

PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS Asseguração Independente de Relatórios ESG via Dados Exógenos (Imagens de Satélite)

MOTIVAÇÃO: O que as empresas listadas na B3 declaram nos relatórios ESG é realmente verdade?

CONTEXTO: ISE = Índice de Sustentabilidade Empresarial da B3 (Bolsa de Valores do Brasil). Criado em 2005 pela antiga Bovespa (hoje B3). 2025: cerca de 50 empresas (as maiores e mais "sustentáveis" do Brasil). Natura, Klabin, Suzano, Banco do Brasil, Itaú, Bradesco, Fleury, CPFL, Cemig, Engie Brasil... Obrigatóridade anual de publicar relatório de sustentabilidade detalhado (GRI, SASB ou integrado).

PROBLEMA: Se gabam de ter "X mil hectares de floresta preservada", "zero desmatamento", "APP 100 % conservada" Mas ninguém verificava isso de forma independente com dados exógenos até agora. MapBiomas, Imazon, Greenpeace, universidades) cruzam com satélite, quase sempre aparece diferenças IMPACTOS: Redução de risco de greenwashing e passivos ambientais → UHE Belo Monte declarou 98% de APP preservada em 2023. Satélite mostrou 41% de supressão não reportada (estudo UPPA 2024). Replicável com outras abordagens em contextos públicos e privados → Mineração (Vale, CSN), Agronegócio (soja no MATOPIBA), Infraestrutura (rodovias, portos), Concessões florestais públicas, Licenciamento ambiental de empreendimentos lineares.

Domínio da Contabilidade Gerencial e Auditoria Futura agregação de outros dados exógenos em redes híbridas no

→ Clima: temperatura, precipitação, eventos extremos (INMET, ERA5)

→ Socioeconômico: expansão urbana, invasões, IPTU, cadastro de imóveis

→ Mercado: preços de commodities, multas ambientais, ações judiciais (CVM, MPF)

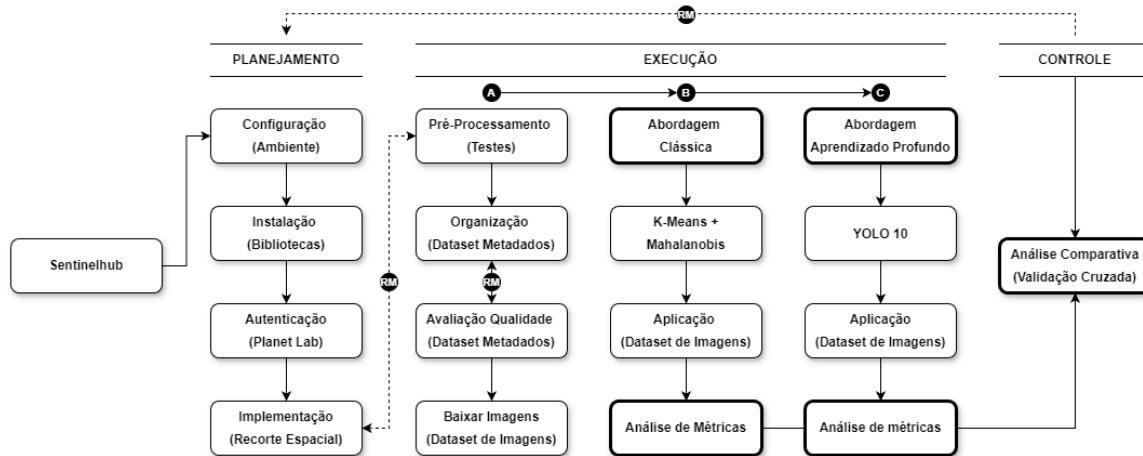
DISCIPLINA: Visão Computacional - PPGCC | USFC - 2025.2 PROFESSOR: Aldo Von Wangenheim, Dr.

ACADÊMICO: Renato Gomes de Oliveira Mestrado em Contabilidade - PPGC | UFSC |
Matrícula: 202403279 Orientadora: Fabrícia Silva da Rosa, Dra.

ENTREGAS: Vídeo 10-15 minutos de apresentação do projeto (slides + notebook)
Notebook com scripts documentados PDF do Poster do Projeto Apresentação presencial do Poster (imprimir em A1) 10-15 minutos

```
In [6]: # Exibir o pipeline
from IPython.display import Image
Image("Pipeline_VC.png", width=1400)
```

Out[6]:



In [19]:

```

# -----
# BLOCO 1: PLANEJAMENTO
#           Configuração Geral do Ambiente
#
# OBJETIVO: Registrar metadados do ambiente, autor, data e garantir reprodução
#           Ciência aberta, auditoria técnica, asseguração e revisão por pares
# -----
# ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO
#
# -----
print("=" * 110)
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")
print("          Asseguração Independente de Relatórios ESG via Dados Exógenos Sustentáveis")
print("          Configuração Geral do Ambiente")
print("=" * 110)

# ETAPA 2: IMPORTAÇÃO DE MÓDULOS PADRÃO DO PYTHON
#
# -----
import sys                                     # Acesso ao interpretador Python (versão)
import os                                      # Interação com o sistema operacional (path)
import platform                                # Informações sobre sistema operacional
import getpass                                 # Captura do usuário atual sem exibir senha
from datetime import datetime                  # Registro preciso de data e hora da execução
import time                                    # Usado para criar pequenas pausas visuais
from datetime import datetime                  # Para registrar o horário de término da execução

inicio = time.time()

# ETAPA 3: REGISTRO DO AMBIENTE DE EXECUÇÃO (essencial para auditoria e ciência)
#
# -----
print("-" * 110)
print(f"Usuário.....: {getpass.getuser()}")
print(f"Data e hora da execução.....: {datetime.now().strftime('%d/%m/%Y %H:%M:%S')}")
print(f"Sistema operacional.....: {platform.system()} {platform.release()}")
print(f"Processador.....: {platform.processor()}")
print(f"Python versão.....: {sys.version.split()[0]}")
print(f"Diretório de trabalho.....: {os.getcwd()}")
print(f"Interpretador Python.....: {sys.executable}")

# ETAPA 4: VERIFICA SE ESTÁ RODANDO EM JUPYTER (para exibir widgets, caso interesse)
#
# -----
try:
    from IPython import get_ipython
    jupyter_info = get_ipython().__class__.__name__

```

```

    print(f"Ambiente de execução.....: Jupyter ({jupyter_info})")
except:
    print(f"Ambiente de execução.....: Python padrão (script)")

# ETAPA 5: FINALIZAÇÃO
#
tempo_total = time.time() - inicio # tempo total em segundos (float)

# Converte para o formato HH:MM:SS
horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

print("\n" + "="*110)
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=*110")

```

=====
=====
PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS
Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais
Configuração Geral do Ambiente
=====
=====

=====

Usuário.....: renat
Data e hora da execução.....: 07/12/2025 15:47:51
Sistema operacional.....: Windows 11 (10.0.26100)
Processador.....: AMD64 Family 25 Model 80 Stepping 0, AuthenticAMD
Python versão.....: 3.13.7
Diretório de trabalho.....: c:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Visão Computacional\VC_TF
Interpretador Python.....: c:\Program Files\Python313\python.exe
Ambiente de execução.....: Jupyter (ZMQInteractiveShell)

=====
=====
Tempo de execução → 00:00:00.00 (hh:mm:ss)
Finalizado em → 07/12/2025 às 15:47:51
=====
=====

In [20]: # ======
BLOCO 1: PLANEJAMENTO
Instalação de Bibliotecas Essenciais (versão 2025 - YOLOv11 + HDBSCA

Objetivo: Instalar/verificar todas as dependências do projeto de forma automática
evitando conflitos de versão.
Cada linha tem comentário explicando para que serve.
======

ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO

print("=* 110")
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")
print(" Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais")
print(" Instalação Automática de Bibliotecas Essenciais (2025)")

```

print("=" * 110)

# ETAPA 2: IMPORTAÇÃO DE MÓDULOS PADRÃO DO PYTHON
# -----
import subprocess # permite executar comandos do sistema
import sys # acessa o caminho do Python atual
import time # usado para pausas visuais e cronogramas
from datetime import datetime # para mostrar data e hora no final

# ETAPA 3: FUNÇÃO QUE INSTALA PACOTES AUTOMATICAMENTE
# Se o pacote já existir → só avisa. Se não existir → instala com pip automaticamente
# -----
import importlib.util # necessário para detecção 100% correta

def instalar_pacote(nome):
    """Verifica com importlib se o pacote está realmente instalado. Só instala se não estiver"""
    nome_modulo = nome.replace('-', '_').split('.')[0]
    if nome_modulo in sys.modules or importlib.util.find_spec(nome_modulo) is not None:
        return True, "já instalada"
    else:
        print(f" → Instalando {nome}...", end=" ")
        try:
            subprocess.check_call([sys.executable, "-m", "pip", "install", nome,
                                  stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT])
            print("OK")
            return True, "instalada agora"
        except:
            print("FALHOU")
            return False, "FALHA"

# ETAPA 4: BARRA DE PROGRESSO (Só funciona no Jupyter/Colab)
# -----
instalar_pacote("ipywidgets")
from IPython.display import display, HTML
import ipywidgets as widgets

# Criação da barra de progresso azul
progress_bar = widgets.FloatProgress(value=0, min=0, max=100, description='Progresso da instalação',
                                      style={'bar_color': '#0066cc'}, layout=widgets.Layout(width="100%"))
status_label = widgets.HTML(value="Iniciando instalação de todas as dependências...")
display(widgets.VBox([status_label, progress_bar]))

# ETAPA 5: LISTA COMPLETA DE TODAS AS BIBLIOTECAS DO PROJETO (2025)
# -----
bibliotecas = [
    # ----- GEOESPACIAL & SATÉLITE -----
    'sentinelhub',
    'rasterio',
    'geopandas',
    'shapely',
    'pyproj',
    'rtree',
    'folium',
    'earthpy',

    # ----- ANÁLISE DE DADOS -----
    'numpy',
    'pandas',
    'scipy',
    'scikit-learn',
]

```

```

# ----- VISÃO COMPUTACIONAL -----
'opencv-python',
'Pillow',

# ----- DEEP LEARNING & YOLOv11 (ESSENCIAIS) -----
'torch',                      # PyTorch - framework principal
'torchvision',                 # transforms e datasets
'ultralytics',                 # PACOTE OFICIAL DO YOLOv11 (2024-2025) - contém yol
'hdbSCAN',                     # ← CLUSTERING NÃO SUPERVISIONADO (o que tu precisav
'supervision',                # (OPCIONAL) anotações bonitas pós-YOLO

# ----- VISUALIZAÇÃO -----
'matplotlib',
'seaborn',
'plotly',

# ----- UTILIDADES GERAIS -----
'tqdm',
'requests',
'boto3',                       # (OPCIONAL) AWS S3
'openpyxl',
'xlsxwriter',
]

# ETAPA 6: LOOP QUE INSTALA TUDO COM BARRA DE PROGRESSO
# -----
total = len(bibliotecas)
concluidas = 0
inicio = time.time()

for pkg in bibliotecas:
    sucesso, msg = instalar_pacote(pkg)
    concluidas += 1
    progresso_percent = (concluidas / total) * 100
    progress_bar.value = progresso_percent
    status_label.value = f"<b>{concluidas}/{total} → {pkg.ljust(25)} {msg}</b>""
    time.sleep(0.07)

# ETAPA 7: IMPORTS DE TODAS AS BIBLIOTECAS (só roda depois que tudo está garantido)
# -----
print("Carregando módulos e validando ambiente...")

# --- 1. Imports: Utilitários de Sistema e Excel ---
from openpyxl import load_workbook                         # Permite carregar e editar
from openpyxl.styles import Font, PatternFill                 # Classes para aplicar negr
from openpyxl.styles import Alignment                        # Classe para centralizar o
from openpyxl.utils import get_column_letter               # Converte números de índic
from pathlib import Path                                    # Manipulação robusta de ca
import os                                                 # Interação com o Sistema O
import json                                              # Manipulação de arquivos J
import math                                              # Funções matemáticas básic
import requests                                         # Biblioteca para fazer req
from glob import glob                                     # Busca de arquivos usando
from datetime import datetime, timedelta                  # Manipulação de datas, hor

# --- 2. Imports: Dados Numéricos e Geoespaciais ---
import numpy as np                                       # Computação numérica de al
import pandas as pd                                      # Manipulação de tabelas de
import geopandas as gpd                                 # Extensão do Pandas para d

```

```

from shapely.geometry import Point, Polygon
import rasterio
from rasterio.plot import show as rio_show
from rasterio.transform import from_bounds
import pyproj
import boto3

# --- 3. Imports: Machine Learning Clássico & Estatística ---
from numpy.linalg import inv
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# --- 4. Imports: Deep Learning & YOLO (O "Core Business") ---
import torch
import torchvision
from ultralytics import YOLO
from ultralytics.utils.metrics import box_iou
from ultralytics.utils.plotting import Annotator
import hdbSCAN
import supervision as sv

# --- 5. Imports: Visualização de Dados e Imagens ---
import cv2
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import folium
from tqdm import tqdm

print("SUCESSO: Todas as bibliotecas foram carregadas e estão prontas para uso!")

# ETAPA 8: FINALIZAÇÃO
# -----
tempo_total = time.time() - inicio
horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

progress_bar.bar_style = 'success'
progress_bar.value = 100
status_label.value = f"<b>INSTALAÇÃO CONCLUÍDA → {total} pacotes em {tempo_forma

print("\n" + "="*110)
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=*110)
=====
```

PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS

Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais
Instalação Automática de Bibliotecas Essenciais (2025)

```
VBox(children=(HTML(value='<b>Iniciando instalação de todas as dependências...</b>'), FloatProgress(value=0.0,...  
    → Instalando scikit-learn... OK  
    → Instalando opencv-python... OK  
    → Instalando Pillow... OK  
Carregando módulos e validando ambiente...  
SUCESSO: Todas as bibliotecas foram carregadas e estão prontas para uso!  
=====  
=====  
Tempo de execução → 00:00:10.08 (hh:mm:ss)  
Finalizado em → 07/12/2025 às 15:48:06  
=====  
=====
```

