners)

        # El siguiente paso sería definir los "Puntos de Destino" y calcular la homografía.
    else:
        print("No se pudo completar la selección de esquinas.")
```

detect_table/filter_table_borders.py

```
import math

import cv2
import numpy as np

# Carga la imagen
image_path = "tests/test_pool_table_1.png"  # La misma que antes
original_image = cv2.imread(image_path)

if original_image is None:
    print(f"Error al cargar la imagen: {image_path}")
    exit()

gray_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
edges = cv2.Canny(blurred_image, 50, 150)  # Ajusta estos umbrales si es necesario

# Detección de Líneas con la Transformada de Hough (Probabilística)
# Ajusta threshold, minLineLength, maxLineGap según tu experimentación anterior
lines = cv2.HoughLinesP(
    edges, 1, np.pi / 180, threshold=80, minLineLength=50, maxLineGap=20
)

image_with_filtered_lines = original_image.copy()
horizontal_lines = []
vertical_lines = []

if lines is not None:
    for line_segment in lines:
        x1, y1, x2, y2 = line_segment[0]

        # Calcular longitud de la línea
        length = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

        # Calcular ángulo de la línea
        if (x2 - x1) == 0:  # Línea vertical
```

```
        angle = 90.0
    else:
        angle = math.degrees(math.atan((y2 - y1) / (x2 - x1)))

    # --- Criterios de Filtrado (¡NECESITARÁS AJUSTAR ESTO!) ---
    # 1. Filtrar por longitud mínima
    MIN_LINE_LENGTH = 100  # Ejemplo: solo líneas de más de 100 píxeles
    if length < MIN_LINE_LENGTH:
        continue

    # 2. Clasificar en horizontales y verticales (aproximadas)
    # Ángulos cercanos a 0° (horizontales), ángulos cercanos a +/-90° (verticales)
    ANGLE_THRESHOLD = (
        30  # Ejemplo: +/- 30 grados de la horizontal/vertical perfecta
    )

    if abs(angle) < ANGLE_THRESHOLD:  # Considerada "horizontal-ish"
        horizontal_lines.append(((x1, y1, x2, y2), length))
        # cv2.line(image_with_filtered_lines, (x1, y1), (x2, y2), (0, 255, 0), 2) # Verde para horizontales
    elif abs(abs(angle) - 90) < ANGLE_THRESHOLD:  # Considerada "vertical-ish"
        vertical_lines.append(((x1, y1, x2, y2), length))
        # cv2.line(image_with_filtered_lines, (x1, y1), (x2, y2), (255, 0, 0), 2) # Azul para verticales
    # else: # Líneas diagonales que no cumplen los umbrales (opcional)
    # cv2.line(image_with_filtered_lines, (x1, y1), (x2, y2), (0,0,0), 1) # Negro para otras

# Opcional: Quedarse con las N líneas más largas de cada categoría
# Por ejemplo, las 2 más largas horizontales y las 2 más largas verticales
horizontal_lines.sort(
    key=lambda item: item[1], reverse=True
)  # Ordenar por longitud
vertical_lines.sort(key=lambda item: item[1], reverse=True)

# Dibujar, por ejemplo, las 4 líneas horizontales y 4 verticales más largas (o menos si no hay tantas)
# Esto es una simplificación; idealmente buscarías las líneas que mejor forman el rectángulo de la mesa

# Dibujamos las líneas seleccionadas
selected_lines_for_corners = []

print(
    f"Líneas horizontales detectadas (filtradas por longitud > {MIN_LINE_LENGTH}): {len(horizontal_lines)}"
)
for i, (line_coords, length) in enumerate(
    horizontal_lines[:4]
):  # Tomar hasta las 4 más largas
    x1, y1, x2, y2 = line_coords
    cv2.line(
        image_with_filtered_lines, (x1, y1), (x2, y2), (0, 255, 0), 3
    )  # Verde más grueso
    selected_lines_for_corners.append(line_coords)
    print(f"  H{i+1}: ({x1},{y1})-({x2},{y2}), Longitud: {length:.0f}")

print(
    f"Líneas verticales detectadas (filtradas por longitud > {MIN_LINE_LENGTH}): {len(vertical_lines)}"
```

```
    )
    for i, (line_coords, length) in enumerate(
        vertical_lines[:4]
    ):  # Tomar hasta las 4 más largas
        x1, y1, x2, y2 = line_coords
        cv2.line(
            image_with_filtered_lines, (x1, y1), (x2, y2), (255, 0, 0), 3
        )  # Azul más grueso
        selected_lines_for_corners.append(line_coords)
        print(f"  V{i+1}: ({x1},{y1})-({x2},{y2}), Longitud: {length:.0f}")

else:
    print("No se detectaron líneas con los parámetros actuales.")

cv2.imshow("Imagen Original", original_image)
cv2.imshow("Bordes (Canny)", edges)
cv2.imshow("Lineas Filtradas", image_with_filtered_lines)
cv2.waitKey(0)
cv2.destroyAllWindows()

# El siguiente paso sería tomar 'selected_lines_for_corners' y encontrar sus intersecciones.
```

detect_table/find_table_borders.py

```
import cv2
import numpy as np

# Carga la imagen
image_path = "tests/test_pool_table_1.png"  # Usa la misma imagen que antes
original_image = cv2.imread(image_path)

if original_image is None:
    print(f"Error al cargar la imagen: {image_path}")
    exit()

# Convertir a escala de grises (común para la detección de bordes)
gray_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)

# Aplicar un desenfoque gaussiano para suavizar la imagen y reducir el ruido
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

# 1. Detección de Bordes con Canny
# Los umbrales (50 y 150) pueden necesitar ajuste
edges = cv2.Canny(blurred_image, 40, 50)
# edges = cv2.Canny(blurred_image, 50, 150)
cv2.imshow("Bordes Detectados (Canny)", edges)
cv2.waitKey(0)  # Presiona una tecla para continuar

# 2. Detección de Líneas con la Transformada de Hough
# cv2.HoughLinesP(imagen_bordes, rho, theta, umbral_interseccion, minLineLength, maxLineGap)
```

```
# rho: resolución de la distancia en píxeles
# theta: resolución del ángulo en radianes (np.pi/180 para 1 grado)
# umbral_interseccion: número mínimo de votos (intersecciones en el espacio de Hough)
# minLineLength: longitud mínima de una línea para ser considerada.
# maxLineGap: máxima brecha permitida entre puntos en la misma línea.
# Estos parámetros también necesitarán ajuste.
lines = cv2.HoughLinesP(
    edges, 1, np.pi / 180, threshold=15, minLineLength=200, maxLineGap=10
)
# edges, 1, np.pi / 180, threshold=80, minLineLength=50, maxLineGap=10

# Dibuja las líneas detectadas en la imagen original (o una copia)
image_with_lines = original_image.copy()
if lines is not None:
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(
            image_with_lines, (x1, y1), (x2, y2), (0, 0, 255), 2
        )  # Dibuja líneas rojas

cv2.imshow("Lineas Detectadas (Hough)", image_with_lines)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

detect_table/perspective_transform.py

```
import cv2
import numpy as np

# Puntos de origen (los que seleccionaste manualmente)
source_points = np.array(
    [
        [95.0, 90.0],  # Superior-Izquierda
        [1206.0, 91.0],  # Superior-Derecha
        [1210.0, 639.0],  # Inferior-Derecha
        [95.0, 637.0],  # Inferior-Izquierda
    ],
    dtype="float32",
)

# Puntos de destino (en la mesa ideal)
# Definamos un ancho y alto para nuestra mesa rectificada
TABLE_VIEW_WIDTH = 1000
TABLE_VIEW_HEIGHT = 500  # Manteniendo una proporción aproximada de 2:1

destination_points = np.array(
    [
        [0, 0],  # Superior-Izquierda
        [TABLE_VIEW_WIDTH - 1, 0],  # Superior-Derecha
        [TABLE_VIEW_WIDTH - 1, TABLE_VIEW_HEIGHT - 1],  # Inferior-Derecha
```

```
        [0, TABLE_VIEW_HEIGHT - 1],  # Inferior-Izquierda
    ],
    dtype="float32",
)

# Calcular la matriz de homografía
H_matrix, status = cv2.findHomography(source_points, destination_points)
# Alternativamente, y a menudo preferido para solo 4 puntos:
# H_matrix = cv2.getPerspectiveTransform(source_points, destination_points)


if H_matrix is not None:
    print("Matriz de Homografía calculada (H):")
    print(H_matrix)

    # ¿Qué podemos hacer ahora con esta matriz?
    # 1. Aplicar la transformación a toda la imagen para obtener una vista cenital (opcional, bueno para visualiz
    # 2. Transformar las coordenadas específicas de las bolas detectadas por YOLO (nuestro objetivo principa

    # Ejemplo de cómo obtener la vista cenital de la mesa (Paso Opcional de Visualización)
    # Carga la imagen original de nuevo
    # image_path = 'ruta/a/tu/imagen_de_billar.jpg' # La misma que usaste antes
    # original_image = cv2.imread(image_path)
    # if original_image is not None:
    #     warped_table_view = cv2.warpPerspective(original_image, H_matrix, (TABLE_VIEW_WIDTH, TABLE_VIE
    #     cv2.imshow("Imagen Original", original_image)
    #     cv2.imshow("Vista Cenital de la Mesa (Rectificada)", warped_table_view)
    #     cv2.waitKey(0)
    #     cv2.destroyAllWindows()
    # else:
    #     print(f"Error: No se pudo cargar la imagen original {image_path} para la rectificación.")

else:
    print("No se pudo calcular la matriz de homografía.")
```

