```
import numpy as np
import pandas as pd
import rasterio
from rasterio.mask import mask
from rasterio.enums import Resampling
from rasterio.features import shapes
from rasterio.warp import calculate_default_transform, reproject, Resampling as WarpResampling
from shapely.geometry import shape
import geopandas as gpd
import os
import io
import base64
import matplotlib.pyplot as plt

# --- 1. LEYENDA Y COLORES ---
LAND_COVER_LEGEND = {
    1: "Zonas Urbanas",
    2: "Zonas industriales o comerciales",
    3: "Zonas degradadas -canteras, escombreras, minas",
    4: "Zonas Verdes artificializadas No Agrícolas",
    5: "Cultivos transitorios",
    6: "Cultivos permanentes",
    7: "Pastos",
    8: "Areas Agrícolas Heterogéneas",
    9: "Bosque",
    10: "Vegetación Herbácea / Arbustiva",
    11: "Areas abiertas sin o con poca cobertura vegetal",
    12: "Humedales",
    13: "Agua / Cuerpos de Agua"
}

LAND_COVER_COLORS = {
    1: "#A9A9A9",   # Gris
    2: "#FFFF00",   # Amarillo
    3: "#FFA500",   # Naranja
    4: "#FFD700",   # Oro
    5: "#006400",   # Verde Oscuro
    6: "#32CD32",   # Verde Lima
    7: "#F4A460",   # Arena
    8: "#2E8B57",   # Verde Mar
    9: "#228B22",   # Verde Forestal
    10: "#9ACD32",  # Verde Claro
    11: "#8B4513",  # Café
    12: "#00CED1",  # Turquesa
    13: "#0000FF"   # Azul Puro
}
```

```
# --- 2. FUNCIONES AUXILIARES ---

def get_pixel_area_in_km2(transform, crs, height, width):
    """
    Calcula el área de un píxel en km². Detecta si el mapa está en Grados o Metros.
    Soluciona el problema de 'Área: 0.00 km²'.
    """
    # Tamaño del pixel en las unidades del mapa
    px_w = abs(transform[0])
    px_h = abs(transform[4])

    # Si es Geográfico (WGS84, EPSG:4326), las unidades son GRADOS.
    if crs.is_geographic:
        # Usamos una latitud promedio (ej. 6.5 para Antioquia) para el factor de longitud
        lat_factor = 111.32
        lon_factor = 110.5 # Aprox en el trópico

        area_km2 = (px_w * lon_factor) * (px_h * lat_factor)
    else:
        # Si es Proyectado (Metros), convertimos m² a km²
        area_km2 = (px_w * px_h) / 1_000_000

    return area_km2

# --- 3. PROCESAMIENTO ROBUSTO (CORRECCIÓN PROYECCIONES) ---

def process_land_cover_raster(raster_path, gdf_mask=None, scale_factor=1):
    """
    Lee el raster y lo recorta por la máscara.
    CORRECCIÓN CRÍTICA: Asegura que la máscara se reproyecte al CRS del Raster antes de cortar.
    """
    if not os.path.exists(raster_path):
        return None, None, None, None

    with rasterio.open(raster_path) as src:
        nodata = src.nodata
        crs = src.crs

        if gdf_mask is not None:
            # 1. Verificar y corregir proyección de la máscara
            # El raster manda. Si el raster es 3116, la máscara debe ser 3116.
            if gdf_mask.crs != src.crs:
                try:
                    gdf_proj = gdf_mask.to_crs(src.crs)
                except:
```

```
                # Si falla (ej: CRS origen desconocido), forzamos reproyección si asumimos LatLon
                gdf_mask.set_crs("EPSG:4326", inplace=True, allow_override=True)
                gdf_proj = gdf_mask.to_crs(src.crs)
            else:
                gdf_proj = gdf_mask

            # 2. Recortar
            try:
                # crop=True ajusta la ventana al tamaño de la máscara
                out_image, out_transform = mask(src, gdf_proj.geometry, crop=True)
                data = out_image[0]
            except ValueError:
                # Pasa si la geometría no se superpone con el raster
                return None, None, None, None
        else:
            # Modo Regional: Aplicamos scale_factor solo si se pide
            new_height = int(src.height / scale_factor)
            new_width = int(src.width / scale_factor)
            data = src.read(1, out_shape=(new_height, new_width), resampling=Resampling.nearest)
            out_transform = src.transform * src.transform.scale(
                (src.width / data.shape[-1]), (src.height / data.shape[-2])
            )

    return data, out_transform, crs, nodata

def calculate_land_cover_stats(data, transform, crs, nodata, manual_area_km2=None):
    """Calcula estadísticas corrigiendo el problema de área cero."""