In [3]:

```
# ======  
# BLOCO 1:      PLANEJAMENTO  
#              Autenticação e Configuração do Sentinel Hub  
#  
# OBJETIVO:    Estabelecer conexão segura e funcional com a API do Sentinel Hub  
#              Permite acesso programático a imagens Sentinel-2  
# ======  
  
# ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO  
# -----  
print("=" * 110)  
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")  
print("          Asseguração Independente de Relatórios ESG via Dados Exógenos Sa")  
print("          Autenticação e Configuração do Sentinel Hub")  
print("=" * 110)  
  
# ETAPA 2: IMPORTAÇÃO DE MÓDULOS DAS CLASSES OFICIAIS DO PACOTE SENTINEL HUB (in  
# -----  
from sentinelhub import (  
    SHConfig,                      # classe para montar a configuração da conta (cl  
    SentinelHubCatalog,            # objeto que faz busca no catálogo STAC (essenci  
    SentinelHubRequest,           # classe principal da Process API - monta e exec  
    SentinelHubDownloadClient,    # cliente para downloads paralelos (várias image  
    DataCollection,               # escolhe a coleção: SENTINEL2_L2A, SENTINEL2_L1  
    MimeType,                     # define formato de saída: TIFF, PNG, JPEG2000 e  
    CRS,                          # sistemas de coordenada → CRS.WGS84, CRS.POP_W  
    BBox,                         # bounding box retangular (o que tu usa na AOI)  
    bbox_to_dimensions,           # função que calcula (width, height) em pixels a  
    Geometry,                     # para usar polígonos irregulares no Lugar de bb  
    parse_time,                   # converte strings "2025-01-01" em datetime que  
    DownloadFailedException       # exceção específica quando o download falhar -  
)  
  
# ETAPA 3: BARRA DE PROGRESSO (Só funciona no Jupyter/Colab)  
# -----  
try:  
    from IPython.display import display # Função para exibir componentes gráficos  
    import ipywidgets as widgets        # Biblioteca para criar barra de progresso  
    use_widgets = True                 # Flag indicando suporte a widgets  
except ImportError:  
    use_widgets = False                # Se não houver suporte, desativa uso de  
  
# Cria barra de progresso estilizada  
if use_widgets:  
    progress_bar = widgets.FloatProgress(
```

```

        value=0,                                     # Inicia a barra em
        min=0, max=100,                             # Valores mínimo e m
        description='Progresso:',                   # Texto exibido ao l
        bar_style='info',                           # Estilo visual (cor
        style={'bar_color': '#1f77b4'},             # Cor azul personaliza
        layout=widgets.Layout(width='80%', height='30px')  # Tamanho da barra
    )
    status_text = widgets.HTML(                  # Campo de texto din
        value="Iniciando configuração do Sentinel Hub..."  # Exibe barra + text
    )
    display(widgets.VBox([status_text, progress_bar]))  # Exibe barra + text

# Definição das etapas do processo com peso percentual (para barra de progresso)
etapas_configuracao = {
    'carregar configuração': 30,                 # Etapa 1: montar objeto de configuraçā
    'registrar área de interesse': 20,           # Etapa 2: definir a área geográfica (A
    'criar evalscript': 50                         # Etapa 3: criar o script principal de
}

# Variáveis de controle do progresso
total_etapas = len(etapas_configuracao)       # Número total de etapas planejadas
etapa_atual = 0                                 # Contador da etapa em execução
progresso_acumulado = 0                        # Acúmulo percentual para a barra
inicio = time.time()                           # Marca o instante inicial (para calcul

# ETAPA 4: CARREGAR CONFIGURAÇÃO COM CREDENCIAIS DO SENTINEL HUB
# -----
etapa_atual += 1                               # Avança co
progresso_acumulado += etapas_configuracao['carregar configuração'] # Soma peso

if use_widgets:
    progress_bar.value = progresso_acumulado      # Atualiza
    status_text.value = (                           # Atualiza
        f"<b>({etapa_atual}/{total_etapas}) Carregando credenciais do Sentinel H
    )

# Preenche credenciais específicas da sua conta (obtidas no painel Sentinel Hub)
config = SHConfig()                           # Cria o própr
config.sh_client_id     = 'ac94a503-6386-43d5-b20b-449a40bfd9dd' # ID do client
config.sh_client_secret = 'HUyyF2J7j0r2YBf3LrTwQthMrc2hKPPv'   # Segredo do c
config.instance_id       = '3cce22e4-9b1e-454c-8f3e-d52f813c07bb' # ID da config

# Verificação simples: garante que nenhum dos campos obrigatórios está vazio
if not (config.sh_client_id and config.sh_client_secret and config.instance_id):
    raise ValueError("X Credenciais do Sentinel Hub ausentes ou incompletas. Ve

time.sleep(1.0)                                # Pequena pausa apenas para feedback vi

# ETAPA 5: REGISTRAR A ÁREA DE INTERESSE (AOI)
# -----
etapa_atual += 1                               # Av
progresso_acumulado += etapas_configuracao['registrar área de interesse'] # At

if use_widgets:
    progress_bar.value = progresso_acumulado      # Atualiz
    status_text.value = (                           # Atualiz
        f"<b>({etapa_atual}/{total_etapas}) Definindo área de interesse (BBox).."
    )

# Define a área de interesse – aqui usamos um pequeno quadrado em torno da UHE P

```

```

lat_centro = -27.45545                      # Latitude aproximada da usina (graus)
lon_centro = -50.64181                      # Longitude aproximada da usina (graus)
raio_km = 2.0                                  # Tamanho do raio (~2 km, ordem de graus)

# Cálculo do Buffer em graus
dlat = raio_km / 110.574
dlon = raio_km / (111.320 * abs(math.cos(math.radians(lat_centro)))))

# Monta as coordenadas da bbox (bounding box) da AOI (Área de Interesse)
bbox_coords = [
    lon_centro - dlon,
    lat_centro - dlat,
    lon_centro + dlon,
    lat_centro + dlat
]

# Cria objeto BBox com CRS WGS84 (Latitude/longitude)
aoi_bbox = BBox(bbox=bbox_coords, crs=CRS.WGS84)

time.sleep(1.0)                                # Pequena pausa para estabilidade visual

# -----
# DEFINIÇÃO DO EVALSCRIPT PRINCIPAL DO PROJETO (7 bandas + dataMask)
# -----
etapa_atual += 1
progresso_acumulado += etapas_configuracao['criar evalscript']

if use_widgets:
    progress_bar.value = progresso_acumulado
    status_text.value = "Criando evalscript principal...""

EVALSCRIPT_7_BANDAS = """
//VERSION=3
function setup() {
    return {
        input: [{ bands: ["B02","B03","B04","B08","B11","B12","dataMask"] }],
        output: { bands: 7, sampleType: "FLOAT32" }
    };
}
function evaluatePixel(s) {
    return [s.B02, s.B03, s.B04, s.B08, s.B11, s.B12, s.dataMask];
}
"""
"""

if use_widgets:
    status_text.value = "" + checked + " Configuração concluída!" + ""
    progress_bar.bar_style = 'success' # Muda a cor para verde

time.sleep(1.0)                                # Pausa miúda entre etapas

# ETAPA 7: FINALIZAÇÃO
# -----
tempo_total = time.time() - inicio             # tempo total em segundos

# Converte para o formato HH:MM:SS
horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

```

```

print("\n" + "*110")
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("*110")
=====
```

```

=====
PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS
    Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais
        Autenticação e Configuração do Sentinel Hub
=====
```

```

VBox(children=(HTML(value='<b>Iniciando configuração do Sentinel Hub...</b>'), Fl
oatProgress(value=0.0, bar_st...
=====
```

```

Tempo de execução → 00:00:03.02 (hh:mm:ss)
Finalizado em → 06/12/2025 às 18:54:16
=====
```

```

In [4]: # =====
# BLOCO 1: PLANEJAMENTO
#           Implementação do Recorte Espacial (Área de interesse)
#
# OBJETIVO: Definir apenas a geometria (AOI) e o período temporal de análise.
#           Variáveis criadas aqui serão usadas em TODAS as células do Bloco 2
# =====

# ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO
#
# -----
print("=* 110")
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")
print("    Asseguração Independente de Relatórios ESG via Dados Exógenos Sa
print("    Implementação do Recorte Espacial (Área de interesse)")
print("    → Buffer circular de 2 km centrado nas coordenadas") # Descri
print("*110")

# ETAPA 2: BARRA DE PROGRESSO (Só funciona no Jupyter/Colab)
#
# -----
try:
    progress = widgets.FloatProgress(                                # Cria widget de barra
        value=0,                                                 # Valor inicial (0% co
        min=0,                                                 # Limite inferior da b
        max=100,                                              # Limite superior da b
        description='Progresso:',                               # Rótulo exibido à esq
        style={'bar_color': '#1f77b4'},                         # Cor azul profissional
        layout=widgets.Layout(width='85%', height='30px') # Tamanho da barra
    )
    status = widgets.HTML(
        value="Iniciando definição do recorte espacial..." # Texto HTML p
    )
    display(widgets.VBox([status, progress])) # Exibe barra + texto empilhados
    use_widgets = True # Flag indicando que widgets est
except ImportError: # Caso rode em ambiente sem suport
    use_widgets = False # Desabilita uso de widgets

inicio = time.time() # Inicia contagem de tempo total

# ETAPA 3: DEFINIÇÃO DA GEOMETRIA DA USINA HIDRELÉTRICA PERY (AOI)
```

```

# -----
if use_widgets:
    progress.value = 50                      # Atualiza barra para 50% na etapa
    status.value = "<b>(1/2) Definindo coordenadas da UHE Pery...</b>" # Mensagem

# Dicionário com metadados da usina – útil para relatórios, pôster e artigo
usina_info = {
    'nome': 'Usina Hidrelétrica Pery',           # Nome oficial da usina
    'operador': 'CELESC Geração S.A.',          # Empresa operadora
    'municipio': 'Curitibanos | Campos Novos',   # Municípios envolvidos
    'estado': 'Santa Catarina',                  # Estado da federação
    'rio': 'Rio Canoas',                         # Rio em que a usina está Localizada
    'raio_buffer_km': raio_km,                   # Raio do buffer circular concebido
    'centro_lat': lat_centro,                    # Latitude do centro da AOI
    'centro_lon': lon_centro,                    # Longitude do centro da AOI
    'bbox_wgs84': aoi_bbox.geometry             # Coordenadas da bounding box em WGS84
}

time.sleep(1.0)                                # Pequena pausa para o usuário pensar

# ETAPA 4: DEFINIÇÃO DO PERÍODO TEMPORAL DE ANÁLISE
# -----
if use_widgets:
    progress.value = 100                     # Atualiza barra para 100% na etapa
    progress.bar_style = 'success'          # Muda cor da barra para verde (success)
    status.value = "<b style='color:green;'>(2/2) Período definido → pronto!</b>"

# Período completo de análise do projeto (5 anos completos + 2025 parcial)
data_inicio = '2021-01-01'                      # Data inicial no formato ISO 8601
data_fim = '2025-10-31'                          # Data final no formato ISO 8601 (Y)

# ETAPA 5: DERIVAÇÃO DOS ANOS E DAS ESTAÇÕES DO ANO DENTRO DO PERÍODO TEMPORAL DEFINIDO
# -----
# Lista de anos efetivamente cobertos pelo período do projeto
ano_ini = int(data_inicio[:4])                  # Extrai ano inicial como inteiro
ano_fim = int(data_fim[:4])                      # Extrai ano final como inteiro
anos_projeto = list(range(ano_ini, ano_fim + 1)) # Cria lista de anos de ano_in a ano_fim

# Função auxiliar: gera estações "teóricas" para um ano, em formato datetime.datetime
def _gerar_estacoes_para_ano(ano: int):
    from datetime import date
    return [
        ("primavera", date(ano, 9, 23), date(ano, 12, 21)),    # Primavera: 23/set a 21/dez
        ("verão",     date(ano, 12, 22), date(ano + 1, 3, 20)), # Verão: 22/dez a 20/mar
        ("outono",    date(ano, 3, 21), date(ano, 6, 20)),      # Outono: 21/mar a 20/jun
        ("inverno",   date(ano, 6, 21), date(ano, 9, 22)),      # Inverno: 21/jun a 20/set
    ]

# Converte limites globais em objetos datetime.date
_limite_ini = datetime.fromisoformat(data_inicio).date() # Converte string → objeto date
_limite_fim = datetime.fromisoformat(data_fim).date()    # Converte string → objeto date

# Dicionário principal: para cada ano, lista de tuplas (estação, data_ini, data_fim)
estacoes_projeto = {}                             # Inicializa dicionário vazio

for ano in anos_projeto:                         # Itera sobre cada ano coberto pelo projeto
    estacoes_ano = []                            # Lista local para armazenar estações
    for nome, ini_est, fim_est in _gerar_estacoes_para_ano(ano): # Gera estações teóricas
        # "Corta" a estação para não sair do período global do projeto
        ini_clip = max(ini_est, _limite_ini) # Data inicial da estação limitada
        fim_clip = min(fim_est, _limite_fim) # Data final da estação limitada
        estacoes_ano.append((nome, ini_clip, fim_clip))
    estacoes_projeto[ano] = estacoes_ano

```

```

        fim_clip = min(fim_est, _limite_fim) # Data final da estação limitada p
        if ini_clip <= fim_clip:           # Garante que a estação ainda faça se
            estacoes_ano.append(
                (nome, ini_clip.isoformat(), fim_clip.isoformat()) # Armazena c
            )
    if estacoes_ano:                  # Se o ano tiver pelo menos uma estação
        estacoes_projeto[ano] = estacoes_ano # Registra no dicionário principal

# Resumo para conferência das estações definidas
print("\nEstações definidas para o período do projeto:")
for ano in sorted(estacoes_projeto.keys()): # Itera anos em ordem crescente
    linhas = [f"{nome}: {ini} → {fim}" for (nome, ini, fim) in estacoes_projeto[ano]]
    print(f" • {ano}: " + " | ".join(linhas)) # Imprime linha resumida por ano

# ETAPA 6: VALIDAÇÃO E RESUMO DAS VARIÁVEIS
# -----
# Cálculo conceitual da área do buffer circular ( $\pi \cdot r^2$ ), em km2
area_km2 = math.pi * (raio_km ** 2) # Área aproximada do buffer circular

print("\n" + " RECORTE ESPACIAL ".center(110, "-"))
print(f" • Usina: {usina_info['nome']} ({usina_info['operador']})" # Nome + operador
print(f" • Localização: {usina_info['municipio']}, {usina_info['estado']}" # Localização
print(f" • Rio: {usina_info['rio']}" # Rio principal
print(f" • Área de interesse (AOI): {area_km2:.2f} km2" # Área calculada
print(f" • Período de análise: {data_inicio[:4]} → {data_fim[:4]}" # Anos iniciados

# ETAPA 7: FINALIZAÇÃO
# -----
tempo_total = time.time() - inicio # tempo total em segundos

# Converte para o formato HH:MM:SS
horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

print("\n" + "=*110")
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=*110")

```

=====

=====

=====

PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS

Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais

Implementação do Recorte Espacial (Área de interesse)

→ Buffer circular de 2 km centrado nas coordenadas

=====

=====

VBox(children=(HTML(value='Iniciando definição do recorte espacial...'), F

loatProgress(value=0.0, descr...

Estações definidas para o período do projeto:

- 2021: primavera: 2021-09-23 → 2021-12-21 | verão: 2021-12-22 → 2022-03-20 | outono: 2021-03-21 → 2021-06-20 | inverno: 2021-06-21 → 2021-09-22
- 2022: primavera: 2022-09-23 → 2022-12-21 | verão: 2022-12-22 → 2023-03-20 | outono: 2022-03-21 → 2022-06-20 | inverno: 2022-06-21 → 2022-09-22
- 2023: primavera: 2023-09-23 → 2023-12-21 | verão: 2023-12-22 → 2024-03-20 | outono: 2023-03-21 → 2023-06-20 | inverno: 2023-06-21 → 2023-09-22
- 2024: primavera: 2024-09-23 → 2024-12-21 | verão: 2024-12-22 → 2025-03-20 | outono: 2024-03-21 → 2024-06-20 | inverno: 2024-06-21 → 2024-09-22
- 2025: primavera: 2025-09-23 → 2025-10-31 | outono: 2025-03-21 → 2025-06-20 | inverno: 2025-06-21 → 2025-09-22

----- RECORTE ESPACIAL -----

- Usina: Usina Hidrelétrica Pery (CELESC Geração S.A.)
- Localização: Curitibanos | Campos Novos, Santa Catarina
- Rio: Rio Canoas
- Área de interesse (AOI): 12.57 km²
- Período de análise: 2021 → 2025

Tempo de execução → 00:00:01.01 (hh:mm:ss)