determine_game/convert_all_xml_to_csv.py

```
# prompt: con el ultimo bloque que te he puesto, la conversion es correcta, pero con el primero que conviert
import csv
import os
import xml.etree.ElementTree as ET


def convert_xml_to_csv_all(xml_file_path):
    """
    Converts data from the 'Points' section of an Excel XML file to a CSV file.

    Args:
        xml_file_path (str): The path to the input XML file.
    """
```

```
    try:
        # Register namespaces
        namespaces = {
            "ss": "urn:schemas-microsoft-com:office:spreadsheet",
            "o": "urn:schemas-microsoft-com:office:office",
            "x": "urn:schemas-microsoft-com:office:excel",
            "html": "http://www.w3.org/TR/REC-html40",
            "": "urn:schemas-microsoft-com:office:spreadsheet",  # Default namespace
        }
        for prefix, uri in namespaces.items():
            ET.register_namespace(prefix, uri)

        tree = ET.parse(xml_file_path)
        root = tree.getroot()

        # Use the default namespace explicitly for find/findall
        ns_default = {"d": namespaces[""]}

        points_data = []
        header_row = []
        in_points_section = False
        header_collected = False

        for row_element in root.findall(".//d:Worksheet/d:Table/d:Row", ns_default):
            cells = row_element.findall("d:Cell", ns_default)

            if not cells:
                continue

            # Detect the "Points" section
            first_cell_data = cells[0].find("d:Data", ns_default)
            if first_cell_data is not None and first_cell_data.text == "Points":
                in_points_section = True
                continue

            if in_points_section:
                if not header_collected:
                    # The row after "Points" should be the header
                    temp_header = []
                    for cell in cells:
                        data_element = cell.find("d:Data", ns_default)
                        if data_element is not None:
                            temp_header.append(data_element.text)
                        else:
                            temp_header.append("")  # Empty cell

                    # Validate if this is the expected header for points
                    if temp_header == ["Name", "X", "Y", "Time"]:
                        header_row = temp_header
                        header_collected = True
                        points_data.append(header_row)
                    else:
                        # If not the expected header, might be the end of the section or unexpected data
```

```
                # We could potentially stop collecting data here if the pattern breaks
                # For now, we just ignore this row and keep searching for the header/data rows
                pass
            continue

        # Collect data rows after the header is found
        if header_collected:
            current_row_data = []
            # Ensure the row has the expected number of cells for point data
            if len(cells) >= len(
                header_row
            ):  # Use >= in case there are extra empty cells at the end
                for i in range(len(header_row)):
                    if i < len(cells):
                        cell = cells[i]
                        data_element = cell.find("d:Data", ns_default)
                        if data_element is not None:
                            current_row_data.append(data_element.text)
                        else:
                            current_row_data.append("")  # Empty cell
                    else:
                        current_row_data.append(
                            ""
                        )  # Missing cell for a header column

                # Add the row data if it seems valid (e.g., first element is not empty, indicating a point name)
                # This is a simple check; adjust if needed based on actual data patterns
                if current_row_data and current_row_data[0]:
                    points_data.append(current_row_data)
                else:
                    # If the first cell is empty, it might indicate the end of the data section
                    # or an empty row we should ignore. We can potentially stop collecting here.
                    in_points_section = False  # Stop collecting data
                    header_collected = False  # Reset state

    if not points_data or len(points_data) <= 1:  # No data rows found after header
        print(
            f"No se encontraron datos de 'Points' o la sección está vacía/malformada en {xml_file_path}"
        )
        return

    # Determine the output CSV file path
    base, ext = os.path.splitext(xml_file_path)
    csv_file_path = base + ".csv"

    with open(csv_file_path, "w", newline="", encoding="utf-8") as csv_file:
        writer = csv.writer(csv_file)
        writer.writerows(points_data)

    print(f"Archivo CSV generado exitosamente: {csv_file_path}")

except ET.ParseError as e:
    print(f"Error al parsear el archivo XML {xml_file_path}: {e}")
```

```
    except FileNotFoundError:
        print(f"Archivo XML no encontrado: {xml_file_path}")
    except Exception as e:
        print(f"Ocurrió un error inesperado al procesar {xml_file_path}: {e}")


def convert_xmls_in_directory_improved(directory):
    """
    Recursively finds and converts all XML files in a directory to CSV,
    using the improved conversion logic for "Points" data.

    Args:
        directory (str): The starting directory to search for XML files.
    """
    print(f"Searching for XML files in: {directory}")
    found_xml = False
    for root, _, files in os.walk(directory):
        for file in files:
            if file.endswith(".xml"):
                found_xml = True
                xml_file_path = os.path.join(root, file)
                print(f"\nProcessing file: {xml_file_path}")
                convert_xml_to_csv_all(xml_file_path)

    if not found_xml:
        print(f"No XML files found in {directory} or its subdirectories.")
    else:
        print("\nFinished processing XML files.")


# Specify the starting directory
start_directory = "data/data_layouts/AllCordinates"

# Create the directory if it doesn't exist (for demonstration purposes)
# You might want to add some dummy XML files here for testing
if not os.path.exists(start_directory):
    print(f"Creating directory: {start_directory}")
    os.makedirs(start_directory)
    # Example dummy file creation (replace with your actual file structure if possible)
    # It's crucial that the dummy files mimic the structure of your real XML files
    # for the parsing logic to work.
    example_xml_content = """<?xml version="1.0"?>
    <?mso-application progid="Excel.Sheet"?>
    <Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
     xmlns:o="urn:schemas-microsoft-com:office:office"
     xmlns:x="urn:schemas-microsoft-com:office:excel"
     xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
     xmlns:html="http://www.w3.org/TR/REC-html40">
     <Worksheet ss:Name="Sheet1">
      <Table ss:ExpandedColumnCount="4" ss:ExpandedRowCount="10" x:FullColumns="1"
       x:FullRows="1" ss:DefaultRowHeight="15">
       <Row>
        <Cell><Data ss:Type="String">Header Before Points</Data></Cell>
```

```
    </Row>
    <Row>
     <Cell ss:Index="1" ss:MergeAcross="3"><Data ss:Type="String">Points</Data></Cell>
    </Row>
    <Row>
     <Cell><Data ss:Type="String">Name</Data></Cell>
     <Cell><Data ss:Type="String">X</Data></Cell>
     <Cell><Data ss:Type="String">Y</Data></Cell>
     <Cell><Data ss:Type="String">Time</Data></Cell>
    </Row>
    <Row>
     <Cell><Data ss:Type="String">PointA</Data></Cell>
     <Cell><Data ss:Type="Number">10</Data></Cell>
     <Cell><Data ss:Type="Number">20</Data></Cell>
     <Cell><Data ss:Type="Number">1.0</Data></Cell>
    </Row>
    <Row>
     <Cell><Data ss:Type="String">PointB</Data></Cell>
     <Cell><Data ss:Type="Number">30</Data></Cell>
     <Cell><Data ss:Type="Number">40</Data></Cell>
     <Cell><Data ss:Type="Number">1.5</Data></Cell>
    </Row>
     <Row>
     <Cell><Data ss:Type="String">PointC</Data></Cell>
     <Cell><Data ss:Type="Number">50</Data></Cell>
     <Cell><Data ss:Type="Number">60</Data></Cell>
     <Cell><Data ss:Type="Number">2.0</Data></Cell>
    </Row>
     <Row>
     <Cell ss:Index="1" ss:MergeAcross="3"><Data ss:Type="String">Header After Points</Data></Cell>
    </Row>
   </Table>
  </Worksheet>
 </Workbook>
 """
 dummy_dir = os.path.join(start_directory, "2002MosconiCup", "ArcherVSDavis-M-6")
 os.makedirs(dummy_dir, exist_ok=True)
 dummy_file_path = os.path.join(dummy_dir, "Frame1.xml")
 with open(dummy_file_path, "w", encoding="utf-8") as f:
     f.write(example_xml_content)
 print(f"Created dummy XML file: {dummy_file_path}")


# Run the conversion for all files in the directory
convert_xmls_in_directory_improved(start_directory)
```

determine_game/convert_xml_to_csv.py

```
import xml.etree.ElementTree as ET
import csv
```

```
import os

def convert_xml_to_csv(xml_file_path):
    """
    Convierte datos de puntos específicos de un archivo XML de Excel a un archivo CSV.

    Args:
        xml_file_path (str): La ruta al archivo XML de entrada.
    """
    try:
        # Registrar los namespaces para facilitar la búsqueda de elementos
        namespaces = {
            'ss': 'urn:schemas-microsoft-com:office:spreadsheet',
            'o': 'urn:schemas-microsoft-com:office:office',
            'x': 'urn:schemas-microsoft-com:office:excel',
            'html': 'http://www.w3.org/TR/REC-html40',
            '': 'urn:schemas-microsoft-com:office:spreadsheet' # Namespace por defecto
        }
        for prefix, uri in namespaces.items():
            ET.register_namespace(prefix, uri)

        tree = ET.parse(xml_file_path)
        root = tree.getroot()

        # El namespace por defecto debe usarse explícitamente en find/findall
        ns_default = {'d': namespaces['']}

        points_data = []
        header_row = []
        in_points_section = False
        header_collected = False

        for row_element in root.findall('.//d:Worksheet/d:Table/d:Row', ns_default):
            cells = row_element.findall('d:Cell', ns_default)

            if not cells:
                continue

            # Detectar la sección "Points"
            first_cell_data = cells[0].find('d:Data', ns_default)
            if first_cell_data is not None and first_cell_data.text == "Points":
                in_points_section = True
                continue

            if in_points_section:
                if not header_collected:
                    # La siguiente fila después de "Points" es la cabecera
                    for cell in cells:
                        data_element = cell.find('d:Data', ns_default)
                        if data_element is not None:
                            header_row.append(data_element.text)
                    if header_row == ["Name", "X", "Y", "Time"]: # Confirmar que es la cabecera correcta
                        header_collected = True
```

```
                    points_data.append(header_row)
                else: # Si no es la cabecera esperada, resetear
                    header_row = []
                continue

            # Recolectar filas de datos
            if header_collected:
                current_row_data = []
                # Asegurarse de que la fila tiene el número esperado de celdas para los datos de puntos
                if len(cells) == len(header_row):
                    for cell in cells:
                        data_element = cell.find('d:Data', ns_default)
                        if data_element is not None:
                            current_row_data.append(data_element.text)
                        else:
                            current_row_data.append('') # Celda vacía
                    points_data.append(current_row_data)
                else:
                    # Si la fila no coincide con el formato de datos de puntos,
                    # podríamos haber salido de la sección de puntos.
                    # Por ahora, simplemente la ignoramos. Podríamos detener la recolección aquí.
                    pass

        if not points_data or len(points_data) <= 1: # No se encontraron datos o solo la cabecera
            print(f"No se encontraron datos de 'Points' o la sección está vacía en {xml_file_path}")
            return

        # Determinar la ruta del archivo CSV de salida
        base, ext = os.path.splitext(xml_file_path)
        csv_file_path = base + ".csv"

        with open(csv_file_path, 'w', newline='', encoding='utf-8') as csv_file:
            writer = csv.writer(csv_file)
            writer.writerows(points_data)

        print(f"Archivo CSV generado exitosamente: {csv_file_path}")

    except ET.ParseError as e:
        print(f"Error al parsear el archivo XML {xml_file_path}: {e}")
    except FileNotFoundError:
        print(f"Archivo XML no encontrado: {xml_file_path}")
    except Exception as e:
        print(f"Ocurrió un error inesperado: {e}")

if __name__ == '__main__':
    # Ruta al archivo XML que proporcionaste
    # Puedes cambiar esto o pasarla como argumento al script si lo haces más genérico
    xml_input_path = 'data/data_layouts/AllCordinates/2002MosconiCup/ArcherVSDavis-M-6/Frame1.xml'

    # Verificar si el archivo XML de entrada existe
    if not os.path.exists(xml_input_path):
        print(f"El archivo XML de entrada no existe: {xml_input_path}")
```

```
    else:
        convert_xml_to_csv(xml_input_path)
```