    if data is None: return pd.DataFrame(), 0

    valid_pixels = data[(data != nodata) & (data > 0)]
    if valid_pixels.size == 0: return pd.DataFrame(), 0

    # Usamos la función inteligente para obtener el área real
    pixel_area_km2 = get_pixel_area_in_km2(transform, crs, data.shape[0], data.shape[1])

    unique, counts = np.unique(valid_pixels, return_counts=True)
    calc_total_area = counts.sum() * pixel_area_km2

    # Ajuste fino si tenemos área vectorial (para que cuadre exacto con el polígono)
    final_total_area = calc_total_area
    factor = 1.0

    # Solo aplicamos factor de corrección si la diferencia es razonable (ej. < 20%)
    # Si difiere mucho, confiamos en el cálculo raster
    if manual_area_km2 and manual_area_km2 > 0 and calc_total_area > 0:
        if 0.8 < (manual_area_km2 / calc_total_area) < 1.2:
```

```
            final_total_area = manual_area_km2
            factor = manual_area_km2 / calc_total_area

    rows = []
    for val, count in zip(unique, counts):
        area = (count * pixel_area_km2) * factor
        pct = (area / final_total_area) * 100
        rows.append({
            "ID": val,
            "Cobertura": LAND_COVER_LEGEND.get(val, f"Clase {val}"),
            "Área (km²)": area,
            "%": pct,
            "Color": LAND_COVER_COLORS.get(val, "#808080")
        })

    return pd.DataFrame(rows).sort_values("%", ascending=False), final_total_area

def get_raster_img_b64(data, nodata):
    """Genera imagen PNG Base64 optimizada."""
    if data is None: return ""

    rgba = np.zeros((data.shape[0], data.shape[1], 4), dtype=np.uint8)
    for val, hex_c in LAND_COVER_COLORS.items():
        if isinstance(hex_c, str):
            hex_c = hex_c.lstrip('#')
            r, g, b = tuple(int(hex_c[i:i+2], 16) for i in (0, 2, 4))

            # Vectorización rápida numpy
            mask_val = (data == val)
            rgba[mask_val, 0] = r
            rgba[mask_val, 1] = g
            rgba[mask_val, 2] = b
            rgba[mask_val, 3] = 200 # Transparencia ligera

    rgba[(data == 0) | (data == nodata), 3] = 0
    image_data = io.BytesIO()
    plt.imsave(image_data, rgba, format='png')
    image_data.seek(0)
    return f"{base64.b64encode(image_data.read()).decode('utf-8')}"

# --- 4. VECTORIZACIÓN SIMPLIFICADA (ESTABILIDAD) ---

def vectorize_raster_optimized(data, transform, crs, nodata, max_shapes=3000):
    """
    Vectoriza el raster y lo reproyecta a EPSG:4326 para visualización web.