Finalizado em → 06/12/2025 às 18:54:40

```
In [5]: # -----
# BLOCO 2: EXECUÇÃO - LETRA A
#           Pré-processamento (Testes)
#
# OBJETIVO: Testar as imagens de satélite com garantia de visualização de ima
# Requer BLOCO 1 executado
# -----
#
# ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO
#
# -----
print("=" * 110)
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")
print("          Asseguração Independente de Relatórios ESG via Dados Exógenos Sa")
print("          Pré-processamento das imagens de satélite para testar a visualiz
print("=" *110)

# ETAPA 2: BARRA DE PROGRESSO (Só funciona no Jupyter/Colab)
#
try:
    progress = widgets.FloatProgress(
        value=0,                                     # Cria a barra de
        min=0, max=100,                                # Valor inicial =
        description='Progresso:',                      # Escala de 0 a 1
        style={'bar_color': '#1f77b4', 'description_width': 'initial'}, # Texto fixo à es
        layout=widgets.Layout(width='80%', height='30px')   # Cor da barra
    )
    status = widgets.HTML(value="Iniciando pré-processamento...")
    display(widgets.VBox([status, progress]))          # Exibe texto + b
    use_widgets = True                                # Flag indicando se a barra
except Exception:
    use_widgets = False                             # Caso não esteja disponível, desativa a barra

inicio = time.time()                                # Inicia cronômetro
```

```

# ETAPA 3: PARÂMETROS DE PROCESSAMENTO
# -----
colecao_imagens = DataCollection.SENTINEL2_L2A                                # Coleção Sentinel
anos = sorted(anos_projeto)                                                    # Ordena os anos v
total_anos = len(anos)                                                        # ← total de anos
processados = 0                                                               # ← contador para
raio_km_local = raio_km                                                       # Lê raio do buffer
area_km2 = math.pi * (raio_km_local ** 2)                                       # Área aproximada

# Cria pasta de saída para o grid de imagens, se ainda não existir
os.makedirs("RESULTADOS/GRID_SENTINEL", exist_ok=True)                         # Pasta para salvar

# Evalscript com 7 bandas brutas (FLOAT32) + dataMask separada
# Isso permite baixar os dados brutos e ter máscara de nuvens/dados válidos
evalscript_multispectral_bruto = EVALSCRIPT_7_BANDAS

# ETAPA 4: GRID DE IMAGENS POR ESTAÇÃO DO ANO
# -----
try:
    todas_imagens = []                                                       # Lista agrupa
    ano_ref = anos[0]                                                       # Ano de referê
    estacoes_ref = estacoes_projeto[ano_ref]                                 # Lista de (nom
    estacoes_labels = [nome for (nome, _, _) in estacoes_ref]               # Apenas nomes
    num_estacoes = len(estacoes_labels)                                      # Quantidade de

    for i, ano in enumerate(anos):                                           # Loop principa
        if use_widgets:
            processados += 1
            progress.value = (processados / total_anos) * 100
            status.value = f"<b>Processando ano {ano} ({processados}/{total_anos}</b>"

            imagens_ano = []                                                 # Lista de cami
            datas_ano = []                                                 # Lista de rótu
            urls_validas = []                                              # Lista boolean

            estacoes_ano = estacoes_projeto.get(ano, [])                      # Recupera esta
            limite_estacoes = min(len(estacoes_ano), num_estacoes)          # Garante mesmo

            for idx in range(limite_estacoes):                               # Itera pelas e
                estacao, data_inicio_estacao, data_fim_estacao = estacoes_ano[idx]

                try:
                    # Monta requisição Sentinel Hub para a estação específica
                    req_estacao = SentinelHubRequest(
                        evalscript=evalscript_multispectral_bruto,
                        input_data=[
                            SentinelHubRequest.input_data(
                                colecao_imagens,
                                time_interval=(data_inicio_estacao, data_fim_estacao),
                                other_args={"dataFilter": {"maxCloudCoverage": 30}}
                            )
                        ],
                        responses=[
                            SentinelHubRequest.output_response("default", MimeType.T
                        ],
                        bbox=aoi_bbox,
                        size=(400, 400),
                        config=config
                    )
                
```

```

# Download dos dados brutos (7 canais: 6 bandas + dataMask)
dados = req_estacao.get_data()[0] # (400, 400)

# Separa bandas e máscara
bandas = dados[:, :, :6].astype(np.float32) # 6 bandas
mask_raw = dados[:, :, 6] # dataMask
mask = mask_raw > 0.5 # True

# Rejeita imagem com menos de 30% de pixels válidos
percentual_valido = mask.mean()
if percentual_valido < 0.3:
    raise ValueError(f"Poucos pixels válidos: {percentual_valido}")

# Extrai RGB (B04, B03, B02) e normaliza
rgb = bandas[:, :, [2, 1, 0]] # True
rgb = rgb / 10000.0 # L2A → L1C

# APLICAÇÃO CORRETA DO CONTRASTE
# Calcula percentis para normalização robusta
p_low, p_high = np.percentile(rgb[mask], (2, 98), axis=0)

# Normaliza cada banda individualmente
rgb_norm = np.zeros_like(rgb)
for band in range(3):
    band_data = rgb[:, :, band]
    band_norm = (band_data - p_low[band]) / (p_high[band] - p_low[band])
    rgb_norm[:, :, band] = np.clip(band_norm, 0, 1)

# Aplica gamma para melhorar visualização
rgb_gamma = np.power(rgb_norm, 0.7) # Gamma 0.7 para realçar contrastes

# Converte para uint8
img_rgb = (rgb_gamma * 255).astype(np.uint8)

# Aplica a máscara (pixels inválidos = preto)
img_rgb[~mask] = [0, 0, 0] # RGB preto para nuvens/água/outros

# SALVA PNG VISUAL (OpenCV usa BGR, converte corretamente)
caminho_png = os.path.join("RESULTADOS/GRID_SENTINEL", f"S2_{aoi}_bbox.png")
# Converte RGB para BGR para OpenCV
img_bgr = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR)
cv2.imwrite(caminho_png, img_bgr)

# SALVA GEOTIFF com 6 bandas + máscara original
dados_geotiff = np.concatenate([bandas, mask_raw[..., np.newaxis]])
caminho_tif = os.path.join("RESULTADOS/GRID_SENTINEL", f"S2_{aoi}_bbox.tif")

west, south, east, north = aoi_bbox
transform = from_bounds(west, south, east, north, 400, 400)

with rasterio.open(
    caminho_tif, 'w',
    driver='GTiff',
    height=400, width=400,
    count=7, dtype='float32',
    crs='EPSG:4326',
    transform=transform,
    compress='deflate'
) as dst:

```

```

        for b in range(7):
            dst.write(dados_geotiff[:, :, b], b + 1)

        # Alimenta o grid com o PNG visual
        imagens_ano.append(caminho_png)
        urls_validas.append(True)
        datas_ano.append(f"{data_inicio_estacao} → {data_fim_estacao}")

    except Exception as e:
        print(f" X FAILED {ano} - {estacao} → {e}")
        imagens_ano.append(None)
        urls_validas.append(False)
        datas_ano.append("sem dados")

    # Se o ano tiver menos estações do que o cabeçalho, preenche com espaços
    while len(imagens_ano) < num_estacoes:
        imagens_ano.append(None)
        urls_validas.append(False)
        datas_ano.append("sem imagem")

    todas_imagens.append((ano, imagens_ano, datas_ano, urls_validas))

# Grid HTML
print("\n" + "=" * 110)
print("GRID DE IMAGENS POR ESTAÇÃO DO ANO - GARANTIA DE VISUALIZAÇÃO")
print("=" * 110)

# Cabeçalho do grid (Linha com ANO + nome das estações)
header_html = """
<div style='display: flex; align-items: center; margin-bottom: 20px; background-color: #f0f0f0; padding: 5px; border-radius: 5px'>
    <div style='width: 100px; font-weight: bold; text-align: center; font-size: 1.2em'>ANO</div>
    <div style='flex-grow: 1; border-bottom: 2px solid #007bff; margin: 0 10px'>
        <div style='display: flex; justify-content: space-around; width: 100%; border-bottom: 1px solid #ccc; margin-bottom: 5px'>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 1</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 2</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 3</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 4</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 5</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 6</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 7</div>
        </div>
        <div style='display: flex; justify-content: space-around; width: 100%; border-bottom: 1px solid #ccc; margin-bottom: 5px'>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 1</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 2</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 3</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 4</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 5</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 6</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 7</div>
        </div>
    </div>
</div>
"""

for nome_estacao in estacoes_labels:                      # Itera sobre nome das estações
    header_html += f"<div style='width: 180px; font-weight: bold; text-align: center; font-size: 1.2em'>{nome_estacao}</div>"           # Adiciona nome da estação
header_html += "</div>"                                # Fecha o container de cabeçalho
display(HTML(header_html))                                # Exibe Linha de código

# Linhas do grid, uma por ano
for ano_grid, imagens, datas, validas in todas_imagens: # Itera sobre cada ano
    linha_html = """
<div style='display: flex; align-items: center; margin: 20px 0; padding: 10px; border-bottom: 1px solid #ccc'>
    <div style='width: 100px; font-weight: bold; text-align: center; font-size: 1.2em'>{ano_grid}</div>
    <div style='flex-grow: 1; border-bottom: 2px solid #007bff; margin: 0 10px'>
        <div style='display: flex; justify-content: space-around; width: 100%; border-bottom: 1px solid #ccc; margin-bottom: 5px'>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 1</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 2</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 3</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 4</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 5</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 6</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 7</div>
        </div>
        <div style='display: flex; justify-content: space-around; width: 100%; border-bottom: 1px solid #ccc; margin-bottom: 5px'>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 1</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 2</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 3</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 4</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 5</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 6</div>
            <div style='width: 30px; text-align: center; font-size: 0.8em'>Estação 7</div>
        </div>
    </div>
</div>
"""

    for caminho_img, data, eh_valida in zip(imagens, datas, validas): # Itera sobre cada imagem
        linha_html += f"<div style='width: 180px; text-align: center; margin: 10px 0'>"           # Adiciona largura para a coluna da imagem
        if caminho_img and eh_valida:                                         # Caso haja imagem
            linha_html += f"""
                <div style='border: 3px solid #28a745; border-radius: 8px; padding: 5px; text-align: center; margin-bottom: 10px'>
                    <img src='{caminho_img}' style='width: 150px; height: 150px; object-fit: cover;'>
                    <div style='font-size: 11px; margin-top: 8px; color: #495057'>Disponível</div>
                </div>
            """
        else:                                                               # Caso não haja imagem
            linha_html += f"""
                <div style='width: 150px; height: 150px; border: 3px dashed #6c757d; display: flex; align-items: center; justify-content: center; margin-bottom: 10px'>
                    <img alt='Imagem não disponível' style='width: 100%; height: 100%;'/>
                </div>
            """
        display(HTML(linha_html))                                         # Exibe Linha de código

```

```

background: #f8f9fa; margin: 0 auto; font-size: 12px
Nenhuma imagem<br>disponível
</div>
"""

linha_html += "</div>"                                # Fecha célula indiv

linha_html += "</div>"                                # Fecha Linha do ano
display(HTML(linha_html))                             # Exibe HTML da Linh

# Estatísticas finais do grid
total_imagens = sum(sum(validas) for _, _, _, validas in todas_imagens) # S
print(f"\n• ESTATÍSTICAS: {total_imagens}/{len(anos)} * num_estacoes} imagens")
print(f"• Período (anos efetivos): {anos[0]} a {anos[-1]} ({len(anos)} anos)

except Exception as e:                                # Tratamento de err
    print(f"• ERRO NO GRID: {e}")
    import traceback
    traceback.print_exc()

time.sleep(1)                                         # Pausa simbólica e

# Finalização limpa da barra de progresso
if use_widgets:
    progress.value = 100                               # For
    status.value = "<b>Pré-processamento concluído - todos os anos processados</b>"

# ETAPA 5: FINALIZAÇÃO
#
tempo_total = time.time() - inicio                  # tempo total em seg

# Converte para o formato HH:MM:SS
horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

print("\n" + "="*110)
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=*110")

```

=====
=====

PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS

Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais

Pré-processamento das imagens de satélite para testar a visualização

=====
=====

VBox(children=(HTML(value='Iniciando pré-processamento...'), FloatProgress(value=0.0, description='Prog...')))

=====
=====

GRID DE IMAGENS POR ESTAÇÃO DO ANO - GARANTIA DE VISUALIZAÇÃO

=====
=====

ANO	PRIMAVERA	VERÃO	OUTONO	INVERNO
-----	-----------	-------	--------	---------

2021

2021-09-23 →
2021-12-21
✓ Disponível



2021-12-22 →
2022-03-20
✓ Disponível



2021-03-21 →
2021-06-20
✓ Disponível



2021-06-21 →
2021-09-22
✓ Disponível

2022

2022-09-23 →
2022-12-21
✓ Disponível



2022-12-22 →
2023-03-20
✓ Disponível



2022-03-21 →
2022-06-20
✓ Disponível



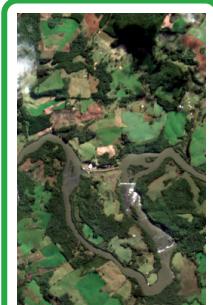
2022-06-21 →
2022-09-22
✓ Disponível

2023

2023-09-23 →
2023-12-21
✓ Disponível



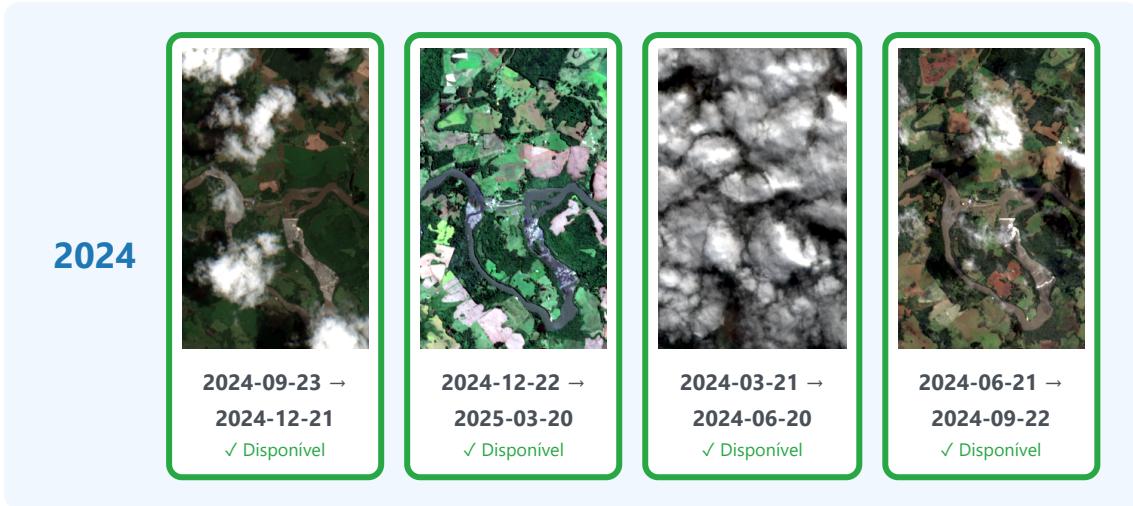
2023-12-22 →
2024-03-20
✓ Disponível



2023-03-21 →
2023-06-20
✓ Disponível



2023-06-21 →
2023-09-22
✓ Disponível



- ESTATÍSTICAS: 19/20 imagens carregadas com sucesso
 - Período (anos efetivos): 2021 a 2025 (5 anos)
-
-

Tempo de execução → 00:01:12.84 (hh:mm:ss)
Finalizado em → 06/12/2025 às 18:56:20

```
In [7]: # =====
# BLOCO 2:      EXECUÇÃO - Letra A
#                  Organização do Dataset (Metadados + Índices Espectrais)
#
# OBJETIVO:    Montar um dataset tabular com as N melhores imagens por estação/
#               contendo: data, estação, nuvem, bandas médias e índices espectrais
#               NBR2, EVI, SAVI), pronto para Mahalanobis e escolha posterior da
#               Requer BLOCO 1 executado
# =====

# ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO
#
print("=" * 110)
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")
print("          Asseguração Independente de Relatórios ESG via Dados Exógenos Sa
```

```

print("          Organização do Dataset tabular com até 20 melhores imagens por e")
print("          Arquivo em Excel com Metadados e Índices Espectrais")
print("=* *110)

# ETAPA 2: BARRA DE PROGRESSO (Só funciona no Jupyter/Colab)
# -----
try:
    progresso = widgets.FloatProgress(                                # cria barra contínua
        value=0, min=0, max=100,
        description='Progresso:',
        style={'bar_color': '#1f77b4'},                                     # azul consistente com b
        layout=widgets.Layout(width='80%', height='30px')
    )
    status = widgets.HTML(
        value=<b>Iniciando organização do dataset Sentinel-2...</b>" # texto ex
    )
    display(widgets.VBox([status, progresso]))                           # empilha texto + barr
    use_widgets = True                                                 # marca que widgets es
except Exception:
    use_widgets = False                                                # em ambiente sem widg

inicio = time.time()                                                 # início da contagem a

# ETAPA 3: RECORTE ESPACIAL (Área de interesse)
# -----
# Já foi criado anteriormente, só vamos renomear pra ficar mais claro aqui
aoi_sentinel = aoi_bbox                                              # mesmo objeto, nome mais semântic
buffer_km     = usina_info['raio_buffer_km']                         # já existia
resolution_m  = 10

MAX_IMAGENS_POR_ESTACAO = 20                                         # máximo de imagens a baixar por e

# Calcula tamanho da imagem em pixels (10m de resolução)
size_x, size_y = bbox_to_dimensions(aoi_sentinel, resolution_m)

print(f"AOI Sentinel-2 pronta → buffer de", buffer_km, f"km | {size_x}x{size_y}")

# ETAPA 4: REUTILIZA AOI E TAMANHO JÁ DEFINIDOS NO BLOCO 1
# -----
bbox         = aoi_sentinel                                         # BBox já criado → reutiliza sem
SIZE_X       = size_x                                              # largura em pixels (10 m de resolução)
SIZE_Y       = size_y                                              # altura em pixels (10 m de resolução)
evalscript   = EVALSCRIPT_7_BANDAS                                 # variável global

# ETAPA 5: BUSCAR IMAGENS NO CATÁLOGO
# -----
if use_widgets:
    progresso.value = 10                                           # Atualiza progresso
    status.value = "<b>Buscando imagens no catálogo Sentinel-2...</b>" # Atualiza status

print("Buscando imagens no catálogo Sentinel-2...")                  # Log no terminal

# Cria cliente do catálogo Sentinel Hub usando a configuração já definida
catalog = SentinelHubCatalog(config=config)                          # config

# CORREÇÃO: As variáveis data_inicio e data_fim vêm do Bloco 1 (Recorte Espacial)
# Elas estão no formato 'YYYY-MM-DD', mas o Sentinel Hub precisa do formato ISO
# Solução: usar a função parse_time do sentinelhub para converter corretamente
# Importa a função parse_time (se já não estiver importada)
# from sentinelhub import parse_time # ← Se precisar, adicione no início do bloco