determine_game/convert_xlsx_to_csv.py

```
import pandas as pd
import os

def convert_xlsx_to_csv(xlsx_file_path, sheet_name=0, index=False):
    """
    Convierte una hoja específica de un archivo XLSX a un archivo CSV.

    Args:
        xlsx_file_path (str): La ruta al archivo XLSX de entrada.
        sheet_name (str or int, optional): Nombre o índice de la hoja a convertir.
                                           Por defecto es 0 (la primera hoja).
        index (bool, optional): Escribir el índice del DataFrame como una columna.
                                Por defecto es False.
    """
    try:
        # Comprobar si el archivo de entrada existe
        if not os.path.exists(xlsx_file_path):
            print(f"Error: El archivo XLSX de entrada no existe en la ruta: {xlsx_file_path}")
            return

        # Leer la hoja especificada del archivo Excel
        df = pd.read_excel(xlsx_file_path, sheet_name=sheet_name)

        # Determinar la ruta del archivo CSV de salida
        base, ext = os.path.splitext(xlsx_file_path)
        csv_file_path = base + ".csv"

        # Convertir el DataFrame a CSV
        df.to_csv(csv_file_path, index=index, encoding='utf-8')

        print(f"Archivo CSV generado exitosamente: {csv_file_path}")

    except FileNotFoundError:
        print(f"Error: Archivo XLSX no encontrado en la ruta: {xlsx_file_path}")
    except Exception as e:
        # Capturar otros errores posibles, como que el archivo no sea un XLSX válido
        # o que la hoja especificada no exista.
        print(f"Ocurrió un error inesperado: {e}")
        if "No sheet named" in str(e):
            print(f"Asegúrate de que la hoja '{sheet_name}' exista en el archivo '{xlsx_file_path}'.")
        elif isinstance(e, pd.errors.EmptyDataError):
            print(f"La hoja '{sheet_name}' en '{xlsx_file_path}' está vacía.")
        elif "Excel file format cannot be determined" in str(e) or "Unsupported format, or corrupt file" in str(e):
             print(f"El archivo '{xlsx_file_path}' podría no ser un archivo Excel válido o está corrupto.")
```

```
if __name__ == '__main__':
    # Especifica la ruta a tu archivo Variables.xlsx
    # Asegúrate de que este archivo exista en la ruta especificada.
    # Por ejemplo, si está en una carpeta 'data':
    input_xlsx_path = 'data/data_layouts/Variables/2002MosconiCup/ArcherVSDavis-M-6/Variable.xlsx'

    # Llama a la función de conversión
    # Por defecto, convierte la primera hoja (índice 0)
    # Si quieres convertir una hoja con un nombre específico, usa sheet_name='NombreDeLaHoja'
    convert_xlsx_to_csv(input_xlsx_path)

    # Ejemplo para convertir una hoja llamada 'DatosImportantes':
    # convert_xlsx_to_csv(input_xlsx_path, sheet_name='DatosImportantes')
```

motor_inferencia.py

```
# motor_inferencia.py
import os

import cv2
import joblib
import pandas as pd
from ultralytics import YOLO

# --- 1. CONFIGURACIÓN Y CARGA DE MODELOS ---
print("Iniciando motor de inferencia...")
DETECTION_MODEL_PATH = "pool_hybrid.pt"
CONTEXT_MODEL_PATH = "context_classifier.joblib"

# Listas de clases y reglas
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",
]
CLASES_COMPARTIDAS = {
    "black_8",
    "blue_2",
    "dred_7",
    "green_6",
    "red_3",
    "white",
    "yellow_1",
}
DELATORES_BE = {
    "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_CLASSIC = {
    "blue_10",
    "dred_15",
    "green_14",
    "orange_13",
    "orange_5",
    # "purple_12",
    # "purple_4",
    "red_11",
    "yellow_9",
}
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",
    "purple_4": "be_purple_5",
    "purple_12": "be_purple_13",
}

# Cargar los dos modelos
try:
    detection_model = YOLO(DETECTION_MODEL_PATH)
```

```
    context_model = joblib.load(CONTEXT_MODEL_PATH)
    print("Motor de IA: Modelos de detección y contexto cargados correctamente.")
except Exception as e:
    print(f"Error fatal al cargar modelos en el motor: {e}")
    detection_model = None
    context_model = None

# --- 2. FUNCIONES DE LÓGICA ---


def extraer_features_para_contexto(detecciones_yolo):
    """Prepara los datos de entrada para el clasificador de contexto (Random Forest)."""
    puntuacion_be = sum(
        d["conf"] for d in detecciones_yolo if d["clase"] in DELATORES_BE
    )
    puntuacion_classic = sum(
        d["conf"] for d in detecciones_yolo if d["clase"] in DELATORES_CLASSIC
    )

    counts = {f"count_{cls}": 0 for cls in CLASES_HIBRIDAS}
    confianza_total = sum(d["conf"] for d in detecciones_yolo)

    for d in detecciones_yolo:
        counts[f"count_{d['clase']}"] += 1

    total_detecciones = len(detecciones_yolo)
    confianza_media = (
        confianza_total / total_detecciones if total_detecciones > 0 else 0
    )

    features = {
        "puntuacion_be": puntuacion_be,
        "puntuacion_classic": puntuacion_classic,
        "total_detecciones": total_detecciones,
        "confianza_media": confianza_media,
    }
    features.update(counts)

    # Asegurar el orden de las columnas para que coincida con el del entrenamiento
    column_order = [
        "puntuacion_be",
        "puntuacion_classic",
        "total_detecciones",
        "confianza_media",
    ] + [f"count_{cls}" for cls in CLASES_HIBRIDAS]

    return pd.DataFrame([features], columns=column_order)


def refinar_etiquetas(detecciones_yolo, contexto):
    """
    Aplica corrección contextual y luego simplifica TODAS las etiquetas
    para que no contengan el prefijo 'be_'.
```

```
    """
    detecciones_refinadas = []
    for deteccion in detecciones_yolo:
        etiqueta_yolo = deteccion["clase"]
        etiqueta_corregida = etiqueta_yolo

        # 1. Aplicar corrección contextual si es necesario
        if contexto == "classic" and etiqueta_yolo in DELATORES_BE:
            etiqueta_corregida = MAPA_CORRECCION_CONTEXTUAL.get(
                etiqueta_yolo, etiqueta_yolo
            )
            # print(f"  - CORRECCIÓN (Classic): YOLO dijo '{etiqueta_yolo}', se corrige a '{etiqueta_corregida}'.")
        elif contexto == "black_edition" and etiqueta_yolo in DELATORES_CLASSIC:
            etiqueta_corregida = MAPA_CORRECCION_CONTEXTUAL.get(
                etiqueta_yolo, etiqueta_yolo
            )
            # print(f"  - CORRECCIÓN (BE): YOLO dijo '{etiqueta_yolo}', se corrige a '{etiqueta_corregida}'.")

        # 2. Simplificar la etiqueta final, eliminando siempre el prefijo 'be_'
        etiqueta_final_simple = etiqueta_corregida.replace("be_", "")

        deteccion["etiqueta_final"] = etiqueta_final_simple
        detecciones_refinadas.append(deteccion)

    return detecciones_refinadas


# --- 3. FUNCIÓN PRINCIPAL DE INFERENCIA ---
def realizar_inferencia(ruta_imagen_entrada, ruta_imagen_salida):
    if not detection_model or not context_model:
        raise Exception("Los modelos de IA no están cargados.")

    # 1. El "Ojo" (YOLO) detecta las bolas
    print("1. El Ojo (YOLO) detecta las bolas")
    results = detection_model.predict(ruta_imagen_entrada, verbose=False, conf=0.4)
    detecciones_yolo = [
        {
            "clase": CLASES_HIBRIDAS[int(box.cls[0])],
            "conf": float(box.conf[0]),
            "box_xyxy": box.xyxy[0].tolist(),
        }
        for box in results[0].boxes
    ]

    if not detecciones_yolo:
        img = cv2.imread(ruta_imagen_entrada)
        cv2.imwrite(ruta_imagen_salida, img)
        return [], "No se detectaron bolas"

    # 2. El "Cerebro" (Random Forest) determina el contexto
    print("2. El Cerebro (Random Forest) determina el contexto")
    features = extraer_features_para_contexto(detecciones_yolo)
    contexto_pred_id = context_model.predict(features)[0]
```

```
    contexto = "black_edition" if contexto_pred_id == 1 else "classic"

    # 3. La lógica final refina las etiquetas
    print("3. La lógica final refina las etiquetas")
    detecciones_finales = refinar_etiquetas(detecciones_yolo, contexto)

    # 4. Dibujar resultados y guardar
    img = cv2.imread(ruta_imagen_entrada)
    for d in detecciones_finales:
        x1, y1, x2, y2 = map(int, d["box_xyxy"])
        label = f"{d['etiqueta_final']} {d['conf']:.2f}"
        color = (0, 255, 0)  # Verde en BGR
        cv2.rectangle(img, (x1, y1), (x2, y2), color, 2)
        (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, 0, 0), 2
        )

    cv2.imwrite(ruta_imagen_salida, img)

    return detecciones_finales, contexto
```