    """
```

```
    if data is None: return gpd.GeoDataFrame()

    mask_arr = (data != nodata) & (data != 0)

    # Shapes genera geometrías en las coordenadas del raster (posiblemente Metros EPSG:3116)
    shapes_gen = shapes(data, mask=mask_arr, transform=transform)

    geoms = []
    values = []
    count = 0

    for geom, val in shapes_gen:
        if count > max_shapes: break

        # Convertimos diccionario geojson a objeto shapely
        s_geom = shape(geom)

        # Simplificación para rendimiento web
        s_geom_simp = s_geom.simplify(tolerance=10, preserve_topology=True) # Tolerancia en metros si es
proyectado
        if s_geom_simp.is_empty: continue

        geoms.append(s_geom_simp)
        values.append(val)
        count += 1

    if not geoms: return gpd.GeoDataFrame()

    # Creamos el GeoDataFrame con el CRS original del raster
    gdf = gpd.GeoDataFrame({'ID': values}, geometry=geoms, crs=crs)

    # Mapear nombres y colores
    gdf['Cobertura'] = gdf['ID'].map(lambda x: LAND_COVER_LEGEND.get(int(x), f"Clase {int(x)}"))
    gdf['Color'] = gdf['ID'].map(lambda x: LAND_COVER_COLORS.get(int(x), "#808080"))

    # REPROYECCIÓN OBLIGATORIA A LAT/LON PARA FOLIUM
    # Esto soluciona que el mapa vectorial no aparezca
    if gdf.crs is not None and gdf.crs.to_string() != "EPSG:4326":
        try:
            gdf = gdf.to_crs("EPSG:4326")
        except:
            pass # Si falla, enviamos original y esperamos lo mejor

    return gdf

def generate_legend_html():
```

```
    """Leyenda HTML fija."""
    html = """
    <div style="position: fixed; bottom: 30px; left: 30px; z-index:9999;
        background-color: white; padding: 10px; border: 2px solid #ccc;
        border-radius: 5px; font-size: 11px; max-height: 250px; overflow-y: auto;">
        <b>Leyenda</b><br>
    """
    for id_cov, name in sorted(LAND_COVER_LEGEND.items()):
        color = LAND_COVER_COLORS.get(id_cov, "#808080")
        html += f'<div style="display:flex; align-items:center; margin-bottom:2px;"><span
style="background:{color}; width:12px; height:12px; display:inline-block; margin-right:5px; border:1px solid
#333;"></span>{name}</div>'
    html += "</div>"
    return html

def get_tiff_bytes(data, transform, crs, nodata):
    if data is None: return None
    mem_file = io.BytesIO()
    with rasterio.open(
        mem_file, 'w', driver='GTiff', height=data.shape[0], width=data.shape[1],
        count=1, dtype=data.dtype, crs=crs, transform=transform, nodata=nodata
    ) as dst:
        dst.write(data, 1)
    mem_file.seek(0)
    return mem_file

# --- MÉTODOS SCS ---
def calculate_weighted_cn(df_stats, cn_config):
    if df_stats.empty: return 0
    cn_pond = 0; total_pct = 0
    for _, row in df_stats.iterrows():
        cob = str(row["Cobertura"]) # Asegurar string
        pct = row["%"]
        val = 85 # Default
        if "Bosque" in cob: val = cn_config['bosque']
        elif "Pasto" in cob or "Herbácea" in cob: val = cn_config['pasto']
        elif "Urban" in cob: val = cn_config['urbano']
        elif "Agua" in cob: val = 100
        elif "Suelo" in cob or "Degradada" in cob: val = cn_config['suelo']
        elif "Cultivo" in cob: val = cn_config['cultivo']
        cn_pond += val * pct / 100
        total_pct += pct
    return (cn_pond / total_pct) * 100 if total_pct > 0 else 0

def calculate_scs_runoff(cn, ppt_mm):
    if cn >= 100: return ppt_mm
```

```
    if cn <= 0: return 0
    s = (25400 / cn) - 254; ia = 0.2 * s
    return ((ppt_mm - ia) ** 2) / (ppt_mm - ia + s) if ppt_mm > ia else 0

def get_land_cover_at_point(lat, lon, raster_path):
    if not os.path.exists(raster_path):
        return "Raster no encontrado"
    try:
        # Aquí el raster manda. Si el raster es proyectado (metros),
        # debemos reproyectar el punto (lat, lon) a las coordenadas del raster.
        with rasterio.open(raster_path) as src:
            if src.crs.to_string() != "EPSG:4326":
                # Transformación rápida de coordenadas
                from pyproj import Transformer
                transformer = Transformer.from_crs("EPSG:4326", src.crs, always_xy=True)
                x, y = transformer.transform(lon, lat)
                coords = [(x, y)]
            else:
                coords = [(lon, lat)]

            val_gen = src.sample(coords)
            try:
                val = next(val_gen)[0]
                return LAND_COVER_LEGEND.get(int(val), f"Clase {val}")
            except StopIteration:
                return "Fuera de rango"

    except Exception as e:
        return f"Error: {str(e)}"
```