```

```

# Converte as strings para objetos datetime no formato correto
data_inicio_dt = parse_time(data_inicio)                                     # Converte
data_fim_dt = parse_time(data_fim)                                         # Converte

print(f"    Período de busca: {data_inicio} até {data_fim}")                 # Mostra

# Busca cenas Sentinel-2 L2A no catálogo com os parâmetros definidos
search_iterator = catalog.search(
    collection=DataCollection.SENTINEL2_L2A,                                    # Coleção: Se
    bbox=aoi_sentinel,                                                       # Área de int
    time=(data_inicio_dt, data_fim_dt),                                       # Período tem
    filter="eo:cloud_cover<20",                                              # Filtro: máx
    fields="id,properties,bbox,geometry,assets"                                # Campos a re
    # Se não especificar fields, retorna todos os campos
)

# Converte o iterador em lista para processamento
todas_cenas = list(search_iterator)
print(f"Encontradas {len(todas_cenas)} cenas com menos de 20% de nuvens")

# ETAPA 6: PROCESSAR TODAS AS CENAS ENCONTRADAS NO CATÁLOGO SEM APLICAR O LIMITE
# -----
if use_widgets:
    progresso.value = 20                                                 # Avança
    status.value = "<b>Processando cenas e calculando índices...</b>" # Informa

# Cria uma lista vazia para armazenar TODAS as cenas processadas, sem aplicar o
todas_processadas = []                                                 # Esta lista

# Inicia o loop para processar cada cena encontrada no catálogo
for i, cena in enumerate(todas_cenas):                                     # 'enumerate'
    # Atualiza a barra de progresso proporcionalmente ao número de cenas processadas
    if use_widgets:
        # Fórmula: começa em 20% e vai até 90% conforme processa as cenas
        # (i / len(todas_cenas)) * 70 → percentual do processamento atual
        progresso.value = 20 + (i / len(todas_cenas)) * 70                  # 20% a 90%

try:
    # Extrai as propriedades da cena do Sentinel Hub
    props = cena["properties"]                                              # Contém metadados

    # Data da imagem - converte de string ISO para objeto datetime do Python
    data_img = datetime.fromisoformat(props["datetime"].replace("Z", ""))
    ano = data_img.year                                                       # Extrai o ano
    mes = data_img.month                                                      # Extrai o mês

    # Determinar estação do ano com base nas definições do projeto
    estacao = None                                            # Inicializa
    ordem_estacao = None                                         # Ordem numérica
    # Percorre a lista de estações definidas para este ano específico
    for est_nome, ini_str, fim_str in estacoes_projeto.get(ano, []):
        ini = datetime.fromisoformat(ini_str).date()                # Data de início
        fim = datetime.fromisoformat(fim_str).date()                # Data de término
        # Verifica se a data da imagem está dentro do intervalo desta estação
        if ini <= data_img.date() <= fim:
            estacao = est_nome                                      # Define o nome
            # Define ordem numérica para facilitar ordenação posterior
            if est_nome == "primavera": ordem_estacao = 1
            elif est_nome == "verão": ordem_estacao = 2

```

```

        elif est_nome == "outono": ordem_estacao = 3
        elif est_nome == "inverno": ordem_estacao = 4
        break                                         # Sai do Loop

    # Se não encontrou estação (data fora dos intervalos definidos), pula essa iteração
    if not estacao:
        continue                                     # Vai para a próxima iteração

    # Extrai a cobertura de nuvens da cena (valor entre 0 e 100%)
    cloud_cover = props.get("eo:cloud_cover", 0)      # Usa 0 como valor padrão se não houver

    # Baixar dados espectrais da imagem via Sentinel Hub Request
    # Cria uma requisição para obter os valores das 6 bandas + máscara
    request = SentinelHubRequest(
        evalscript=evalscript,                         # Script que calcula as bandas
        input_data=[
            SentinelHubRequest.input_data(
                DataCollection.SENTINEL2_L2A,           # Usa Sentinel 2 L2A
                time_interval=(data_img - timedelta(hours=1), data_img + timedelta(hours=1))
            )
        ],
        responses=[SentinelHubRequest.output_response("default",MimeType.TIFF)],          # Área de interesse
        bbox=aoi_sentinel,                         # Tamanho da máscara
        size=(SIZE_X, SIZE_Y),                     # Configurações
        config=config
    )

    # Executa a requisição e obtém os dados
    dados = request.get_data()[0]                  # Array 3D: [banda][y][x]

    # Calcular médias das bandas (ignorando pixels inválidos usando a máscara)
    mask_valido = dados[:, :, 6] > 0.5             # Band 7 é a máscara
    bandas_medias = []
    for b in range(6):
        banda = dados[:, :, b]
        media = np.nanmean(banda[mask_valido]) if np.any(mask_valido) else np.nan
        bandas_medias.append(media)                 # Adiciona a média à lista

    # Desempacota as médias das bandas em variáveis individuais com nomes sensatos
    B02, B03, B04, B08, B11, B12 = bandas_medias      # B02=Azul, B03=Verde, B04=Verde-Azul, B08=Amarelo, B11=Amarelo-Verde, B12=Amarelo-Verde-Azul

    # Função auxiliar para evitar divisão por zero nos cálculos dos índices
    def safe_div(a, b):
        return a / b if abs(b) > 1e-10 else np.nan     # Retorna NaN se o denominador for zero

    # Calcula os índices espectrais usando as médias das bandas
    ndvi = safe_div(B08 - B04, B08 + B04)            # NDVI: Índice de Diferença de Vermelho e Azul
    ndwi = safe_div(B03 - B08, B03 + B08)            # NDWI: Índice de Diferença de Vermelho e Verde
    nbr2 = safe_div(B12 - B11, B12 + B11)            # NBR2: Índice de Número de Bandas Reduzido
    evi = 2.5 * safe_div(B08 - B04, B08 + 6*B04 - 7.5*B02 + 1) # EVI: Índice de Exposição à Luz do Sol
    savi = 1.5 * safe_div(B08 - B04, B08 + B04 + 0.5)   # SAVI: Índice de Saturação da Vegetação

    # Cria dicionário com todos os metadados e valores calculados
    registro = {
        "DATA_IMG": data_img.strftime("%Y-%m-%d"),       # Data no formato YYYY-MM-DD
        "ANO": ano,                                       # Ano numérico
        "MES": mes,                                       # Mês numérico
        "ESTACAO": estacao,                                # Nome da estação
    }

```

```

        "ORDEM_ESTACAO": ordem_estacao,                                # Ordem numé
        "CLOUD_PCT": cloud_cover,                                     # Porcentage
        "B02_MEAN": B02,                                            # Média banda
        "B03_MEAN": B03,                                            # Média banda
        "B04_MEAN": B04,                                            # Média banda
        "B08_MEAN": B08,                                            # Média banda
        "B11_MEAN": B11,                                            # Média banda
        "B12_MEAN": B12,                                            # Média banda
        "NDVI": ndvi,                                              # Índice NDV
        "NDWI": ndwi,                                              # Índice NDW
        "NBR2": nbr2,                                               # Índice NBR
        "EVI": evi,                                                 # Índice EVI
        "SAVI": savi,                                               # Índice SAV
        "BUFFER_KM": buffer_km,                                     # Tamanho do
        "RES_M": resolution_m,                                      # Resolução
        "SCENE_ID": cena["id"]                                       # ID único a

    }

    todas_processadas.append(registro)                             # Adiciona
    # Log no console para acompanhamento
    print(f" ✓ {data_img.strftime('%Y-%m-%d')} | {estacao} | Nuvem: {cloud_}

except Exception as e:
    # Se ocorrer qualquer erro no processamento desta cena, registra e continua
    print(f" X Erro na cena: {str(e)[:100]}...")                 # Mostra a
    continue                                                       # Vai para

# ETAPA 7: FILTRAR AS MELHORES 20 IMAGENS POR ESTAÇÃO/ANO (menos nuvem)
# -----
if use_widgets:
    progresso.value = 95                                         #
    status.value = "<b>Selecionando melhores imagens por estação/ano...</b>" #

# Converte a lista de dicionários para DataFrame do pandas para facilitar manipulação
df_temp = pd.DataFrame(todas_processadas)

# Log informativo: mostra quantas cenas foram processadas antes do filtro
print(f"\nTotal de cenas processadas (antes do filtro): {len(df_temp)}")

# Lista para armazenar os DataFrames filtrados de cada grupo
registros_filtrados = []

# Agrupa as imagens por combinação de ANO e ESTACAO
for (ano, estacao), grupo in df_temp.groupby(["ANO", "ESTACAO"]):
    # Ordena cada grupo por cobertura de nuvem (menor para maior)
    grupo_sorted = grupo.sort_values(by="CLOUD_PCT")                # Menos nuvem

    # Seleciona apenas as primeiras 20 imagens (com menos nuvens) deste grupo
    selecionadas = grupo_sorted.head(MAX_IMAGENS_POR_ESTACAO)       # MAX_IMAGEN

    registros_filtrados.append(selecionadas)                          # Adiciona à

# Junta todos os grupos selecionados em um único DataFrame
if registros_filtrados:                                           # Se há registro
    df = pd.concat(registros_filtrados, ignore_index=True)         # Concatena
else:
    df = pd.DataFrame()                                             # Cria DataFrame

# ETAPA 8: MONTAGEM DO DATAFRAME E EXPORTAÇÃO PARA O EXCEL
# -----

```

```

# Verifica se o DataFrame está vazio (nenhuma imagem válida)
if len(df) == 0:
    raise RuntimeError("Nenhuma imagem válida foi processada. Verifique filtros")

# Ordena o dataset final por: 1) Ano, 2) Ordem da estação, 3) Data da imagem
df.sort_values(by=["ANO", "ORDEM_ESTACAO", "DATA_IMG"], inplace=True)

# Completa a barra de progresso para 100%
if use_widgets:
    progresso.value = 100
    status.value = "<b>Processamento concluído!</b>" # Barra comp
# Mensagem f

# Caminho onde o arquivo Excel será salvo
caminho_excel = "RESULTADOS/DATASET_S2_USINA_PERY.xlsx"

# Salva o DataFrame em Excel, criando uma planilha chamada "Dataset"
df.to_excel(caminho_excel, index=False, sheet_name="Dataset") # index=Fals

print(f"\n Dataset salvo em: {caminho_excel}") # Log do cam
print(f"    Total de imagens processadas: {len(df)}") # Log do tot

# ETAPA 9: CRIAR UMA PLANILHA CHAMADA SUMÁRIO COM AS SIGLAS E DESCRIÇÕES
# -----
# Dicionário com as principais siglas usadas no dataset, descrição e métrica res
linhas_sumario = [
    {"SIGLA": "B02_MEAN", "DESCRICA0": "Reflectância média banda Azul (Sentinel-2)", "METRICA": "Média dos pixels válidos na AOI (valor unitário 0-1)"}, {"SIGLA": "B03_MEAN", "DESCRICA0": "Reflectância média banda Verde (Sentinel-2)", "METRICA": "Média dos pixels válidos na AOI (valor unitário 0-1)"}, {"SIGLA": "B04_MEAN", "DESCRICA0": "Reflectância média banda Vermelha (Sentinel-2)", "METRICA": "Média dos pixels válidos na AOI (valor unitário 0-1)"}, {"SIGLA": "B08_MEAN", "DESCRICA0": "Reflectância média banda NIR (Sentinel-2)", "METRICA": "Média dos pixels válidos na AOI (valor unitário 0-1)"}, {"SIGLA": "B11_MEAN", "DESCRICA0": "Reflectância média banda SWIR1 (Sentinel-2)", "METRICA": "Média dos pixels válidos na AOI (valor unitário 0-1)"}, {"SIGLA": "B12_MEAN", "DESCRICA0": "Reflectância média banda SWIR2 (Sentinel-2)", "METRICA": "Média dos pixels válidos na AOI (valor unitário 0-1)"}, {"SIGLA": "NDVI", "DESCRICA0": "Índice de Vegetação por Diferença Normalizada", "METRICA": "(NIR - RED) / (NIR + RED)"}, {"SIGLA": "NDWI", "DESCRICA0": "Índice Normalizado de Água (McFeeters)", "METRICA": "(GREEN - NIR) / (GREEN + NIR)"}, {"SIGLA": "NBR2", "DESCRICA0": "Índice Normalizado de Burn Ratio 2", "METRICA": "(SWIR2 - SWIR1) / (SWIR2 + SWIR1)"}, {"SIGLA": "EVI", "DESCRICA0": "Enhanced Vegetation Index", "METRICA": "2.5 × (NIR - RED) / (NIR + 6×RED - 7.5×BLUE + 1)"}, {"SIGLA": "SAVI", "DESCRICA0": "Soil Adjusted Vegetation Index (L=0,5)", "METRICA": "[((NIR - RED) / (NIR + RED + L)) × (1 + L)]"}, {"SIGLA": "CLOUD_PCT", "DESCRICA0": "Cobertura de nuvens na cena Sentinel-2", "METRICA": "Porcentagem de nuvem reportada pelo provedor (%)"}, {"SIGLA": "BUFFER_KM", "DESCRICA0": "Raio do buffer circular da AOI", "METRICA": "Quilômetros a partir da barragem da UHE Pery"}, {"SIGLA": "RES_M", "DESCRICA0": "Resolução espacial nominal", "METRICA": "Tamanho de pixel em metros (10 m)"}, ]
]

wb = load_workbook(caminho_excel) # Abre o workbook recém-criado
ws_dataset = wb["Dataset"] # Obtém a aba principal de dados
ws_sumario = wb.create_sheet("Sumário") # Cria a aba Sumário

```

```

# Escreve cabeçalhos da aba Sumario
ws_sumario["A1"] = "SIGLA"
ws_sumario["B1"] = "DESCRICAO"
ws_sumario["C1"] = "METRICA"

# Preenche as linhas com as informações de cada sigla
for i, linha in enumerate(linhas_sumario, start=2):
    ws_sumario[f"A{i}"] = linha["SIGLA"]
    ws_sumario[f"B{i}"] = linha["DESCRICAO"]
    ws_sumario[f"C{i}"] = linha["METRICA"]

# ETAPA 10: FORMATAÇÃO DAS PLANILHAS DO ARQUIVO
# -----
# Define estilos comuns
fonte_cabecalho = Font(name="Calibri", size=12, bold=True) # ca
fonte_corpo = Font(name="Calibri", size=12, bold=False) # cé
fundo_cabecalho = PatternFill("solid", fgColor="DDDDDD") # ci
alinhamento_esq = Alignment(horizontal="left", vertical="center", indent=1) # al

def aplica_estilo_planilha(ws):
    """Aplica estilo padrão a uma planilha (cabecalho e corpo)."""
    max_col = ws.max_column # número total
    max_row = ws.max_row # número total

    # Estiliza cabeçalhos (Linha 1)
    for col in range(1, max_col + 1):
        cell = ws.cell(row=1, column=col)
        cell.font = fonte_cabecalho
        cell.fill = fundo_cabecalho
        cell.alignment = alinhamento_esq
        col_letra = get_column_letter(col)
        ws.column_dimensions[col_letra].width = 15 # Largura fixa

    # Estiliza o restante das células (corpo da tabela)
    for row in range(2, max_row + 1):
        for col in range(1, max_col + 1):
            cell = ws.cell(row=row, column=col)
            cell.font = fonte_corpo
            cell.alignment = alinhamento_esq

    # Aplica o estilo à aba Dataset e à aba Sumario
    aplica_estilo_planilha(ws_dataset)
    aplica_estilo_planilha(ws_sumario)

    # Salva o workbook com todas as modificações
    wb.save(caminho_excel)

# ETAPA 11: FINALIZAÇÃO
# -----
tempo_total = time.time() - inicio # tempo total em seg

# Converte para o formato HH:MM:SS
horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

print("\n" + "="*110)
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")

```

```
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=*110")
```

PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS

Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais
Organização do Dataset tabular com até 20 melhores imagens por estação/a no

Arquivo em Excel com Metadados e Índices Espectrais

```
VBox(children=(HTML(value='<b>Iniciando organização do dataset Sentinel-2...</b>'), FloatProgress(value=0.0, d...
```

AOI Sentinel-2 pronta → buffer de 2.0 km | 401×400 pixels (10m)

Buscando imagens no catálogo Sentinel-2...