prepare_amd_gpu/verify_rocm_tensorflow.py

```
import tensorflow as tf

print(f"TensorFlow Version: {tf.__version__}")

# Intentar configurar la memoria de la GPU para evitar problemas de asignación comunes
gpus = tf.config.list_physical_devices("GPU")
if gpus:
    try:
        # Configura el crecimiento de memoria para todas las GPUs disponibles
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print(f"Memory growth set for {len(gpus)} GPU(s)")
    except RuntimeError as e:
        # El crecimiento de memoria debe establecerse antes de que las GPUs hayan sido inicializadas
        print(f"Error setting memory growth: {e}")

print("Num GPUs Available: ", len(gpus))

if len(gpus) > 0:
    print("GPU(s) detected:", gpus)
    for i, gpu_device in enumerate(gpus):
        print(f"--- Details for GPU {i} ---")
        print(f"  Name: {gpu_device.name}")
        """
        try:
```

```
            print(f"  Attempting operation on GPU {i}...")
            with tf.device(gpu_device.name):
                a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
                b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])
                c = tf.matmul(a, b)
                print(f"  Matrix multiplication successful on GPU {i}:\n{c.numpy()}")
        except RuntimeError as e:
            print(f"  Error during GPU {i} operation test: {e}")
        """
else:
    print("No GPU detected by TensorFlow.")
    print(
        "Ensure ROCm is correctly installed, environment variables (PATH, LD_LIBRARY_PATH) point to ROCm,"
    )
    print("and your TensorFlow version supports your ROCm version.")

# Información adicional de la build de TensorFlow
print(
    f"Is TensorFlow built with CUDA support? {tf.test.is_built_with_cuda()}"
)  # Esperamos False o que no importe si ROCm es True
print(
    f"Is TensorFlow built with ROCm support? {tf.test.is_built_with_rocm()}"
)  # ¡Esperamos True!

from tensorflow.python.framework import test_util

if tf.config.list_physical_devices("GPU"):  # Usando la forma actualizada
    print(
        "TensorFlow reports GPU is available via tf.config.list_physical_devices('GPU')."
    )
else:
    print(
        "TensorFlow reports GPU is NOT available via tf.config.list_physical_devices('GPU')."
    )
```

prepare_amd_gpu/verify_rocm_pytorch.py

```
import torch

print(f"PyTorch version: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}")  # Para NVIDIA
print(
    f"ROCm available: {torch.version.hip is not None and torch.cuda.is_available()}"
)  # Para AMD con ROCm (PyTorch lo reporta como CUDA-like)
if torch.cuda.is_available():
    print(f"Device name: {torch.cuda.get_device_name(0)}")
```

Proyecto_Bolas_LS/aumentar_dataset_be.py

```
# aumentar_dataset_be.py
import os
import random

import albumentations as A
import cv2

# --- CONFIGURACIÓN ---
# Directorios de origen (tus 60 imágenes y etiquetas corregidas)
IMAGENES_ORIGINALES_DIR = "Proyecto_Bolas_LS/imagenes/"
LABELS_ORIGINALES_DIR = "Proyecto_Bolas_LS/final_yolo_labels/"

# Directorios de salida para los datos aumentados
IMAGENES_AUMENTADAS_DIR = "Proyecto_Bolas_LS/dataset_be_aumentado/images/"
LABELS_AUMENTADAS_DIR = "Proyecto_Bolas_LS/dataset_be_aumentado/labels/"

# Número de versiones nuevas que quieres crear por cada imagen original
NUM_AUMENTACIONES_POR_IMAGEN = 15
# --- FIN DE LA CONFIGURACIÓN ---


def augment_dataset():
    # Crear directorios de salida
    os.makedirs(IMAGENES_AUMENTADAS_DIR, exist_ok=True)
    os.makedirs(LABELS_AUMENTADAS_DIR, exist_ok=True)

    # Definir el pipeline de aumentación. ¡Aquí está la magia!
    # BboxParams se asegura de que las cajas se transformen junto con la imagen.
    transform = A.Compose(
        [
            A.RandomBrightnessContrast(p=0.3),
            A.GaussNoise(p=0.2),
            A.Blur(blur_limit=3, p=0.2),
            A.Rotate(limit=20, p=0.5, border_mode=cv2.BORDER_CONSTANT),
            A.HorizontalFlip(p=0.5),
            A.ShiftScaleRotate(
                shift_limit=0.05,
                scale_limit=0.1,
                rotate_limit=10,
                p=0.3,
                border_mode=cv2.BORDER_CONSTANT,
            ),
        ],
        bbox_params=A.BboxParams(format="yolo", label_fields=["class_labels"]),
    )

    image_files = [
        f
        for f in os.listdir(IMAGENES_ORIGINALES_DIR)
```

```
    if f.lower().endswith((".png", ".jpg", ".jpeg"))
]

print(
    f"Iniciando aumentación para {len(image_files)} imágenes. Se crearán {NUM_AUMENTACIONES_POR_IM
)

for image_name in image_files:
    image_path = os.path.join(IMAGENES_ORIGINALES_DIR, image_name)
    base_name = os.path.splitext(image_name)[0]
    label_path = os.path.join(LABELS_ORIGINALES_DIR, base_name + ".txt")

    if not os.path.exists(label_path):
        continue

    # Cargar imagen y etiquetas
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    bboxes = []
    class_labels = []
    with open(label_path, "r") as f:
        for line in f:
            parts = line.strip().split()
            class_id, x, y, w, h = map(float, parts)
            bboxes.append([x, y, w, h])
            class_labels.append(int(class_id))

    # Generar N versiones aumentadas de la imagen
    for i in range(NUM_AUMENTACIONES_POR_IMAGEN):
        augmented = transform(image=image, bboxes=bboxes, class_labels=class_labels)

        aug_image = augmented["image"]
        aug_bboxes = augmented["bboxes"]

        # Guardar la nueva imagen
        aug_image_name = f"{base_name}_aug_{i}.jpg"
        cv2.imwrite(
            os.path.join(IMAGENES_AUMENTADAS_DIR, aug_image_name),
            cv2.cvtColor(aug_image, cv2.COLOR_RGB2BGR),
        )

        # Guardar las nuevas etiquetas
        aug_label_name = f"{base_name}_aug_{i}.txt"
        with open(
            os.path.join(LABELS_AUMENTADAS_DIR, aug_label_name), "w"
        ) as f_out:
            for bbox, class_id in zip(aug_bboxes, augmented["class_labels"]):
                x, y, w, h = bbox
                f_out.write(f"{class_id} {x} {y} {w} {h}\n")

print(
    f"\n¡Aumentación completada! Se han creado {len(image_files) * NUM_AUMENTACIONES_POR_IMAGEN
```

```
    )


if __name__ == "__main__":
    augment_dataset()
```

Proyecto_Bolas_LS/verificar_etiquetas_yolo.py

```
# verificar_etiquetas_yolo.py
import os
import random

import cv2
import matplotlib.patches as patches
import matplotlib.pyplot as plt

# --- 1. CONFIGURACIÓN ---

# Directorio que contiene las imágenes del dataset "black edition"
# IMAGENES_DIR = "Proyecto_Bolas_LS/dataset_be_aumentado/images/"
IMAGENES_DIR = "detect_balls/dataset_unificado_aumentado/images/train/"
IMAGENES_DIR = "detect_balls/dataset_hibrido/images/train/"
print("Images: ", IMAGENES_DIR)

# Directorio que contiene los archivos .txt finales en formato YOLO
# YOLO_LABELS_DIR = "Proyecto_Bolas_LS/dataset_be_aumentado/labels/"
YOLO_LABELS_DIR = "detect_balls/dataset_unificado_aumentado/labels/train/"
YOLO_LABELS_DIR = "detect_balls/dataset_hibrido/labels/train/"
print("Labels: ", YOLO_LABELS_DIR)

# Directorio donde se guardarán las imágenes con las cajas dibujadas
# VERIFICATION_DIR = "Proyecto_Bolas_LS/dataset_be_aumentado/verification_results/"
VERIFICATION_DIR = (
    "detect_balls/dataset_unificado_aumentado/images/verification_results/"
)
VERIFICATION_DIR = "detect_balls/dataset_hibrido/images/verification_results/"
print("Verification: ", VERIFICATION_DIR)

# YOLO_LABELS_OUTPUT_DIR = "Proyecto_Bolas_LS/final_yolo_labels/"

# Tu lista de clases maestra. ¡DEBE SER IDÉNTICA A LA DEL SCRIPT ANTERIOR!
CLASES_MAESTRA = [
    "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_black_8",
    "be_blue_10",
    "be_blue_2",
    "be_dred_15",
    "be_dred_7",
    "be_green_14",
    "be_green_6",
    "be_purple_13",
    "be_purple_5",
    "be_pink_4",
    "be_pink_12",
    "be_red_11",
    "be_red_3",
    "be_white",
    "be_yellow_1",
    "be_yellow_9",
]
CLASES_MAESTRA = [
    "black_8",  # solida negra 8, valida para 'black_8' y 'be_black_8'
    "blue_10",  # rayada azul y blanca 10, valida para 'blue_10'
    "blue_2",  # solida azul 2, valida para 'blue_2' y 'be_blue_2'
    "dred_15",  # rayada marron y blanca 15, valida para 'dred_15'
    "dred_7",  # solida marron 7, valida para 'dred_7' y 'be_dred_7'
    "green_14",  # rayada verde y blanca 14, valida para 'green_14'
    "green_6",  # solida verde 6, valida para 'green_6' y 'be_green_6'
    "orange_13",  # rayada naranja y blanca 13, valida para 'orange_13'
    "orange_5",  # solida naranja 5, valida para 'orange_5'
    "purple_12",  # rayada morada y blanca 12, valida para 'purple_12'
    "purple_4",  # solida morada 4, valida para 'purple_4'
    "red_11",  # rayada roja y blanca 11, valida para 'red_11'
    "red_3",  # solida roja 3, valida para 'red_3' y 'be_red_3'
    "white",  # solida blanca, valida para 'white' y 'be_white'
    "yellow_1",  # solida amarilla 1, valida para 'yellow_1' y 'be_yellow_1'
    "yellow_9",  # rayada amarilla y blanca 9, valida para 'yellow_9'
    "be_blue_10",  # rayada azul y negra 10, valida para 'be_blue_10'
    "be_dred_15",  # rayada morada y negra 15, valida para 'be_dred_15'
    "be_green_14",  # rayada verde y negra 14, valida para 'be_green_14'
    "be_purple_13",  # rayada morada y negra 13, valida para 'be_purple_13'
    "be_purple_5",  # solida morada 5, valida para 'be_purple_5'
    "be_pink_4",  # solida rosa 4, valida para 'be_pink_4'
    "be_pink_12",  # rayada rosa y negra 12, valida para 'be_pink_12'
    "be_red_11",  # rayada roja y negra 11, valida para 'be_red_11'
    "be_yellow_9",  # rayada amarilla y negra 9, valida para 'be_yellow_9'
]
print("Clases_maestra: ", CLASES_MAESTRA)
```

```
# Número de imágenes aleatorias que quieres verificar
NUM_IMAGENES_A_VERIFICAR = 5

# --- FIN DE LA CONFIGURACIÓN ---


def verificar_etiquetas():
    print("\n--- INICIANDO FASE 2: Verificación visual de las etiquetas generadas ---")
    os.makedirs(VERIFICATION_DIR, exist_ok=True)

    label_files = os.listdir(YOLO_LABELS_DIR)

    # print("Total files: ", len(label_files))

    # archivos_a_verificar = random.sample(label_files, min(5, len(label_files)))
    archivos_a_verificar = random.sample(label_files, NUM_IMAGENES_A_VERIFICAR)

    # print(archivos_a_verificar)

    for label_file in archivos_a_verificar:
        base_name = os.path.splitext(label_file)[0]

        # print(base_name)

        image_path_to_verify = None
        for ext in [".jpg", ".jpeg", ".png"]:
            potential_path = os.path.join(IMAGENES_DIR, base_name + ext)
            # print(potential_path)
            if os.path.exists(potential_path):
                image_path_to_verify = potential_path
                break

        # print(image_path_to_verify)

        if not image_path_to_verify:
            continue

        image = cv2.imread(image_path_to_verify)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        h, w, _ = image.shape

        fig, ax = plt.subplots(1, figsize=(12, 8))
        ax.imshow(image)
        ax.set_title(f"Verificación de: {os.path.basename(image_path_to_verify)}")

        label_path = os.path.join(YOLO_LABELS_DIR, label_file)
        with open(label_path, "r") as f:
            for line in f:
                parts = line.strip().split()
                class_id, x_center, y_center, width, height = map(float, parts)

                box_w = width * w
```

```
            box_h = height * h
            x1 = (x_center * w) - (box_w / 2)
            y1 = (y_center * h) - (box_h / 2)

            class_name = CLASES_MAESTRA[
                int(class_id)
            ]  # Usamos la misma lista maestra

            rect = patches.Rectangle(
                (x1, y1),
                box_w,
                box_h,
                linewidth=2,
                edgecolor="cyan",
                facecolor="none",
            )
            ax.add_patch(rect)
            plt.text(
                x1,
                y1 - 10,
                class_name,
                color="cyan",
                fontsize=12,
                bbox=dict(facecolor="black", alpha=0.7),
            )

        plt.axis("off")
        output_save_path = os.path.join(
            VERIFICATION_DIR, os.path.basename(image_path_to_verify)
        )
        plt.savefig(output_save_path, bbox_inches="tight", pad_inches=0)
        plt.show()
        print(f"Imagen de verificación guardada en: {output_save_path}")

    print("--- FASE 2 COMPLETADA ---")


if __name__ == "__main__":
    verificar_etiquetas()
```