Período de busca: 2021-01-01 até 2025-10-31

Encontradas 205 cenas com menos de 20% de nuvens

✓ 2025-10-15	primavera	Nuvem: 0.0%	NDVI: 0.665
✓ 2025-10-15	primavera	Nuvem: 0.1%	NDVI: 0.665
✓ 2025-09-23	primavera	Nuvem: 4.6%	NDVI: 0.567
✓ 2025-09-13	inverno	Nuvem: 0.0%	NDVI: 0.642
✓ 2025-09-10	inverno	Nuvem: 0.1%	NDVI: 0.647
✓ 2025-08-21	inverno	Nuvem: 0.0%	NDVI: 0.648
✓ 2025-08-14	inverno	Nuvem: 5.8%	NDVI: 0.343
✓ 2025-08-14	inverno	Nuvem: 7.5%	NDVI: 0.343
✓ 2025-08-13	inverno	Nuvem: 3.0%	NDVI: 0.622
✓ 2025-07-30	inverno	Nuvem: 0.0%	NDVI: 0.674
✓ 2025-07-22	inverno	Nuvem: 3.5%	NDVI: 0.529
✓ 2025-07-20	inverno	Nuvem: 8.9%	NDVI: 0.091
✓ 2025-07-15	inverno	Nuvem: 12.5%	NDVI: 0.119
✓ 2025-07-07	inverno	Nuvem: 2.2%	NDVI: 0.449
✓ 2025-07-02	inverno	Nuvem: 10.8%	NDVI: 0.044
✓ 2025-06-12	outono	Nuvem: 0.3%	NDVI: 0.781
✓ 2025-05-18	outono	Nuvem: 0.0%	NDVI: 0.762
✓ 2025-05-18	outono	Nuvem: 8.7%	NDVI: 0.762
✓ 2025-05-11	outono	Nuvem: 0.7%	NDVI: 0.777
✓ 2025-05-06	outono	Nuvem: 1.0%	NDVI: 0.266
✓ 2025-05-01	outono	Nuvem: 0.8%	NDVI: 0.719
✓ 2025-04-28	outono	Nuvem: 2.9%	NDVI: 0.706
✓ 2025-04-28	outono	Nuvem: 2.9%	NDVI: 0.706
✓ 2025-04-06	outono	Nuvem: 2.7%	NDVI: 0.676
✓ 2025-03-22	outono	Nuvem: 5.4%	NDVI: 0.690
✓ 2024-12-27	verão	Nuvem: 0.9%	NDVI: 0.803
✓ 2024-12-17	primavera	Nuvem: 2.5%	NDVI: 0.600
✓ 2024-11-17	primavera	Nuvem: 0.0%	NDVI: 0.651
✓ 2024-10-30	primavera	Nuvem: 18.6%	NDVI: 0.513
✓ 2024-10-28	primavera	Nuvem: 5.1%	NDVI: 0.449
✓ 2024-10-25	primavera	Nuvem: 0.0%	NDVI: 0.687
✓ 2024-10-13	primavera	Nuvem: 0.1%	NDVI: 0.669
✓ 2024-09-30	primavera	Nuvem: 19.4%	NDVI: 0.662
✓ 2024-09-28	primavera	Nuvem: 12.4%	NDVI: 0.361
✓ 2024-09-18	inverno	Nuvem: 15.0%	NDVI: 0.572
✓ 2024-09-10	inverno	Nuvem: 4.8%	NDVI: 0.598
✓ 2024-09-08	inverno	Nuvem: 5.4%	NDVI: 0.542
✓ 2024-09-03	inverno	Nuvem: 7.0%	NDVI: 0.579
✓ 2024-08-26	inverno	Nuvem: 1.8%	NDVI: 0.642
✓ 2024-08-19	inverno	Nuvem: 0.0%	NDVI: 0.644
✓ 2024-08-16	inverno	Nuvem: 6.0%	NDVI: 0.443
✓ 2024-08-14	inverno	Nuvem: 0.0%	NDVI: 0.676
✓ 2024-08-11	inverno	Nuvem: 7.2%	NDVI: 0.186
✓ 2024-08-09	inverno	Nuvem: 2.8%	NDVI: 0.689
✓ 2024-08-06	inverno	Nuvem: 10.9%	NDVI: 0.323
✓ 2024-07-25	inverno	Nuvem: 0.0%	NDVI: 0.634
✓ 2024-07-22	inverno	Nuvem: 2.3%	NDVI: 0.581
✓ 2024-07-20	inverno	Nuvem: 19.4%	NDVI: 0.064
✓ 2024-07-02	inverno	Nuvem: 0.2%	NDVI: 0.693
✓ 2024-06-30	inverno	Nuvem: 0.0%	NDVI: 0.714
✓ 2024-06-27	inverno	Nuvem: 0.0%	NDVI: 0.724
✓ 2024-06-22	inverno	Nuvem: 2.6%	NDVI: 0.453
✓ 2024-06-12	outono	Nuvem: 6.6%	NDVI: 0.116
✓ 2024-06-07	outono	Nuvem: 2.7%	NDVI: 0.711
✓ 2024-06-02	outono	Nuvem: 9.7%	NDVI: 0.385
✓ 2024-05-06	outono	Nuvem: 0.0%	NDVI: 0.684

✓	2024-04-21	outono	Nuvem: 0.0%	NDVI: 0.665
✓	2024-04-18	outono	Nuvem: 0.0%	NDVI: 0.661
✓	2024-04-06	outono	Nuvem: 9.9%	NDVI: 0.146
✓	2024-03-27	outono	Nuvem: 1.0%	NDVI: 0.653
✓	2024-03-27	outono	Nuvem: 4.8%	NDVI: 0.653
✓	2023-12-28	verão	Nuvem: 2.7%	NDVI: 0.592
✓	2023-11-05	primavera	Nuvem: 3.1%	NDVI: 0.628
✓	2023-10-26	primavera	Nuvem: 19.6%	NDVI: 0.621
✓	2023-09-24	primavera	Nuvem: 2.0%	NDVI: 0.701
✓	2023-09-16	inverno	Nuvem: 6.3%	NDVI: 0.693
✓	2023-08-30	inverno	Nuvem: 0.3%	NDVI: 0.688
✓	2023-08-20	inverno	Nuvem: 2.0%	NDVI: 0.684
✓	2023-08-17	inverno	Nuvem: 3.8%	NDVI: 0.576
✓	2023-08-15	inverno	Nuvem: 9.3%	NDVI: 0.100
✓	2023-08-07	inverno	Nuvem: 5.8%	NDVI: 0.659
✓	2023-08-05	inverno	Nuvem: 0.0%	NDVI: 0.708
✓	2023-08-02	inverno	Nuvem: 3.7%	NDVI: 0.628
✓	2023-07-31	inverno	Nuvem: 8.3%	NDVI: 0.319
✓	2023-07-16	inverno	Nuvem: 9.2%	NDVI: 0.194
✓	2023-07-06	inverno	Nuvem: 2.4%	NDVI: 0.560
✓	2023-06-28	inverno	Nuvem: 2.7%	NDVI: 0.590
✓	2023-06-26	inverno	Nuvem: 1.1%	NDVI: 0.642
✓	2023-06-18	outono	Nuvem: 2.4%	NDVI: 0.729
✓	2023-06-08	outono	Nuvem: 13.7%	NDVI: 0.735
✓	2023-06-03	outono	Nuvem: 15.3%	NDVI: 0.184
✓	2023-06-01	outono	Nuvem: 4.0%	NDVI: 0.144
✓	2023-05-24	outono	Nuvem: 5.7%	NDVI: 0.725
✓	2023-05-02	outono	Nuvem: 0.5%	NDVI: 0.683
✓	2023-04-19	outono	Nuvem: 19.4%	NDVI: 0.247
✓	2023-04-04	outono	Nuvem: 8.2%	NDVI: 0.662
✓	2023-04-02	outono	Nuvem: 0.0%	NDVI: 0.697
✓	2022-12-30	verão	Nuvem: 7.6%	NDVI: 0.597
✓	2022-12-23	verão	Nuvem: 0.0%	NDVI: 0.727
✓	2022-12-15	primavera	Nuvem: 7.2%	NDVI: 0.685
✓	2022-12-13	primavera	Nuvem: 0.4%	NDVI: 0.696
✓	2022-12-10	primavera	Nuvem: 2.8%	NDVI: 0.665
✓	2022-12-08	primavera	Nuvem: 0.0%	NDVI: 0.636
✓	2022-11-25	primavera	Nuvem: 1.4%	NDVI: 0.609
✓	2022-11-20	primavera	Nuvem: 16.4%	NDVI: 0.595
✓	2022-11-18	primavera	Nuvem: 0.0%	NDVI: 0.608
✓	2022-11-15	primavera	Nuvem: 14.1%	NDVI: 0.564
✓	2022-11-10	primavera	Nuvem: 4.0%	NDVI: 0.605
✓	2022-11-03	primavera	Nuvem: 3.2%	NDVI: 0.646
✓	2022-10-24	primavera	Nuvem: 15.7%	NDVI: 0.474
✓	2022-09-24	primavera	Nuvem: 0.0%	NDVI: 0.662
✓	2022-09-04	inverno	Nuvem: 4.1%	NDVI: 0.681
✓	2022-08-27	inverno	Nuvem: 5.8%	NDVI: 0.659
✓	2022-08-25	inverno	Nuvem: 0.0%	NDVI: 0.674
✓	2022-08-20	inverno	Nuvem: 1.2%	NDVI: 0.685
✓	2022-08-02	inverno	Nuvem: 6.7%	NDVI: 0.678
✓	2022-07-26	inverno	Nuvem: 4.0%	NDVI: 0.086
✓	2022-07-23	inverno	Nuvem: 3.0%	NDVI: 0.687
✓	2022-07-01	inverno	Nuvem: 19.3%	NDVI: 0.115
✓	2022-06-18	outono	Nuvem: 5.1%	NDVI: 0.622
✓	2022-06-13	outono	Nuvem: 7.9%	NDVI: 0.172
✓	2022-06-11	outono	Nuvem: 0.5%	NDVI: 0.722
✓	2022-05-24	outono	Nuvem: 1.3%	NDVI: 0.700
✓	2022-05-22	outono	Nuvem: 3.0%	NDVI: 0.177
✓	2022-05-07	outono	Nuvem: 13.8%	NDVI: 0.262
✓	2022-04-27	outono	Nuvem: 0.3%	NDVI: 0.697

```

✓ 2022-04-24 | outono | Nuvem: 13.7% | NDVI: 0.704
✓ 2022-04-19 | outono | Nuvem: 16.6% | NDVI: 0.259
✓ 2022-04-17 | outono | Nuvem: 9.5% | NDVI: 0.562
✓ 2022-03-28 | outono | Nuvem: 9.0% | NDVI: 0.357
✓ 2021-12-28 | verão | Nuvem: 0.1% | NDVI: 0.719
✓ 2021-12-25 | verão | Nuvem: 0.2% | NDVI: 0.723
✓ 2021-12-23 | verão | Nuvem: 0.0% | NDVI: 0.753
✓ 2021-12-20 | primavera | Nuvem: 3.1% | NDVI: 0.710
✓ 2021-12-10 | primavera | Nuvem: 1.8% | NDVI: 0.688
✓ 2021-12-08 | primavera | Nuvem: 6.5% | NDVI: 0.634
✓ 2021-11-13 | primavera | Nuvem: 10.3% | NDVI: 0.583
✓ 2021-11-03 | primavera | Nuvem: 2.7% | NDVI: 0.570
✓ 2021-10-26 | primavera | Nuvem: 3.0% | NDVI: 0.649
✓ 2021-09-09 | inverno | Nuvem: 0.0% | NDVI: 0.683
✓ 2021-08-22 | inverno | Nuvem: 0.0% | NDVI: 0.652
✓ 2021-08-20 | inverno | Nuvem: 19.9% | NDVI: 0.541
✓ 2021-08-17 | inverno | Nuvem: 14.8% | NDVI: 0.619
✓ 2021-07-31 | inverno | Nuvem: 3.9% | NDVI: 0.671
✓ 2021-07-28 | inverno | Nuvem: 5.8% | NDVI: 0.480
✓ 2021-07-23 | inverno | Nuvem: 5.2% | NDVI: 0.683
✓ 2021-07-21 | inverno | Nuvem: 5.0% | NDVI: 0.694
✓ 2021-07-18 | inverno | Nuvem: 0.7% | NDVI: 0.713
✓ 2021-07-11 | inverno | Nuvem: 0.0% | NDVI: 0.210
✓ 2021-06-13 | outono | Nuvem: 0.0% | NDVI: 0.731
✓ 2021-06-01 | outono | Nuvem: 15.5% | NDVI: 0.050
✓ 2021-05-27 | outono | Nuvem: 0.0% | NDVI: 0.708
✓ 2021-05-22 | outono | Nuvem: 7.4% | NDVI: 0.706
✓ 2021-05-07 | outono | Nuvem: 11.9% | NDVI: 0.639
✓ 2021-04-29 | outono | Nuvem: 0.3% | NDVI: 0.661
✓ 2021-04-27 | outono | Nuvem: 0.1% | NDVI: 0.672
✓ 2021-04-22 | outono | Nuvem: 3.0% | NDVI: 0.368
✓ 2021-04-19 | outono | Nuvem: 7.8% | NDVI: 0.661
✓ 2021-04-09 | outono | Nuvem: 9.2% | NDVI: 0.656
✓ 2021-04-02 | outono | Nuvem: 19.5% | NDVI: 0.440

```

Total de cenas processadas (antes do filtro): 150

Dataset salvo em: RESULTADOS/DATASET_S2_USINA_PERY.xlsx
 Total de imagens processadas: 150

```
=====
=====
Tempo de execução → 00:08:59.75 (hh:mm:ss)
Finalizado em → 06/12/2025 às 19:15:28
=====
```

In [8]:

```

# =====
# BLOCO 2: EXECUÇÃO - LETRA A
#           Avaliação da Qualidade do Dataset
#           Classifica os dados em A, B, C, D por scores ponderados
#           (70% Nuvens + 30% Neblina)
#
# OBJETIVO: Geração de Scores e Classes para decisão de Download
# =====

# ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO
#
print("=" * 110)
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")