Proyecto_Bolas_LS/generar_json_para_label_studio.py

```
# generar_json_definitivo.py
import json
import os
import uuid
from urllib.parse import quote

# --- 1. CONFIGURACIÓN ---
```

```
# La ruta a la carpeta que contiene tus archivos .txt de pre-etiquetado
ETIQUETAS_DIR = "/home/oscar/Documentos/Estudios/Curso.Especialista.IA/Proyecto/src/detect_balls/data

# La ruta RELATIVA desde tu "DOCUMENT_ROOT" de Label Studio hasta las imágenes
# Tu DOCUMENT_ROOT es: .../Proyecto/
# Tus imágenes están en: .../Proyecto/src/Proyecto_Bolas_LS/imagenes/
# Por lo tanto, la ruta relativa es:
RUTA_RELATIVA_IMAGENES = "src/Proyecto_Bolas_LS/imagenes"

# El nombre del archivo JSON de salida
ARCHIVO_SALIDA_JSON = "Proyecto_Bolas_LS/tasks.json"

# Lista de clases ORIGINAL para interpretar los IDs del modelo antiguo
CLASSES_ORIGINAL = [
    "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",
]
# --- FIN DE LA CONFIGURACIÓN ---


def crear_tareas_finales():
    tasks = []
    print(f"Leyendo etiquetas desde: {ETIQUETAS_DIR}")

    for label_file in os.listdir(ETIQUETAS_DIR):
        if not label_file.endswith(".txt"):
            continue

        base_name = os.path.splitext(label_file)[0]
        # Asumimos que la imagen tiene extensión .png, .jpg, o .jpeg
        # Esto es para encontrar el nombre completo del archivo de imagen
        nombre_imagen_completo = None
        # Recreamos el nombre original buscando la extensión
        # NOTA: Esto asume que tienes las imágenes originales en alguna parte para saber su extensión
        # Si todas son png, por ejemplo, puedes simplificarlo.
        for ext in [".png", ".jpg", ".jpeg"]:
            # Solo para obtener el nombre correcto, no para acceder al archivo
            if os.path.exists(
                os.path.join(ETIQUETAS_DIR.replace("labels", "images"), base_name + ext)
```

```
            ):
                nombre_imagen_completo = base_name + ext
                break

        if not nombre_imagen_completo:
            print(f"No se pudo inferir la extensión para {base_name}. Saltando.")
            continue

        # Codificar el nombre del archivo para la URL (maneja espacios y caracteres especiales)
        nombre_imagen_url_encoded = quote(nombre_imagen_completo)

        # Construir la URL interna de Label Studio
        ruta_en_servidor = f"/data/local-files/?d={os.path.join(RUTA_RELATIVA_IMAGENES, nombre_imagen_url

        task = {
            "data": {"image": ruta_en_servidor},
            "predictions": [{"model_version": "yolov8_pretrained", "result": []}],
        }

        with open(os.path.join(ETIQUETAS_DIR, label_file), "r") as f:
            for line in f:
                parts = line.strip().split()
                if len(parts) != 5:
                    continue
                class_id, x, y, w, h = map(
                    float, [parts[0], parts[1], parts[2], parts[3], parts[4]]
                )

                task["predictions"][0]["result"].append(
                    {
                        "id": str(uuid.uuid4()),
                        "from_name": "label",
                        "to_name": "image",
                        "type": "rectanglelabels",
                        "value": {
                            "x": (x - w / 2) * 100,
                            "y": (y - h / 2) * 100,
                            "width": w * 100,
                            "height": h * 100,
                            "rotation": 0,
                            "rectanglelabels": [CLASSES_ORIGINAL[int(class_id)]],
                        },
                    }
                )
        tasks.append(task)

    with open(ARCHIVO_SALIDA_JSON, "w") as f:
        json.dump(tasks, f, indent=2)
    print(f"\n¡Éxito! Se ha generado '{ARCHIVO_SALIDA_JSON}' con {len(tasks)} tareas.")


if __name__ == "__main__":
    crear_tareas_finales()
```

Proyecto_Bolas_LS/unificar_y_dividir_completo.py

```
# unificar_y_dividir_completo.py
import os
import shutil

from sklearn.model_selection import train_test_split

# --- CONFIGURACIÓN ---
# BE_IMAGENES_DIR = "Proyecto_Bolas_LS/imagenes/"
# BE_LABELS_DIR = "Proyecto_Bolas_LS/final_yolo_labels/"
BE_IMAGENES_DIR = "Proyecto_Bolas_LS/dataset_be_aumentado/images/"
BE_LABELS_DIR = "Proyecto_Bolas_LS/dataset_be_aumentado/labels/"

DEST_IMG_TRAIN, DEST_LBL_TRAIN = (
    "detect_balls/dataset_unificado_aumentado/images/train/",
    "detect_balls/dataset_unificado_aumentado/labels/train/",
)

os.makedirs(DEST_IMG_TRAIN, exist_ok=True)
os.makedirs(DEST_LBL_TRAIN, exist_ok=True)

DEST_IMG_VALID, DEST_LBL_VALID = (
    "detect_balls/dataset_unificado_aumentado/images/valid/",
    "detect_balls/dataset_unificado_aumentado/labels/valid/",
)

os.makedirs(DEST_IMG_VALID, exist_ok=True)
os.makedirs(DEST_LBL_VALID, exist_ok=True)

DEST_IMG_TEST, DEST_LBL_TEST = (
    "detect_balls/dataset_unificado_aumentado/images/test/",
    "detect_balls/dataset_unificado_aumentado/labels/test/",
)

os.makedirs(DEST_IMG_TEST, exist_ok=True)
os.makedirs(DEST_LBL_TEST, exist_ok=True)


# Proporciones: 10% para test, del resto, 20% para validación (total: 72% train, 18% valid, 10% test)
VALID_SIZE = 0.20
TEST_SIZE = 0.10
# --- FIN DE LA CONFIGURACIÓN ---


def main():
    print(
        "Iniciando la división (train/valid/test) y copia del dataset 'black edition' aumentado..."
    )
    image_files = [
        f
```

```
        for f in os.listdir(BE_IMAGENES_DIR)
        if f.lower().endswith((".png", ".jpg", ".jpeg"))
    ]

    # Primera división: separamos el conjunto de test
    train_val_files, test_files = train_test_split(
        image_files, test_size=TEST_SIZE, random_state=42
    )
    # Segunda división: del resto, separamos train y valid
    train_files, val_files = train_test_split(
        train_val_files, test_size=VALID_SIZE, random_state=42
    )

    print(f"Total imágenes 'black edition': {len(image_files)}")
    print(f"Añadidas a entrenamiento: {len(train_files)}")
    print(f"Añadidas a validación: {len(val_files)}")
    print(f"Añadidas a test: {len(test_files)}")

    def copy_files(file_list, dest_img_dir, dest_lbl_dir):
        for img_file in file_list:
            base_name = os.path.splitext(img_file)[0]
            label_file = base_name + ".txt"
            shutil.copy(
                os.path.join(BE_IMAGENES_DIR, img_file),
                os.path.join(dest_img_dir, img_file),
            )
            shutil.copy(
                os.path.join(BE_LABELS_DIR, label_file),
                os.path.join(dest_lbl_dir, label_file),
            )

    copy_files(train_files, DEST_IMG_TRAIN, DEST_LBL_TRAIN)
    copy_files(val_files, DEST_IMG_VALID, DEST_LBL_VALID)
    copy_files(test_files, DEST_IMG_TEST, DEST_LBL_TEST)

    print(
        "\n¡Copia completada! El dataset unificado v2 está listo con las 3 divisiones."
    )


if __name__ == "__main__":
    main()
```

Proyecto_Bolas_LS/label-studio.py

```
# Install the package
# into python virtual environment
# pip install -U label-studio
```

```
# Launch it!
# label-studio
```

Proyecto_Bolas_LS/convertir_ls_a_yolo.py

```
# convertir_ls_a_yolo_FINAL.py
import json
import os
import uuid
import xml.etree.ElementTree as ET
from urllib.parse import unquote

# --- 1. CONFIGURACIÓN ---

# Ruta al archivo JSON exportado desde Label Studio
LABEL_STUDIO_JSON_PATH = "Proyecto_Bolas_LS/ls-export.json"

# Ruta al archivo XML que guardaste desde la interfaz de Label Studio
LABEL_STUDIO_XML_CONFIG_PATH = "Proyecto_Bolas_LS/config.xml"

# Carpeta donde se guardarán las nuevas etiquetas en formato YOLO (.txt)
YOLO_LABELS_OUTPUT_DIR = "Proyecto_Bolas_LS/final_yolo_labels/"

# --- FIN DE LA CONFIGURACIÓN ---


def leer_clases_desde_xml(xml_path):
    """Lee el archivo de configuración XML de Label Studio para extraer las clases en orden."""
    try:
        tree = ET.parse(xml_path)
        root = tree.getroot()
        clases = [label.get("value") for label in root.findall(".//Label")]
        print(f"Clases leídas desde '{xml_path}': {len(clases)} clases encontradas.")
        return clases
    except Exception as e:
        print(f"Error al leer o parsear el archivo XML '{xml_path}': {e}")
        return None


def convertir_anotaciones_final():
    # 1. Construir la lista de clases desde el XML (Fuente Única de Verdad)
    clases_maestra = leer_clases_desde_xml(LABEL_STUDIO_XML_CONFIG_PATH)
    if not clases_maestra:
        return
    class_to_id = {name: i for i, name in enumerate(clases_maestra)}

    # 2. Cargar el archivo JSON de Label Studio
    os.makedirs(YOLO_LABELS_OUTPUT_DIR, exist_ok=True)
    try:
        with open(LABEL_STUDIO_JSON_PATH, "r") as f:
```

```
        data = json.load(f)
except FileNotFoundError:
    print(f"Error: No se encontró el archivo '{LABEL_STUDIO_JSON_PATH}'.")
    return

print(f"Se han cargado {len(data)} tareas desde Label Studio.")

# 3. Procesar cada tarea y convertir a formato YOLO
for task in data:
    try:
        image_path = task["image"]
        image_filename = os.path.basename(unquote(image_path))
        parts = image_filename.split("-", 1)
        if len(parts) > 1 and (len(parts[0]) == 8 or len(parts[0]) > 20):
            image_filename = parts[1]
        yolo_txt_filename = os.path.splitext(image_filename)[0] + ".txt"
    except KeyError:
        print(
            f"Advertencia: La tarea con ID {task.get('id')} no tiene clave 'image'. Saltando."
        )
        continue

    output_path = os.path.join(YOLO_LABELS_OUTPUT_DIR, yolo_txt_filename)
    with open(output_path, "w") as f_yolo:
        if "label" in task and task["label"]:
            for annotation in task["label"]:
                try:
                    class_name = annotation["rectanglelabels"][0]
                    if class_name in class_to_id:
                        class_id = class_to_id[class_name]
                        x_ls, y_ls = annotation["x"], annotation["y"]
                        w_ls, h_ls = annotation["width"], annotation["height"]

                        x_norm, y_norm, w_norm, h_norm = (
                            x_ls / 100.0,
                            y_ls / 100.0,
                            w_ls / 100.0,
                            h_ls / 100.0,
                        )
                        x_center, y_center = x_norm + (w_norm / 2), y_norm + (
                            h_norm / 2
                        )

                        f_yolo.write(
                            f"{class_id} {x_center:.6f} {y_center:.6f} {w_norm:.6f} {h_norm:.6f}\n"
                        )
                    else:
                        print(
                            f"ADVERTENCIA: La clase '{class_name}' de '{image_filename}' no se encontró en 'config
                        )
                except KeyError as e:
                    print(
                        f"Advertencia: Datos incompletos en anotación para '{image_filename}'. Error: {e}."
```

```
                    )

    print(f"\n¡Conversión completada! Revisa la carpeta '{YOLO_LABELS_OUTPUT_DIR}'.")


if __name__ == "__main__":
    convertir_anotaciones_final()
```

Proyecto_Bolas_LS/procesador_final.py

```
# procesador_final.py
import json
import os
import random
from urllib.parse import unquote

import cv2
import matplotlib.patches as patches
import matplotlib.pyplot as plt

# --- 1. CONFIGURACIÓN ---

# ¡¡¡PASO MÁS IMPORTANTE!!!
# REEMPLAZA ESTA LISTA CON TU LISTA DE CLASES, EN EL ORDEN EXACTO EN QUE APARECE EN LABEL
CLASES_MAESTRA = [
    "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",
    # Ejemplo de cómo podrían seguir las clases 'be_'
    "be_black_8",
    "be_blue_10",
    "be_blue_2",
    "be_dred_15",
    "be_dred_7",
    "be_green_14",
    "be_green_6",
    "be_purple_13",
```

```
    "be_purple_5",
    "be_pink_4",
    "be_pink_12",
    "be_red_11",
    "be_red_3",
    "be_white",
    "be_yellow_1",
    "be_yellow_9",
]

# Ruta al JSON exportado desde Label Studio
LABEL_STUDIO_JSON_PATH = "Proyecto_Bolas_LS/ls-export.json"

# Ruta a la carpeta que contiene las imágenes originales que usaste en Label Studio
IMAGENES_DIR = "Proyecto_Bolas_LS/imagenes/"

# Carpeta de salida para las etiquetas YOLO (.txt)
YOLO_LABELS_OUTPUT_DIR = "Proyecto_Bolas_LS/final_yolo_labels/"

# Carpeta de salida para las imágenes de verificación
VERIFICATION_DIR = "Proyecto_Bolas_LS/verification_results/"

# --- FIN DE LA CONFIGURACIÓN ---


def convertir_y_verificar():
    # --- FASE 1: CONVERSIÓN DE LS-JSON A YOLO-TXT ---
    print("--- INICIANDO FASE 1: Conversión de Label Studio a formato YOLO ---")

    os.makedirs(YOLO_LABELS_OUTPUT_DIR, exist_ok=True)
    class_to_id = {name: i for i, name in enumerate(CLASES_MAESTRA)}

    with open(LABEL_STUDIO_JSON_PATH, "r") as f:
        data = json.load(f)

    for task in data:
        image_path = task["image"]
        image_filename = os.path.basename(unquote(image_path))

        # Limpieza del hash de LS
        parts = image_filename.split("-", 1)
        if len(parts) > 1 and len(parts[0]) != 1:
            image_filename = parts[1]

        yolo_txt_filename = os.path.splitext(image_filename)[0] + ".txt"
        output_path = os.path.join(YOLO_LABELS_OUTPUT_DIR, yolo_txt_filename)

        with open(output_path, "w") as f_yolo:
            if "label" in task and task["label"]:
                for annotation in task["label"]:
                    class_name = annotation["rectanglelabels"][0]
                    if class_name in class_to_id:
                        class_id = class_to_id[class_name]
```

```
                x_ls, y_ls = annotation["x"], annotation["y"]
                w_ls, h_ls = annotation["width"], annotation["height"]

                x_norm, y_norm = x_ls / 100.0, y_ls / 100.0
                w_norm, h_norm = w_ls / 100.0, h_ls / 100.0

                x_center = x_norm + (w_norm / 2)
                y_center = y_norm + (h_norm / 2)

                f_yolo.write(
                    f"{class_id} {x_center:.6f} {y_center:.6f} {w_norm:.6f} {h_norm:.6f}\n"
                )

print("--- FASE 1 COMPLETADA: Se han generado los archivos .txt ---")

# --- FASE 2: VERIFICACIÓN VISUAL ---
print("\n--- INICIANDO FASE 2: Verificación visual de las etiquetas generadas ---")
os.makedirs(VERIFICATION_DIR, exist_ok=True)

label_files = os.listdir(YOLO_LABELS_OUTPUT_DIR)
archivos_a_verificar = random.sample(label_files, min(5, len(label_files)))

for label_file in archivos_a_verificar:
    base_name = os.path.splitext(label_file)[0]

    image_path_to_verify = None
    for ext in [".jpg", ".jpeg", ".png"]:
        potential_path = os.path.join(IMAGENES_DIR, base_name + ext)
        if os.path.exists(potential_path):
            image_path_to_verify = potential_path
            break

    if not image_path_to_verify:
        continue

    image = cv2.imread(image_path_to_verify)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    h, w, _ = image.shape

    fig, ax = plt.subplots(1, figsize=(12, 8))
    ax.imshow(image)
    ax.set_title(f"Verificación de: {os.path.basename(image_path_to_verify)}")

    label_path = os.path.join(YOLO_LABELS_OUTPUT_DIR, label_file)
    with open(label_path, "r") as f:
        for line in f:
            parts = line.strip().split()
            class_id, x_center, y_center, width, height = map(float, parts)

            box_w = width * w
            box_h = height * h
            x1 = (x_center * w) - (box_w / 2)
            y1 = (y_center * h) - (box_h / 2)
```

```
            class_name = CLASES_MAESTRA[
                int(class_id)
            ]  # Usamos la misma lista maestra

            rect = patches.Rectangle(
                (x1, y1),
                box_w,
                box_h,
                linewidth=2,
                edgecolor="cyan",
                facecolor="none",
            )
            ax.add_patch(rect)
            plt.text(
                x1,
                y1 - 10,
                class_name,
                color="cyan",
                fontsize=12,
                bbox=dict(facecolor="black", alpha=0.7),
            )

        plt.axis("off")
        output_save_path = os.path.join(
            VERIFICATION_DIR, os.path.basename(image_path_to_verify)
        )
        plt.savefig(output_save_path, bbox_inches="tight", pad_inches=0)
        plt.show()
        print(f"Imagen de verificación guardada en: {output_save_path}")

    print("--- FASE 2 COMPLETADA ---")


if __name__ == "__main__":
    convertir_y_verificar()
```