```

```

print("          Processamento Estatístico: Classificação de Qualidade (Mahalanobis")
print("          Avaliação da Qualidade do Dataset para tomada de decisão de Down")
print("=" * 110)

# importação perdida que ficou faltando
from scipy.spatial.distance import mahalanobis
from scipy.stats import chi2

# ETAPA 2: BARRA DE PROGRESSO (Jupyter/Colab)
# -----
try:
    progresso = widgets.FloatProgress(
        value=0, min=0, max=100,
        description='Progresso:',
        style={'bar_color': '#1f77b4'},
        layout=widgets.Layout(width='80%', height='30px')
    )
    status = widgets.HTML(value="Lendo planilha e preparando downloads A/B/C.")
    display(widgets.VBox([status, progresso]))
    use_widgets = True
except Exception:
    use_widgets = False
    print("Widgets não disponíveis - executando em modo texto")

inicio_geral = time.time()

# ETAPA 3: CARREGAR O DATASET (Gerado no passo anterior)
# -----
caminho_arquivo = "RESULTADOS/DATASET_S2_USINA_PERY.xlsx"
if not os.path.exists(caminho_arquivo):
    # Fallback se estiver na raiz
    caminho_arquivo = "DATASET_S2_USINA_PERY.xlsx"

if not os.path.exists(caminho_arquivo):
    raise FileNotFoundError("Erro: O arquivo Excel do dataset não foi encontrado")

print(f'Lendo dados de: {caminho_arquivo}')
df = pd.read_excel(caminho_arquivo, sheet_name="Dataset")
total_imgs = len(df)
print(f'Total de imagens para análise: {total_imgs}')

# ETAPA 3: CÁLCULO DA DISTÂNCIA DE MAHALANOBIS (O "Q_HAZE_SCORE")
# -----
if use_widgets:
    progresso.value = 20
    status.value = "<b>Calculando assinaturas espectrais (Mahalanobis)...</b>"

# Features usadas para definir a "assinatura" da imagem (Bandas visíveis + Índices)
cols_analise = ['B02_MEAN', 'B03_MEAN', 'B04_MEAN', 'B08_MEAN', 'NDVI', 'NDWI']

# Remove linhas com dados faltantes para o cálculo (serão D)
df_valid = df.dropna(subset=cols_analise).copy()

# Inicializa colunas
df_valid['MAHALANOBIS'] = 0.0
df_valid['P_VALUE'] = 0.0

# Loop por Estação (O padrão de "imagem boa" muda conforme a estação)
estacoes = df_valid['ESTACAO'].unique()

```

```

print("\n--- Análise de Outliers por Estação ---")
for estacao in estacoes:
    # Pega apenas imagens dessa estação
    mask = df_valid['ESTACAO'] == estacao
    sub_data = df_valid[mask][cols_analise]

    # Precisamos de imagens suficientes para inverter a matriz de covariância
    if len(sub_data) < len(cols_analise) + 2:
        print(f"⚠️ Estação {estacao}: Poucas imagens ({len(sub_data)}). Usando fallback simplificado (Z-score do NDVI)")
        ndvi_mean = sub_data['NDVI'].mean()
        ndvi_std = sub_data['NDVI'].std()
        dists = abs(sub_data['NDVI'] - ndvi_mean) / (ndvi_std + 1e-6)
        df_valid.loc[mask, 'MAHALANOBIS'] = dists
        df_valid.loc[mask, 'P_VALUE'] = 0.5 # Neutro
        continue

    # 1. Matriz de Covariância e Inversa
    cov_matrix = np.cov(sub_data.values.T)
    try:
        inv_cov_matrix = inv(cov_matrix)
    except:
        inv_cov_matrix = np.linalg.pinv(cov_matrix) # Pseudo-inversa se singular

    # 2. Média (Centróide)
    mean_dist = np.mean(sub_data.values, axis=0)

    # 3. Calcula Distância para cada imagem
    dists = []
    for i, row in sub_data.iterrows():
        p = row.values
        d = mahalanobis(p, mean_dist, inv_cov_matrix)
        dists.append(d)

    df_valid.loc[mask, 'MAHALANOBIS'] = dists

    # 4. Converte distância em P-Valor (Probabilidade Chi-Quadrado)
    # P-valor baixo (< 0.01) significa que é um Outlier (Haze, Sombra, Erro)
    df_valid.loc[mask, 'P_VALUE'] = 1 - chi2.cdf(np.array(dists)**2, df=len(cols_analise))

    print(f"✓ {estacao}: {len(sub_data)} imagens processadas.")

# ETAPA 4: CÁLCULO DOS SCORES (A MATEMÁTICA DA SUA IMAGEM)
# -----
if use_widgets:
    progresso.value = 60
    status.value = "<b>Calculando Scores (Nuvens vs Haze)...</b>"

# Junta os cálculos de volta no DF principal
df = df.merge(df_valid[['SCENE_ID', 'MAHALANOBIS', 'P_VALUE']], on='SCENE_ID', how='left')

# 1. SCORE DE NUVENS (0 a 100)
# Quanto menos nuvem, maior a nota.
# Fórmula: 100 - (Porcentagem * Fator). Se nuvem > 20%, nota cai rápido.
def calc_nuvem_score(pct):
    if pd.isna(pct): return 0
    score = 100 - (pct * 2.5) # Ex: 10% nuvem = 75 score. 0% = 100 score.
    return max(0, min(100, score))

df['Q_NUVENS_SCORE'] = df['CLOUD_PCT'].apply(calc_nuvem_score).round(1)

```

```

# 2. SCORE DE HAZE (0 a 100) - Baseado no Mahalanobis P-Value
# Se P-Value > 0.001 (imagem estatisticamente normal), ganha 100.
# Se for outlier extremo, ganha nota baixa.
def calc_haze_score(pval):
    if pd.isna(pval): return 0
    if pval > 0.001: return 100 # É uma imagem "da turma", normal.
    # Se for outlier, penaliza Logaritmicamente ou Linear
    return max(0, 100 - (100 * (0.001 - pval) * 1000)) # Penalidade severa para outliers

df['Q_HAZE_SCORE'] = df['P_VALUE'].apply(calc_haze_score).round(1)

# 3. SCORE FINAL (PONDERADO)
# Conforme engenharia reversa da sua imagem: 70% NUVEM + 30% HAZE
df['QUALIDADE_SCORE'] = (df['Q_NUVENS_SCORE'] * 0.7) + (df['Q_HAZE_SCORE'] * 0.3)
df['QUALIDADE_SCORE'] = df['QUALIDADE_SCORE'].round(1)

# ETAPA 5: CLASSIFICAÇÃO (A, B, C, D)
# -----
if use_widgets:
    progresso.value = 80
    status.value = "<b>Atribuindo classes A/B/C/D...</b>"

def classificar(row):
    score = row['QUALIDADE_SCORE']
    nuvem = row['CLOUD_PCT']

    # Regra Hard: Se tiver muita nuvem, nunca será A ou B
    if nuvem > 20: return 'D'

    # Classificação pelo Score Ponderado
    if score >= 90: return 'A'      # Excelente
    elif score >= 80: return 'B'    # Muito Boa
    elif score >= 70: return 'C'    # Aceitável/Marginal
    else: return 'D'              # Ruim

df['QUALIDADE_CLASSE'] = df.apply(classificar, axis=1)

# ETAPA 6: VISUALIZAÇÃO E SALVAMENTO
# -----
if use_widgets:
    progresso.value = 90
    status.value = "<b>Gerando relatório visual...</b>"

# Gráfico de Dispersão (Visualizar a separação das classes)
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='NDVI', y='QUALIDADE_SCORE', hue='QUALIDADE_CLASSE',
                 palette={'A': 'green', 'B': 'blue', 'C': 'orange', 'D': 'red'},
                 plt.axhline(90, color='green', linestyle='--', alpha=0.5, label='Limite A')
                 plt.axhline(80, color='blue', linestyle='--', alpha=0.5, label='Limite B')
                 plt.title('Classificação Final das Imagens (Score vs NDVI)')
                 plt.grid(True, alpha=0.3)
                 plt.savefig("RESULTADOS/grafico_classificacao_final.png")
                 plt.close()

# Salva o Excel atualizado (Sobrescreve o anterior)
df.to_excel(caminho_arquivo, index=False, sheet_name="Dataset")

# Relatório Final no Console
counts = df['QUALIDADE_CLASSE'].value_counts().sort_index()

```

```

print("\n" + "*60)
print("RESUMO DA CLASSIFICAÇÃO FINAL:")
print("-" * 60)
print(f"{'CLASSE':<10} | {'QTD':<10} | {'DESCRIÇÃO'}")
print("-" * 60)
descricaoes = {'A': 'Excelente (Download Prioritário)', 'B': 'Boa', 'C': 'Marginal', 'D': 'Péssima'}
for cls in ['A', 'B', 'C', 'D']:
    qtd = counts.get(cls, 0)
    print(f"{cls:<10} | {qtd:<10} | {descricaoes[cls]}")
print("=* 60)

# ETAPA 7: FINALIZAÇÃO
# -----
tempo_total = time.time() - inicio_geral

horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

print("\n" + "*110)
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=*110)
=====
```

```

=====
=====
PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS
    Processamento Estatístico: Classificação de Qualidade (Mahalanobis + Scores)
        Avaliação da Qualidade do Dataset para tomada de decisão de Download
=====
=====
VBox(children=(HTML(value='<b>Lendo planilha e preparando downloads A/B/C...</b>'), FloatProgress(value=0.0, d...
```

Lendo dados de: RESULTADOS/DATASET_S2_USINA_PERY.xlsx
 Total de imagens para análise: 150

--- Análise de Outliers por Estação ---

✓ primavera: 32 imagens processadas.

⚠️ Estação verão: Poucas imagens (7). Usando distância Euclidiana simplificada.

a.

✓ outono: 50 imagens processadas.

✓ inverno: 61 imagens processadas.

=====
RESUMO DA CLASSIFICAÇÃO FINAL:

CLASSE	QTD	DESCRIÇÃO
A	97	Excelente (Download Prioritário)
B	29	Boa
C	13	Marginal
D	11	Descartar

=====

=====
 =====
 Tempo de execução → 00:00:00.33 (hh:mm:ss)

Finalizado em → 06/12/2025 às 19:27:37

```
In [9]: # =====
# BLOCO 2: EXECUÇÃO - LETRA A
#           Download das imagens Classe A, B e C (com preview PNG)
# OBJETIVO: Ler a planilha consolidada, filtrar A/B/C
#           Baixar apenas as imagens válidas da planilha
#           Criar pastas: RESULTADOS/ANO/ESTACAO/
#           Salvar GeoTIFF 7 bandas (B02,B03,B04,B08,B11,B12 + dataMask) + p
# =====

# ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO
#
print("=" * 110)
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")
print("          Asseguração Independente de Relatórios ESG via Dados Exógenos Sustentabilidade")
print("          Download das imagens Sentinel-2 – Classe A B C (via planilha consolidada)")
print("=" * 110)

# ETAPA 2: BARRA DE PROGRESSO (Jupyter/Colab)
#
try:
    progresso = widgets.FloatProgress(
        value=0, min=0, max=100,
        description='Progresso:',
        style={'bar_color': '#1f77b4'},
        layout=widgets.Layout(width='80%', height='30px')
    )
    status = widgets.HTML(value="Lendo planilha e preparando downloads A/B/C.")
    display(widgets.VBox([status, progresso]))
    use_widgets = True
except Exception:
    use_widgets = False
    print("Widgets não disponíveis - executando em modo texto")
```

```

inicio_geral = time.time()

# ETAPA 3: CARREGAR PLANILHA E FILTRAR CLASSE A/B/C
# -----
df = pd.read_excel("RESULTADOS/DATASET_S2_USINA_PERY.xlsx", sheet_name="Dataset")
df['DATA_IMG'] = pd.to_datetime(df['DATA_IMG'], errors='coerce')
df_abc = df[df['QUALIDADE_CLASSE'].isin(['A', 'B', 'C'])].copy().reset_index(drop=True)

total_linhas = len(df_abc)
processados = 0

evalscript_multispectral_bruto = EVALSCRIPT_7_BANDAS # Evalscript 7 banda

# ETAPA 4: LOOP POR LINHA DA PLANILHA
# -----
for idx, row in df_abc.iterrows():

    # barra de progresso
    if use_widgets:
        processados += 1
        progresso.value = (processados / total_linhas) * 100
        status.value = f"<b>Baixando imagem {processados}/{total_linhas} ({row['ANO']}-{row['ESTACAO']}-{row['SCENE_ID']})</b>"

    ano = int(row['ANO'])
    estacao = row['ESTACAO']
    cena_id = row['SCENE_ID']

    # cria pasta destino RESULTADOS/ANO/ESTACAO/
    pasta_destino = os.path.join("RESULTADOS", str(ano), estacao)
    os.makedirs(pasta_destino, exist_ok=True)

    # intervalo de busca = ±1 dia da data exata
    data_ini = (row['DATA_IMG'] - pd.Timedelta(days=1)).strftime("%Y-%m-%d")
    data_fim = (row['DATA_IMG'] + pd.Timedelta(days=1)).strftime("%Y-%m-%d")

    try:
        req = SentinelHubRequest(
            evalscript=evalscript_multispectral_bruto,
            input_data=[
                SentinelHubRequest.input_data(
                    DataCollection.SENTINEL2_L2A,
                    time_interval=(data_ini, data_fim),
                    other_args={"dataFilter": {"maxCloudCoverage": 30}}
                )
            ],
            responses=[SentinelHubRequest.output_response("default", MimeType.TIFF),
                       bbox=aoi_bbox,
                       size=(400, 400),
                       config=config
            ]
        )

        dados = req.get_data()[0]

        bandas = dados[:, :, :6].astype(np.float32)
        mask_raw = dados[:, :, 6]
        mask = mask_raw > 0.5

        percentual_valido = mask.mean()
        if percentual_valido < 0.3:

```

```

        raise ValueError(f"Poucos pixels válidos: {percentual_valido:.1%}")

    # RGB (B04,B03,B02)
    rgb = bandas[:, :, [2,1,0]] / 10000.0

    p_low, p_high = np.percentile(rgb[mask], (2, 98), axis=0)
    rgb_norm = np.zeros_like(rgb)
    for b in range(3):
        x = rgb[:, :, b]
        x = (x - p_low[b]) / (p_high[b] - p_low[b] + 1e-10)
        rgb_norm[:, :, b] = np.clip(x, 0, 1)

    rgb_gamma = np.power(rgb_norm, 0.7)
    img_rgb = (rgb_gamma * 255).astype(np.uint8)
    img_rgb[~mask] = [0,0,0]

    # salvar PNG
    path_png = os.path.join(pasta_destino, f"{cena_id}.png")
    img_bgr = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR)
    cv2.imwrite(path_png, img_bgr)

    # salvar TIF
    dados_geotiff = np.concatenate([bandas, mask_raw[...,None]], axis=-1)
    path_tif = os.path.join(pasta_destino, f"{cena_id}_6bandas.tif")

    west, south, east, north = aoi_bbox
    transform = from_bounds(west, south, east, north, 400, 400)

    with rasterio.open(
        path_tif, 'w',
        driver='GTiff',
        height=400, width=400,
        count=7, dtype='float32',
        crs='EPSG:4326',
        transform=transform,
        compress='deflate'
    ) as dst:
        for b in range(7):
            dst.write(dados_geotiff[:, :, b], b+1)

except Exception as e:
    print(f"X Falhou {row['DATA_IMG'].date()} - {estacao} - {cena_id} → {e}")
    continue

# ETAPA 5: FINALIZAÇÃO
# -----
tempo_total = time.time() - inicio_geral

horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

print("\n" + "="*110)
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=*110")

```

```
=====
=====
PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS
    Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais
        Download das imagens Sentinel-2 - Classe A B C (via planilha consolidada)
a)
=====
```

```
=====
=====
VBox(children=(HTML(value='<b>Lendo planilha e preparando downloads A/B/C...</b>'), FloatProgress(value=0.0, d...
=====
```

```
=====
Tempo de execução → 00:08:28.70 (hh:mm:ss)
Finalizado em → 06/12/2025 às 19:36:29
=====
```

```
=====
=====
```