RandomForestClassifier/generar_meta_dataset.py

```
# generar_meta_dataset.py
import os

import pandas as pd
from tqdm import tqdm  # Para una bonita barra de progreso (pip install tqdm)
from ultralytics import YOLO

# --- 1. CONFIGURACIÓN ---

# Rutas a los datasets
DATASET_HIBRIDO_DIR = "detect_balls/dataset_hibrido/"
```

```
DATASET_ORIGINAL_UNIFICADO_DIR = (
    "detect_balls/dataset_hibrido/"  # Necesario para el ground truth
)

# Ruta a tu mejor modelo híbrido
MODEL_PATH = "detect_balls/runs/Modelo_Hibrido_v1/weights/best.pt"

# Nombre del archivo de salida
OUTPUT_CSV_PATH = "RandomForestClassifier/meta_dataset.csv"

# Listas de clases que ya tenemos definidas
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",
    # "be_purple_5",
    "be_pink_4",
    "be_pink_12",
    "be_red_11",
    "be_yellow_9",
}
DELATORES_CLASSIC = {
    "blue_10",
    "dred_15",
    "green_14",
```

```
    "orange_13",
    "orange_5",
    # "purple_12",
    # "purple_4",
    "red_11",
    "yellow_9",
}

# Lista original de 32 clases para determinar el target
CLASES_MAESTRA_ORIGINAL = [
    "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_black_8",
    "be_blue_10",
    "be_blue_2",
    "be_dred_15",
    "be_dred_7",
    "be_green_14",
    "be_green_6",
    "be_purple_13",
    "be_purple_5",
    "be_pink_4",
    "be_pink_12",
    "be_red_11",
    "be_red_3",
    "be_white",
    "be_yellow_1",
    "be_yellow_9",
]

# --- FIN DE LA CONFIGURACIÓN ---


def get_ground_truth_context(label_path_original):
    """
    Lee el archivo de etiqueta del dataset original (32 clases) para determinar el contexto real.
    Devuelve 1 si es 'black_edition', 0 si es 'classic'.
    """
    try:
```

```
        with open(label_path_original, "r") as f:
            for line in f:
                id_original = int(float(line.strip().split()[0]))
                nombre_clase_original = CLASES_MAESTRA_ORIGINAL[id_original]
                if nombre_clase_original.startswith("be_"):
                    return 1  # Es Black Edition
    except (IOError, IndexError, ValueError):
        return 0  # Si hay error o no se encuentra, asumimos classic
    return 0  # Si termina el bucle sin encontrar delatores, es Classic


def generar_dataset():
    print(f"Cargando modelo desde: {MODEL_PATH}")
    model = YOLO(MODEL_PATH)

    all_features = []

    # Iterar sobre train, valid y test
    for split in ["train", "valid", "test"]:
        images_dir = os.path.join(DATASET_HIBRIDO_DIR, "images", split)
        labels_original_dir = os.path.join(
            DATASET_ORIGINAL_UNIFICADO_DIR, "labels", split
        )

        if not os.path.isdir(images_dir):
            continue

        print(f"\nProcesando el conjunto de datos: {split}")
        for image_filename in tqdm(os.listdir(images_dir)):
            if not image_filename.lower().endswith((".png", ".jpg", ".jpeg")):
                continue

            image_path = os.path.join(images_dir, image_filename)
            base_name = os.path.splitext(image_filename)[0]

            # 1. Realizar predicción con el modelo híbrido
            results = model.predict(image_path, verbose=False, conf=0.25)
            result = results[0]

            # 2. Inicializar y extraer características
            puntuacion_be = 0.0
            puntuacion_classic = 0.0
            confianza_total = 0.0
            counts = {f"count_{cls}": 0 for cls in CLASES_HIBRIDAS}

            if result.boxes:
                for box in result.boxes:
                    class_id = int(box.cls[0])
                    clase = CLASES_HIBRIDAS[class_id]
                    conf = float(box.conf[0])

                    counts[f"count_{clase}"] += 1
                    confianza_total += conf
```

```
                if clase in DELATORES_BE:
                    puntuacion_be += conf
                elif clase in DELATORES_CLASSIC:
                    puntuacion_classic += conf

            total_detecciones = len(result.boxes)
            confianza_media = (
                confianza_total / total_detecciones if total_detecciones > 0 else 0
            )

            # 3. Determinar el Ground Truth (el contexto real)
            label_original_path = os.path.join(labels_original_dir, base_name + ".txt")
            target = get_ground_truth_context(label_original_path)

            # 4. Guardar la fila de datos
            row_data = {
                "image_path": image_path,
                "puntuacion_be": puntuacion_be,
                "puntuacion_classic": puntuacion_classic,
                "total_detecciones": total_detecciones,
                "confianza_media": confianza_media,
                "target": target,  # 0 para classic, 1 para black_edition
            }
            row_data.update(counts)
            all_features.append(row_data)

    # 5. Crear y guardar el DataFrame
    print("\nCreando el archivo CSV final...")
    df = pd.DataFrame(all_features)
    df.to_csv(OUTPUT_CSV_PATH, index=False)

    print(f"¡Éxito! Meta-dataset guardado en '{OUTPUT_CSV_PATH}' con {len(df)} filas.")
    print("\nResumen del dataset:")
    print(df["target"].value_counts())


if __name__ == "__main__":
    # Necesitarás instalar pandas y tqdm si no los tienes
    # pip install pandas tqdm
    generar_dataset()
```

RandomForestClassifier/entrenar_clasificador_contexto.py

```
# entrenar_clasificador_contexto.py
import joblib
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
```

```
# --- 1. CONFIGURACIÓN ---
META_DATASET_PATH = "RandomForestClassifier/meta_dataset.csv"
MODELO_SALIDA_PATH = "RandomForestClassifier/context_classifier.joblib"
# --- FIN DE LA CONFIGURACIÓN ---


def entrenar_meta_modelo():
    # Cargar el dataset que hemos creado
    try:
        df = pd.read_csv(META_DATASET_PATH)
        print(
            f"Dataset cargado exitosamente desde '{META_DATASET_PATH}'. Contiene {len(df)} filas."
        )
    except FileNotFoundError:
        print(f"Error: No se encontró el archivo '{META_DATASET_PATH}'.")
        return

    # Separar las características (X) del objetivo (y)
    # Usamos todas las columnas como características excepto el path de la imagen y el target
    X = df.drop(columns=["image_path", "target"])
    y = df["target"]

    # Dividir los datos en conjuntos de entrenamiento y prueba para evaluar nuestro clasificador
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42, stratify=y
    )

    print(
        f"Datos divididos: {len(X_train)} para entrenamiento, {len(X_test)} para prueba."
    )

    # Crear y entrenar el modelo Random Forest
    # class_weight='balanced' es la clave para manejar el desequilibrio de clases
    print("\nIniciando entrenamiento del clasificador de contexto (Random Forest)...")
    model = RandomForestClassifier(
        n_estimators=100, random_state=42, class_weight="balanced", n_jobs=-1
    )
    model.fit(X_train, y_train)
    print("¡Entrenamiento completado!")

    # Evaluar el modelo
    print("\n--- Evaluación del Clasificador de Contexto ---")
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"Precisión (Accuracy): {accuracy:.4f}")

    print("\nInforme de Clasificación:")
    print(
        classification_report(
            y_test, y_pred, target_names=["Classic (0)", "Black Edition (1)"]
        )
```

```
    )

    print("Matriz de Confusión:")
    print(confusion_matrix(y_test, y_pred))

    # Guardar el modelo entrenado para usarlo en la inferencia final
    joblib.dump(model, MODELO_SALIDA_PATH)
    print(f"\n¡Modelo clasificador de contexto guardado en '{MODELO_SALIDA_PATH}'!")


if __name__ == "__main__":
    entrenar_meta_modelo()
```