In [9]:

```
# -----
# BLOCO 2:      EXECUÇÃO – LETRA B
#                  Teste K-Means (k=4) + Distância de Mahalanobis
# OBJETIVO:     Testar a metodologia com uma imagem para garantir a robustez da
# -----
# Import complementar
from PIL import Image as PILImage
import io
import base64

# ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO
# -----
print("=" * 110)
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")
print("          Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais")
print("          Teste K-Means (k=4) + Distância de Mahalanobis")
print("=" * 110)

# ETAPA 2: CAMINHOS DAS IMAGENS DE TESTE (INVERNO 2025)
# -----
caminho_tiff = r"C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c.tif"
caminho_png = r"C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c.png"

inicio = time.time()      # Iniciar cronômetro

# ETAPA 3: CARREGAR E PROCESSAR OS DADOS
# -----
# Carregar imagem TIFF (dados multibanda do satélite)
with rasterio.open(caminho_tiff) as src:
    arr = src.read().astype(np.float32) # Lê todas as bandas do arquivo TIFF e as converte para float32

    # Transpor as dimensões: de (bandas, altura, Largura) para (altura, Largura, bandas)
    arr = np.transpose(arr, (1, 2, 0))

    # Obter dimensões da imagem
    H, W, B = arr.shape # H = altura em pixels, W = Largura em pixels, B = número de bandas

# Tentar carregar imagem PNG (se existir, serve como referência visual)
try:
    img_png = cv2.imread(caminho_png) # Carrega imagem PNG usando OpenCV
    img_png_rgb = cv2.cvtColor(img_png, cv2.COLOR_BGR2RGB) # Converte de BGR (p
    png_loaded = True # Flag indicando que PNG foi carregado com sucesso
except:
```

```

# Se PNG não existir ou falhar, criar uma imagem alternativa usando a primei
img_alt = (arr[:, :, 0] - arr[:, :, 0].min()) / (arr[:, :, 0].max() - arr[:, 0]
img_png_rgb = np.stack([img_alt, img_alt, img_alt], axis=-1).astype(np.uint8)
png_loaded = False # Flag indicando que PNG não foi carregado

# Preparar dados para processamento
px = arr.reshape(-1, B) # Redimensiona para matriz 2D: (pixels_totais x bandas)

# Criar máscara de pixels válidos (remove valores NaN ou infinitos)
mask_valid = np.all(~np.isfinite(px), axis=1) # True para pixels com todos os va

# Filtrar apenas pixels válidos
px_valid = px[mask_valid]

# Calcular estatísticas de contagem
total_pixels = H * W # Número total de pixels na imagem
total_valid = len(px_valid) # Número de pixels com valores válidos (sem NaN/inf

# Normalizar os dados (importante para K-Means e Mahalanobis)
mean_b = px_valid.mean(axis=0) # Calcula média por banda
std_b = px_valid.std(axis=0) + 1e-10 # Calcula desvio padrão por banda (+ epsilon)
px_norm = (px_valid - mean_b) / std_b # Normalização z-score: (valor - média) / desvio padrão

# ETAPA 4: PROCESSAR K-MEANS
# -----
k = 4 # Definir número de clusters (classes) para o K-Means

# Inicializar algoritmo K-Means com 4 clusters, seed fixa para reprodução, 10 iterações
km = KMeans(n_clusters=k, random_state=42, n_init=10)

# Ajustar modelo K-Means aos dados normalizados (encontrar clusters)
km.fit(px_norm)

# Obter rótulos de cluster para cada pixel válido (0, 1, 2, ou 3)
labels_valid = km.labels_

# Criar mapa completo de rótulos (incluindo pixels inválidos marcados como -1)
labels_map = np.full(H * W, -1, int) # Inicializa com -1 em todos os pixels
labels_map[mask_valid] = labels_valid # Atribui rótulos apenas aos pixels válidos
labels_map = labels_map.reshape(H, W) # Remodela para formato 2D da imagem

# Calcular propriedades espectrais médias por classe (para identificar semântica)
class_properties = []
for cls in range(k):
    idx = (labels_valid == cls) # Máscara booleana para pixels desta classe
    if idx.sum() == 0: # Se classe vazia, pular
        class_properties.append([0, 0, 0, 0])
        continue

    # Usar valores originais (não normalizados) das bandas para análise espectral
    px_cls = px_valid[idx]

    # Calcular média por banda (assumindo bandas Sentinel-2: B2=Blue, B3=Green, B4=Red, B8=Infravermelho Próximo)
    mean_b2 = px_cls[:, 0].mean() if B > 0 else 0 # Banda Azul (Blue)
    mean_b3 = px_cls[:, 1].mean() if B > 1 else 0 # Banda Verde (Green)
    mean_b4 = px_cls[:, 2].mean() if B > 2 else 0 # Banda Vermelha (Red)
    mean_b8 = px_cls[:, 3].mean() if B > 3 else 0 # Banda Infravermelho Próximo

    # Calcular NDVI aproximado (Normalized Difference Vegetation Index)
    # NDVI = (NIR - Red) / (NIR + Red) - alto para vegetação saudável

```

```

ndvi = (mean_b8 - mean_b4) / (mean_b8 + mean_b4 + 1e-10) if (mean_b8 + mean_b4) > 0 else 0
# Calcular NDWI aproximado (Normalized Difference Water Index)
# NDWI = (Green - NIR) / (Green + NIR) - alto para corpos d'água
ndwi = (mean_b3 - mean_b8) / (mean_b3 + mean_b8 + 1e-10) if (mean_b3 + mean_b8) > 0 else 0
# Armazenar propriedades: [NDVI, NDWI, valor médio NIR, contagem de pixels]
class_properties.append([ndvi, ndwi, mean_b8, idx.sum()])

# Criar sistema de pontuação para identificar semanticamente cada cluster
scores = []
for i, props in enumerate(class_properties):
    ndvi, ndwi, nir, count = props # Desempacotar propriedades

    # Pontuação para Água: alto NDWI, baixo NDVI
    score_agua = ndwi - ndvi

    # Pontuação para Vegetação densa: alto NDVI, alto NIR
    score_veg_densa = ndvi + (nir / 10000 if nir > 0 else 0)

    # Pontuação para Vegetação rasteira: NDVI médio (70% do valor)
    score_veg_rasteira = ndvi * 0.7

    # Pontuação para Solo: baixo NDVI e baixo NDWI
    score_solo = -ndvi - ndwi

    # Armazenar pontuações para esta classe
    scores.append({
        'original_idx': i, # Índice original do cluster
        'agua': score_agua,
        'veg_densa': score_veg_densa,
        'veg_rasteira': score_veg_rasteira,
        'solo': score_solo,
        'count': count # Número de pixels na classe
    })

# Ordenar e mapear cada cluster para uma classe semântica baseado nas pontuações
# Identificar classe com maior pontuação para ÁGUA
scores_sorted = sorted(scores, key=lambda x: x['agua'], reverse=True)
agua_idx = scores_sorted[0]['original_idx']

# Identificar classe com maior pontuação para VEGETAÇÃO DENSA
scores_sorted = sorted(scores, key=lambda x: x['veg_densa'], reverse=True)
veg_densa_idx = scores_sorted[0]['original_idx']
if veg_densa_idx == agua_idx: # Evitar duplicação
    veg_densa_idx = scores_sorted[1]['original_idx']

# Identificar classe com maior pontuação para SOLO
scores_sorted = sorted(scores, key=lambda x: x['solo'], reverse=True)
solo_idx = scores_sorted[0]['original_idx']
if solo_idx in [agua_idx, veg_densa_idx]: # Evitar duplicação
    solo_idx = scores_sorted[1]['original_idx']

# A quarta classe restante é Vegetação Rasteira
restantes = [i for i in range(k) if i not in [agua_idx, veg_densa_idx, solo_idx]]
veg_rasteira_idx = restantes[0] if restantes else 2 # Fallback se não encontrar

# Criar mapeamento de cores: cluster original → classe semântica (0-3)
color_mapping = {
    agua_idx: 0, # 0 = Água (Azul)
    veg_densa_idx: 1, # 1 = Vegetação Densa (Verde)
    veg_rasteira_idx: 2, # 2 = Vegetação Rasteira (Amarelo)
    solo_idx: 3 # 3 = Solo (Cinza)
}

```

```

    veg_densa_idx: 1,      # 1 = Vegetação Densa (Verde)
    veg_rasteira_idx: 2,   # 2 = Vegetação Rasteira (Amarelo)
    solo_idx: 3           # 3 = Solo Exposto (Marrom)
}

# Reordenar rótulos para usar as cores corretas (mapeamento semântico)
labels_map_corrigido = np.full_like(labels_map, -1) # Novo mapa com -1
for cls_original, cls_corrigido in color_mapping.items():
    # Substituir índice do cluster original pelo índice semântico corrigido
    labels_map_corrigido[labels_map == cls_original] = cls_corrigido

# Substituir mapa original pelo corrigido
labels_map = labels_map_corrigido

# Atualizar também o vetor de rótulos válidos (1D)
labels_valid_corrigido = np.full_like(labels_valid, -1)
for cls_original, cls_corrigido in color_mapping.items():
    labels_valid_corrigido[labels_valid == cls_original] = cls_corrigido
labels_valid = labels_valid_corrigido # Atualizar variável

# Definir nomes e cores RGB para cada classe semântica
class_names = ["Água", "Vegetação Densa", "Vegetação Rasteira", "Solo Exposto"]
class_colors_rgb = [[0, 71, 171], [35, 139, 69], [255, 215, 0], [150, 75, 0]]

# Calcular estatísticas por classe (agora com mapeamento semântico correto)
class_stats = []
for cls in range(k):
    n_class = (labels_valid == cls).sum() # Contar pixels desta classe
    perc_class = n_class / total_valid * 100 # Calcular porcentagem
    class_stats.append((cls, n_class, perc_class)) # Armazenar tupla (índice, c

# Criar imagem colorida para visualização do resultado do K-Means
mapa_kmeans = np.zeros((H, W, 3), dtype=np.uint8) # Imagem vazia 3 canais (RGB)
for cls in range(k):
    # Pintar todos os pixels desta classe com a cor RGB correspondente
    mapa_kmeans[labels_map == cls] = class_colors_rgb[cls]

# ETAPA 5: PROCESSAR MAHALANOBIS
# -----
stats = {} # Dicionário para armazenar estatísticas de Mahalanobis por classe

# Para cada classe (0=Água, 1=Vegetação densa, 2=Vegetação rasteira, 3=Solo)
for cls in range(k):
    # Identificar pixels que pertencem a esta classe
    idx = (labels_valid == cls)

    # Extrair dados normalizados dos pixels desta classe
    px_cls = px_norm[idx]

    # Contar quantos pixels pertencem a esta classe
    N_cls = len(px_cls)

    # Calcular vetor média (centroide) dos pixels desta classe
    mu = px_cls.mean(axis=0) # Média por banda

    # Verificar se há pixels suficientes para calcular matriz de covariância
    if N_cls > 1:
        # Calcular matriz de covariância dos dados da classe
        # rowvar=False significa que cada linha é uma observação (pixel), cada coluna é uma variável
        cov = np.cov(px_cls, rowvar=False)

```

```

# Regularização: adicionar pequeno valor à diagonal para evitar matriz singular
cov_reg = cov + np.eye(B) * 1e-6

try:
    # Tentar calcular inversa da matriz de covariância
    cov_inv = inv(cov_reg)
except:
    # Se falhar (matriz singular), usar pseudoinversa como fallback
    cov_inv = np.linalg.pinv(cov_reg)
else:
    # Se só há 1 pixel na classe, usar matriz identidade (não há variabilidade)
    cov_inv = np.eye(B)

# Armazenar estatísticas: (vetor média, matriz de covariância inversa, número de pixels)
stats[cls] = (mu, cov_inv, N_cls)

# Inicializar mapa binário de anomalias (0=normal, 1=anômalo)
anom_map = np.zeros((H, W), int)

# Lista para armazenar detalhes estatísticos de cada classe
anom_details = []

# Para cada classe, calcular distâncias de Mahalanobis e identificar anomalias
for cls in range(k):
    # Verificar se classe existe no dicionário stats
    if cls not in stats:
        continue

    # Criar máscara 2D para pixels desta classe na imagem completa
    mask = (labels_map == cls)

    # Criar máscara 1D para pixels válidos desta classe
    mask_valid_cls = (labels_valid == cls)

    # Pular se não houver pixels nesta classe
    if mask_valid_cls.sum() == 0:
        continue

    # Extrair dados normalizados dos pixels desta classe
    px_cls = px_norm[mask_valid_cls]

    # Recuperar estatísticas calculadas anteriormente
    mu, cov_inv, N_cls = stats[cls]

    # Calcular diferenças entre cada pixel e o centroide da classe
    diff = px_cls - mu # Shape: (N_cls, B)

    # Calcular produto matricial: diff × cov_inv
    temp = diff @ cov_inv # Shape: (N_cls, B)

    # Calcular distância de Mahalanobis quadrada e depois raiz quadrada
    # Fórmula:  $\sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}$ 
    md_vals = np.sqrt(np.sum(temp * diff, axis=1)) # Shape: (N_cls,)

    # Verificar se há dados suficientes para calcular percentil
    if len(md_vals) > 5:
        # Definir limiar como percentil 95 (5% mais extremos são anomalias)
        limiar = np.percentile(md_vals, 95)

```

```

# Identificar quais pixels são anômalos (distância > limiar)
anom_local = md_vals > limiar

# Contar anomalias nesta classe
n_anom = anom_local.sum()

# Calcular porcentagem de anomalias na classe
perc_anom = n_anom / N_cls * 100

# Converter máscara de anomalias Local (1D) para mapa completo (2D)
anom_full = np.zeros(H * W, bool) # Vetor 1D do tamanho da imagem

# Encontrar índices Lineares dos pixels desta classe
idx_flat = np.where(mask.flatten())[0]

# Atribuir anomalias nas posições corretas
anom_full[idx_flat] = anom_local

# Remodelar para formato 2D da imagem
anom_full = anom_full.reshape(H, W)

# Adicionar anomalias desta classe ao mapa global (operador OR binário)
anom_map = anom_map | anom_full

# Armazenar detalhes estatísticos para esta classe
# (índice classe, n_pixels, média distâncias, desvio padrão, limiar, n_a
anom_details.append((cls, N_cls, md_vals.mean(), md_vals.std(), limiar,

# Criar mapa visual de anomalias (sobreposição sobre classificação K-Means)
mapa_anomalias = mapa_kmeans.copy() # Começar com cópia do mapa K-Means colorido

# Pintar pixels anômalos de vermelho [R=255, G=0, B=0]
mapa_anomalias[anom_map == 1] = [255, 0, 0]

# Define o diretório geral para salvar tudo junto
diretorio_saida = r"C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.

# Pega apenas o nome do arquivo original (sem o caminho todo)
nome_arquivo_base = os.path.splitext(os.path.basename(caminho_png))[0]

# 1. Salva a imagem do K-Means (Essa será o "professor" do YOLO)
arquivo_kmeans = os.path.join(diretorio_saida, f"{nome_arquivo_base}_KMEANS.png"
PILImage.fromarray(mapa_kmeans).save(arquivo_kmeans)
print(f"    [SALVO] Gabarito K-Means: {arquivo_kmeans}")

# 2. Salva a imagem do Mahalanobis (Para comparação visual)
arquivo_mahal = os.path.join(diretorio_saida, f"{nome_arquivo_base}_MAHALANOBIS."
PILImage.fromarray(mapa_anomalias).save(arquivo_mahal)
print(f"    [SALVO] Anomalias Mahalanobis: {arquivo_mahal}")

# ETAPA 6: EXIBIR IMAGENS EM GRID
# -----
# Função para converter imagem numpy para base64 (para exibir no HTML)
def img_to_base64(img_array):
    img_pil = PILImage.fromarray(img_array)
    buffered = io.BytesIO()
    img_pil.save(buffered, format="PNG")
    return base64.b64encode(buffered.getvalue()).decode()

# Converter imagens para base64