RandomForestClassifier/inferencia_sistema_completo.py

```
# inferencia_sistema_completo.py
import os

import cv2
import joblib
import pandas as pd
from ultralytics import YOLO

# --- 1. CONFIGURACIÓN ---
base_project_dir = os.path.join(".", "detect_balls")

# --- Rutas de los Modelos ---
DETECTION_MODEL_PATH = "detect_balls/runs/Modelo_Hibrido_v1/weights/best.pt"
CONTEXT_MODEL_PATH = "RandomForestClassifier/context_classifier.joblib"

# --- Rutas de Archivos ---
TEST_IMAGES_DIR = os.path.join(base_project_dir, "tests", "Mix")
OUTPUT_DIR = os.path.join(base_project_dir, "tests", "results_mix")

# --- Parámetros de Inferencia ---
CONFIDENCE_THRESHOLD = 0.4

# --- Listas de Clases y Reglas de Lógica ---
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",
]
CLASES_COMPARTIDAS = {
    "black_8",
    "blue_2",
    "dred_7",
    "green_6",
    "red_3",
    "white",
    "yellow_1",
}
DELATORES_BE = {
    "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_CLASSIC = {
    "blue_10",
    "dred_15",
    "green_14",
    "orange_13",
    "orange_5",
    # "purple_12",
    # "purple_4",
    "red_11",
    "yellow_9",
}
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",
    "purple_4": "be_pink_4",
```

```
    "purple_12": "be_pink_12",
    "purple_4": "be_purple_5",
    "purple_12": "be_purple_13",
}
# --- FIN DE LA CONFIGURACIÓN ---


def extraer_features_para_contexto(detecciones_yolo):
    """Extrae las características de las detecciones para alimentar al clasificador de contexto."""
    puntuacion_be = sum(
        d["conf"] for d in detecciones_yolo if d["clase"] in DELATORES_BE
    )
    puntuacion_classic = sum(
        d["conf"] for d in detecciones_yolo if d["clase"] in DELATORES_CLASSIC
    )

    counts = {f"count_{cls}": 0 for cls in CLASES_HIBRIDAS}
    confianza_total = 0.0
    for d in detecciones_yolo:
        counts[f"count_{d['clase']}"] += 1
        confianza_total += d["conf"]

    total_detecciones = len(detecciones_yolo)
    confianza_media = (
        confianza_total / total_detecciones if total_detecciones > 0 else 0
    )

    # Crear un DataFrame de una sola fila con las features en el orden correcto
    features = {
        "puntuacion_be": puntuacion_be,
        "puntuacion_classic": puntuacion_classic,
        "total_detecciones": total_detecciones,
        "confianza_media": confianza_media,
    }
    features.update(counts)

    # Asegurar el orden de las columnas
    column_order = [
        "puntuacion_be",
        "puntuacion_classic",
        "total_detecciones",
        "confianza_media",
    ] + [f"count_{cls}" for cls in CLASES_HIBRIDAS]

    return pd.DataFrame([features], columns=column_order)


def main():
    os.makedirs(OUTPUT_DIR, exist_ok=True)

    # Cargar los dos modelos
    print("Cargando modelo de detección (el 'Ojo')...")
    detection_model = YOLO(DETECTION_MODEL_PATH)
```

```
print("Cargando modelo de contexto (el 'Cerebro')...")
context_model = joblib.load(CONTEXT_MODEL_PATH)

image_files = [
    f
    for f in os.listdir(TEST_IMAGES_DIR)
    if f.lower().endswith((".png", ".jpg", ".jpeg"))
]
print(
    f"\nIniciando inferencia del sistema completo en {len(image_files)} imágenes..."
)

for image_filename in image_files:
    test_image_path = os.path.join(TEST_IMAGES_DIR, image_filename)
    print(f"\n--- Procesando: {image_filename} ---")

    # 1. El "Ojo" (YOLO) hace su trabajo
    results = detection_model.predict(
        test_image_path, verbose=False, conf=CONFIDENCE_THRESHOLD
    )
    detecciones_yolo = [
        {
            "clase": CLASES_HIBRIDAS[int(box.cls[0])],
            "conf": float(box.conf[0]),
            "box_xyxy": box.xyxy[0],
        }
        for box in results[0].boxes
    ]

    if not detecciones_yolo:
        print("No se detectaron objetos.")
        continue

    # 2. El "Cerebro" (Random Forest) determina el contexto
    features = extraer_features_para_contexto(detecciones_yolo)
    contexto_pred_id = context_model.predict(features)[0]
    contexto = "black_edition" if contexto_pred_id == 1 else "classic"
    print(f"  > Contexto decidido por el 'Cerebro' de ML: '{contexto}'")

    # 3. Se aplica la lógica de corrección final
    detecciones_finales = []
    for deteccion in detecciones_yolo:
        # ... (código de corrección y simplificación de etiquetas) ...
        etiqueta_yolo = deteccion["clase"]
        etiqueta_corregida = etiqueta_yolo
        if contexto == "classic" and etiqueta_yolo in DELATORES_BE:
            etiqueta_corregida = MAPA_CORRECCION_CONTEXTUAL.get(
                etiqueta_yolo, etiqueta_yolo
            )
        elif contexto == "black_edition" and etiqueta_yolo in DELATORES_CLASSIC:
            etiqueta_corregida = MAPA_CORRECCION_CONTEXTUAL.get(
                etiqueta_yolo, etiqueta_yolo
            )
```

```
            etiqueta_simple = etiqueta_corregida.replace("be_", "")
            if contexto == "black_edition" and etiqueta_simple in CLASES_COMPARTIDAS:
                etiqueta_simple = "be_" + etiqueta_simple

            deteccion["etiqueta_final"] = etiqueta_simple
            detecciones_finales.append(deteccion)

        # 4. Se visualizan los resultados finales
        # ... (código de visualización igual que antes) ...
        img = cv2.imread(test_image_path)
        for d in detecciones_finales:
            x1, y1, x2, y2 = map(int, d["box_xyxy"])
            label = f"{d['etiqueta_final']} {d['conf']:.2f}"
            cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
            (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
            cv2.rectangle(img, (x1, y1 - 20), (x1 + w, y1), (0, 255, 0), -1)
            cv2.putText(
                img, label, (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2
            )

        output_path = os.path.join(OUTPUT_DIR, f"resultado_{image_filename}")
        cv2.imwrite(output_path, img)
        print(f"  > Resultado final guardado en: {output_path}")

    print("\n\n¡PROYECTO FINALIZADO CON ÉXITO!")


if __name__ == "__main__":
    main()
```

show_table/show_pygame.py

```
import sys

import pygame

# --- Tus datos (simulados aquí, pero tú los tendrías de la salida de YOLO) ---
# Esta es la estructura que tu script de YOLO ya está produciendo.
# Pega aquí la salida de tu variable 'detected_balls_on_table_plane'
# o carga los datos si los guardaste en un archivo.
# Por ahora, usaré los datos de tu salida como ejemplo:
detected_balls_on_table_plane = [
    {
        "clase": "red_3",
        "confianza": 0.77,
        "centro_imagen_px": (432.0, 218.5),
        "centro_mesa_units": (303.2, 117.2),
    },
    {
```

```
        "clase": "green_6",
        "confianza": 0.71,
        "centro_imagen_px": (346.5, 591.7),
        "centro_mesa_units": (225.7, 457.4),
    },
    {
        "clase": "green_6",
        "confianza": 0.67,
        "centro_imagen_px": (443.8, 264.6),
        "centro_mesa_units": (313.7, 159.3),
    },
    {
        "clase": "red_3",
        "confianza": 0.43,
        "centro_imagen_px": (1139.9, 101.4),
        "centro_mesa_units": (939.6, 9.6),
    },
    {
        "clase": "black_8",
        "confianza": 0.43,
        "centro_imagen_px": (285.4, 166.1),
        "centro_mesa_units": (171.3, 69.5),
    },
    {
        "clase": "white",
        "confianza": 0.31,
        "centro_imagen_px": (653.4, 38.5),
        "centro_mesa_units": (502.7, -47.6),
    },
]

# --- Dimensiones y escala (ajusta según tu mesa ideal usada para la homografía) ---
TABLE_VIEW_WIDTH_UNITS = 1000  # Las unidades que usaste para destination_corners
TABLE_VIEW_HEIGHT_UNITS = 500

PYGAME_WINDOW_WIDTH = 800  # Ancho de la ventana de Pygame en píxeles
PYGAME_WINDOW_HEIGHT = 400  # Alto de la ventana de Pygame en píxeles

# Calcular factores de escala
# Si TABLE_VIEW_WIDTH_UNITS es 0, evita división por cero
SCALE_X = PYGAME_WINDOW_WIDTH / TABLE_VIEW_WIDTH_UNITS if TABLE_VIEW_WIDTH_UNITS else 0
SCALE_Y = (
    PYGAME_WINDOW_HEIGHT / TABLE_VIEW_HEIGHT_UNITS if TABLE_VIEW_HEIGHT_UNITS else 0
)

# --- Colores ---
COLOR_FELT_GREEN = (34, 139, 34)  # Un verde más de billar
COLOR_BLACK = (0, 0, 0)
COLOR_RED = (255, 0, 0)
COLOR_YELLOW = (255, 255, 0)
COLOR_GREEN = (0, 255, 0)
COLOR_BLUE = (0, 0, 255)
COLOR_PURPLE = (128, 0, 128)
```

```
COLOR_ORANGE = (255, 165, 0)
COLOR_WHITE = (255, 255, 255)
# Mapeo de nombres de clase a colores (¡AJUSTA SEGÚN TUS CLASS_NAMES!)
BALL_PYGAME_COLORS = {
    "white": COLOR_WHITE,
    "yellow_1": COLOR_YELLOW,
    "yellow_9": COLOR_YELLOW,  # Asumiendo mismo color base
    "blue_2": COLOR_BLUE,
    "blue_10": COLOR_BLUE,
    "red_3": COLOR_RED,
    "red_11": COLOR_RED,
    "dred_7": COLOR_RED,
    "dred_15": COLOR_RED,  # 'dred' as red
    "purple_4": COLOR_PURPLE,
    "purple_12": COLOR_PURPLE,  # Suponiendo que tienes purple_4, si no, elimina
    "orange_5": COLOR_ORANGE,
    "orange_13": COLOR_ORANGE,
    "green_6": COLOR_GREEN,
    "green_14": COLOR_GREEN,
    "black_8": COLOR_BLACK,
    # Añade o ajusta según las clases exactas que tengas
}
DEFAULT_BALL_COLOR = (128, 128, 128)  # Gris para bolas no mapeadas

# Radio de las bolas en unidades de la mesa (aprox. 5.715 cm diámetro)
# Usaremos un radio fijo en píxeles para la visualización para simplificar
BALL_RADIUS_PYGAME = 10  # Radio de la bola en píxeles en la ventana de Pygame


# --- Inicialización de Pygame ---
pygame.init()
screen = pygame.display.set_mode((PYGAME_WINDOW_WIDTH, PYGAME_WINDOW_HEIGHT))
pygame.display.set_caption("Visualización de Mesa de Billar con Bolas Detectadas")
font = pygame.font.Font(None, 24)  # Para mostrar texto (opcional)

# --- Bucle principal de Pygame ---
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:  # Salir con la tecla ESC
                running = False

    # --- Lógica de Dibujo ---
    screen.fill(COLOR_FELT_GREEN)  # Dibuja el fondo de la mesa

    # Dibuja las bolas
    for ball_data in detected_balls_on_table_plane:
        clase_bola = ball_data["clase"]
        x_mesa, y_mesa = ball_data["centro_mesa_units"]
```

```
        # Convertir coordenadas de la mesa a coordenadas de Pygame
        # (0,0) en la mesa ideal → (0,0) en la ventana de Pygame
        x_pygame = int(x_mesa * SCALE_X)
        y_pygame = int(y_mesa * SCALE_Y)

        ball_color = BALL_PYGAME_COLORS.get(clase_bola, DEFAULT_BALL_COLOR)

        # Dibujar la bola
        pygame.draw.circle(screen, ball_color, (x_pygame, y_pygame), BALL_RADIUS_PYGAME)

        # Opcional: dibujar el nombre de la clase cerca de la bola
        # text_surface = font.render(clase_bola, True, COLOR_BLACK)
        # screen.blit(text_surface, (x_pygame + BALL_RADIUS_PYGAME, y_pygame))

    pygame.display.flip()  # Actualiza la pantalla completa

pygame.quit()
sys.exit()
```