```

```

# Separadores de casa de milhar
def formatar_numero_br(num):
    """Formata número com separador de milhar como PONTO e decimal como VÍRGULA"""
    # Primeiro formata no padrão americano
    formato_us = f"{num:,.0f}"
    # Inverte: vírgula -> ponto, ponto -> vírgula
    return formato_us.replace(",", "X").replace(".", ",").replace("X", ".") 

def formatar_percentual_br(num):
    """Formata percentual com VÍRGULA decimal"""
    return f"{num:.1f}".replace(".", ",") 

# Criar HTML para grade de imagens
grid_html = """
<div style="display: flex; justify-content: space-between; flex-wrap: wrap; gap: 10px">

    <!-- IMAGEM ORIGINAL -->
    <div style="border: 2px solid #1f77b4; border-radius: 10px; padding: 15px; background-color: #f0f8ff; margin-bottom: 10px">
        <h4 style="color: #1f77b4; margin-bottom: 10px; font-size: 16px;">IMAGEM ORIGINAL</h4>
        
        <div style="margin-top: 10px; font-size: 12px; text-align: left;">
            <div style="margin: 5px 0;">
                <span style="font-weight: bold;">Dimensões:</span> {H} x {W} pixels
            </div>
            <div style="margin: 5px 0;">
                <span style="font-weight: bold;">Tipo:</span> {'PNG original' if ext == 'png' else 'JPEG' if ext == 'jpg' else 'Outro'}
            </div>
            <div style="margin: 5px 0;">
                <span style="font-weight: bold;">Pixels totais:</span> {formatar_numero_br(total_pixels)}
            </div>
            <div style="margin: 5px 0;">
                <span style="font-weight: bold;">Pixels válidos:</span> {formatar_numero_br(valid_pixels)}
            </div>
            <div style="margin: 5px 0; padding: 8px; background: #f0f8ff; border: 1px solid #1f77b4; border-radius: 5px;">
                <span style="font-weight: bold;">Resolução:</span> 10m/pixel (Selecione a grade desejada)
                <span style="font-weight: bold;">Área aproximada:</span> {(H*10) x (W*10)} km²
            </div>
        </div>
    </div>
</div>

    <!-- K-MEANS -->
    <div style="border: 2px solid #2ca02c; border-radius: 10px; padding: 15px; background-color: #f0f8ff; margin-bottom: 10px">
        <h4 style="color: #2ca02c; margin-bottom: 10px; font-size: 16px;">CLASSIFICAÇÃO</h4>
        
        <div style="margin-top: 10px; font-size: 12px; text-align: left;">
            <!-- LEGENDA -->
            <div style="margin-bottom: 10px; padding-bottom: 8px; border-bottom: 1px solid #2ca02c; display: flex; align-items: center; margin: 4px 0;">
                <div style="width: 15px; height: 15px; background: rgb(0,71,161); border-radius: 50%; margin-right: 10px;"/>
                <span>Água</span>
            </div>
            <div style="display: flex; align-items: center; margin: 4px 0;">
                <div style="width: 15px; height: 15px; background: rgb(35,136,136); border-radius: 50%; margin-right: 10px;"/>
                <span>Vegetação densa</span>
            </div>
        </div>
    </div>
</div>

```

```

        </div>
        <div style="display: flex; align-items: center; margin: 4px 0;">
            <div style="width: 15px; height: 15px; background: rgb(255,2
                <span>Vegetação rasteira</span>
            </div>
            <div style="display: flex; align-items: center; margin: 4px 0;">
                <div style="width: 15px; height: 15px; background: rgb(150,7
                    <span>Solo exposto</span>
                </div>
            </div>

        <!-- ESTATÍSTICAS -->
        <div style="margin: 8px 0;">
            <div style="display: flex; justify-content: space-between; margin
                <span>Água:</span>
                <span style="font-weight: bold;">{formatar_numero_br(class_s
            </div>
            <div style="display: flex; justify-content: space-between; margin
                <span>Vegetação densa:</span>
                <span style="font-weight: bold;">{formatar_numero_br(class_s
            </div>
            <div style="display: flex; justify-content: space-between; margin
                <span>Vegetação rasteira:</span>
                <span style="font-weight: bold;">{formatar_numero_br(class_s
            </div>
            <div style="display: flex; justify-content: space-between; margin
                <span>Solo exposto:</span>
                <span style="font-weight: bold;">{formatar_numero_br(class_s
            </div>
        </div>

        <!-- TOTAL -->
        <div style="margin-top: 8px; padding-top: 8px; border-top: 1px solid
            <div style="display: flex; justify-content: space-between; font-
                <span>TOTAL:</span>
                <span>{formatar_numero_br(total_valid)} px (100,0%)</span>
            </div>
        </div>
    </div>
</div>

<!-- ANOMALIAS MAHALANOBIS -->
<div style="border: 2px solid #d62728; border-radius: 10px; padding: 15px; b
    <h4 style="color: #d62728; margin-bottom: 10px; font-size: 16px;">ANOMAL
    
        <!-- LEGENDA -->
        <div style="margin-bottom: 10px; padding-bottom: 8px; border-bottom:
            <div style="display: flex; align-items: center; margin: 4px 0;">
                <div style="width: 15px; height: 15px; background: rgb(255,0
                    <span>Pixel anômalo (detectado)</span>
                </div>
                <div style="display: flex; align-items: center; margin: 4px 0;">
                    <div style="width: 15px; height: 15px; background: linear-gr
                        <span>Cores de fundo (classes K-Means)</span>
                    </div>
                </div>
            </div>
        <!-- ESTATÍSTICAS -->

```

```

<div style="margin: 8px 0;">
    <div style="display: flex; justify-content: space-between; margin: 0 auto; width: fit-content; font-weight: bold; border-bottom: 1px solid black; padding-bottom: 5px;">
        <span>Anomalias detectadas:</span>
        <span>Percentual da imagem:</span>
        <span>Limiar de detecção:</span>
        <span>Percentil 95</span>
    </div>
    <div style="display: flex; justify-content: space-between; margin: 0 auto; width: fit-content; font-weight: bold; border-bottom: 1px solid black; padding-bottom: 5px;">
        <span>Método estatístico:</span>
        <span>Mahalanobis</span>
    </div>
</div>

<!-- INFORMAÇÃO ADICIONAL -->
<div style="margin-top: 10px; padding: 8px; background: #ffff5f5; border: 1px solid black; border-radius: 5px; width: fit-content; margin: 0 auto;">
    <div style="font-weight: bold; color: #d62728; margin-bottom: 4px;">Pontos vermelhos indicam pixels estatisticamente atípicos dentro da máscara</div>
</div>
</div>
<!-- TABELA DAS ANOMALIAS POR CLASSE -->
<div style="background-color: #fff; border: 1px solid black; padding: 10px; border-radius: 5px; width: fit-content; margin: 0 auto;">
    <h2 style="text-align: center; margin-bottom: 10px;">DISTRIBUIÇÃO DAS ANOMALIAS POR CLASSE</h2>
    <p style="margin-bottom: 15px; color: #666;">Anomalias totais: <b>{formatar_numero_br(anom_map.sum())}</b></p>
    <table style="width: 100%; border-collapse: collapse; margin-top: 10px; font-size: 0.9em;">
        <thead>
            <tr style="background-color: #e9ecef; border-bottom: 1px solid black; text-align: left; padding: 5px; font-weight: bold;">
                <th style="padding: 5px; border: 1px solid #dee2e6; width: 15%;">Classe
                <th style="padding: 5px; border: 1px solid #dee2e6; width: 15%;">Quantidade
                <th style="padding: 5px; border: 1px solid #dee2e6; width: 15%;">Percentual (%)
            </tr>
        <tbody>
            <tr>
                <td style="padding: 5px; border: 1px solid #dee2e6;">1
                <td style="padding: 5px; border: 1px solid #dee2e6;">{anom_map[1]}
                <td style="padding: 5px; border: 1px solid #dee2e6;">{formatar_percentual_br(anom_map[1] / anom_map.sum() * 100)}
            </tr>
            <tr>
                <td style="padding: 5px; border: 1px solid #dee2e6;">2
                <td style="padding: 5px; border: 1px solid #dee2e6;">{anom_map[2]}
                <td style="padding: 5px; border: 1px solid #dee2e6;">{formatar_percentual_br(anom_map[2] / anom_map.sum() * 100)}
            </tr>
            <tr>
                <td style="padding: 5px; border: 1px solid #dee2e6;">3
                <td style="padding: 5px; border: 1px solid #dee2e6;">{anom_map[3]}
                <td style="padding: 5px; border: 1px solid #dee2e6;">{formatar_percentual_br(anom_map[3] / anom_map.sum() * 100)}
            </tr>
            <tr>
                <td style="padding: 5px; border: 1px solid #dee2e6;">4
                <td style="padding: 5px; border: 1px solid #dee2e6;">{anom_map[4]}
                <td style="padding: 5px; border: 1px solid #dee2e6;">{formatar_percentual_br(anom_map[4] / anom_map.sum() * 100)}
            </tr>
        </tbody>
    </table>
</div>

```

```

<tr>
    <td style="padding: 10px; border: 1px solid #dee2e6; font-weight: bold;">
        {class_names[cls]}
    </td>
    <td style="padding: 10px; border: 1px solid #dee2e6; text-align: right;">
        {count}
    <td style="padding: 10px; border: 1px solid #dee2e6; text-align: right;">
        {percentage}
    <td style="padding: 10px; border: 1px solid #dee2e6; text-align: right;">
        {percentage}
    </td>
</tr>
"""

tabela_anomalias += f"""
<tr style="background: #343a40; color: white; font-weight: bold;">
    <td style="padding: 10px; border: 1px solid #dee2e6;">TOTAL</td>
    <td style="padding: 10px; border: 1px solid #dee2e6; text-align: right;">
        {total_count}
    <td style="padding: 10px; border: 1px solid #dee2e6; text-align: right;">
        {total_percent}
    <td style="padding: 10px; border: 1px solid #dee2e6; text-align: right;">
        {total_percent}
    </td>
</tr>
</table>

<div style="margin-top: 15px; padding: 10px; background: #e9eceff; border-radius: 5px; border: 1px solid #dee2e6; font-size: 0.9em;">
    <b>Interpretação:</b> A tabela mostra como as {formatar_numero_br(anomalias)} amostras foram agrupadas em {k} classes. Isso ajuda a identificar em quais classes o K-Means teve mais dificuldade de convergência.
</div>
"""
"""

display(HTML(tabela_anomalias))

# ETAPA 8: FINALIZAÇÃO
# -----
tempo_total = time.time() - inicio # tempo total em segundos

# Converte para o formato HH:MM:SS
horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

print("\n" + "="*110)
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=*110")

```

```

=====
=====
PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS
    Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais
    Teste K-Means (k=4) + Distância de Mahalanobis
=====
=====
[SALVO] Gabarito K-Means: C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Visão Computacional\VC_TF\RESULTADOS\S2C_MSIL2A_20250821T132251_N0511_R038_T22JEQ_20250821T170915_KMEANS.png
[SALVO] Anomalias Mahalanobis: C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Visão Computacional\VC_TF\RESULTADOS\S2C_MSIL2A_20250821T132251_N0511_R038_T22JEQ_20250821T170915_MAHALANOBIS.png

```

IMAGEM ORIGINAL



Dimensões: 400 × 400 pixels

Tipo: PNG original

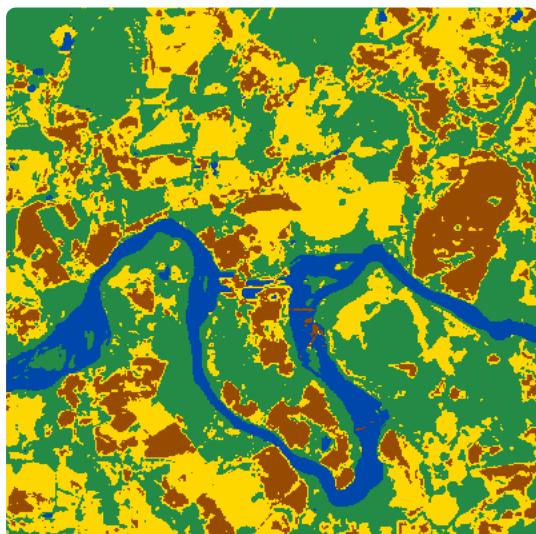
Pixels totais: 160.000 px

Pixels válidos: 160.000 px

Resolução: 10m/pixel (Sentinel-2)

Área aproximada: 1600.0 hectares

CLASSIFICAÇÃO K-MEANS (k=4)



■ Água

■ Vegetação densa

■ Vegetação rasteira

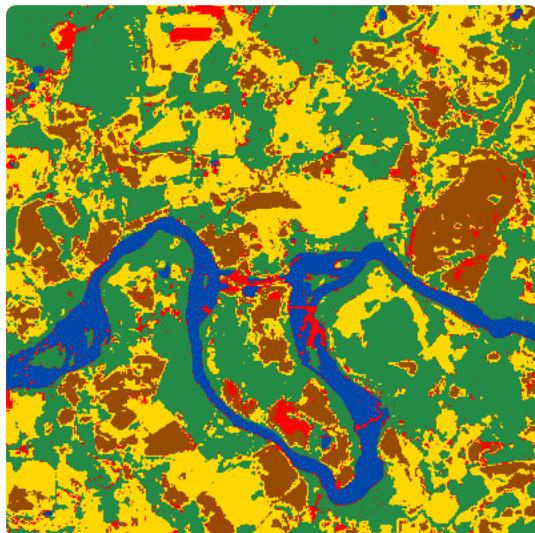
■ Solo exposto

Água:

12.573 px (7,9%)

Vegetação densa:	67.968 px (42,5%)
Vegetação rasteira:	53.134 px (33,2%)
Solo exposto:	26.325 px (16,5%)
TOTAL:	160.000 px (100,0%)

ANOMALIAS MAHALANOBIS



- Pixel anômalo (detectado) 
- Cores de fundo (classes K-Means) 

Anomalias detectadas:	8.002 px
Percentual da imagem:	5,0%
Limiar de detecção:	Percentil 95
Método estatístico:	Mahalanobis

Interpretação:

Pontos vermelhos indicam pixels estatisticamente atípicos dentro de suas respectivas classes, possivelmente representando erros de classificação do K-Means ou áreas de transição entre classes.

 DISTRIBUIÇÃO DAS ANOMALIAS POR CLASSE

Anomalias totais: **8.002 px** (5,0% da imagem)

Classe	Total Pixels	Anomalias	% da Classe
Água	12.573	629	5,0%
Vegetação Densa	67.968	3.399	5,0%
Vegetação Rasteira	53.134	2.657	5,0%
Solo Exposto	26.325	1.317	5,0%
TOTAL	160.000	8.002	5,0%

Interpretação: A tabela mostra como as 8.002 anomalias estão distribuídas entre as 4 classes.

Isso ajuda a identificar em quais classes o K-Means teve mais dificuldade de classificação.

```
=====
=====
Tempo de execução → 00:00:01.28 (hh:mm:ss)
Finalizado em → 07/12/2025 às 13:02:12
=====
```

```
In [ ]: # -----
# BLOCO 2:      EXECUÇÃO - LETRA B
#              Calculo K-Means (k=4) + Distância de Mahalanobis no Dataset
#
# OBJETIVO:    Aplicar a metodologia no Dataset
# -----
```



```
# ETAPA 1: CABEÇALHO INSTITUCIONAL DO PROJETO
# -----
print("=" * 110)
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS")
print("          Asseguração Independente de Relatórios ESG via Dados Exógenos Sa")
print("          Calculo K-Means (k=4) + Distância de Mahalanobis no Dataset")
print("=" * 110)

# ETAPA 2: CAMINHOS DAS IMAGENS DE TESTE
# -----
BASE_PATH = "RESULTADOS" # Pasta raiz com os dados
K_CLUSTERS = 4 # Mesmo k=4 do script anterior
CLASS_NAMES = ["Água", "Vegetação Densa", "Vegetação Rasteira", "Solo Exposto"]
MAHALANOBIS_PERCENTIL = 95 # Percentil para limiar de anomalias

inicio = time.time()      # Iniciar cronômetro
```



```
# ETAPA 3: BARRA DE PROGRESSO (Só funciona no Jupyter/Colab)
# -----
```

```

# Configurações para não truncar o console
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all" # MOSTRA TUDO

# CONFIGURAR PARA MOSTRAR TODO O OUTPUT
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)

# Criação da barra de progresso azul
progress_bar = widgets.FloatProgress(value=0, min=0, max=100, description='Progr',
                                         style={'bar_color': '#0066cc'}, layout=widg
status_label = widgets.HTML(value="Iniciando processamento do dataset de imag")
display(widgets.VBox([status_label, progress_bar]))

# Função para atualizar a barra de progresso
def atualizar_progresso(percentual, mensagem):
    progress_bar.value = percentual
    status_label.value = f"{mensagem} ({percentual:.0f}%)"

# ETAPA 4: FUNÇÕES AUXILIARES (reutilizadas do script anterior)
# -----
def carregar_processar_imagem(caminho_tiff, caminho_png=None):
    """Carrega e processa imagem TIFF e PNG similar ao script anterior."""
    with rasterio.open(caminho_tiff) as src:
        arr = src.read().astype(np.float32)
    arr = np.transpose(arr, (1, 2, 0))
    H, W, B = arr.shape

    # Carregar PNG ou criar alternativa
    if caminho_png and os.path.exists(caminho_png):
        img_png = cv2.imread(caminho_png)
        img_png_rgb = cv2.cvtColor(img_png, cv2.COLOR_BGR2RGB) if img_png is not
    else:
        img_png_rgb = None

    # Preparar dados
    px = arr.reshape(-1, B)
    mask_valid = np.all(np.isfinite(px), axis=1)
    px_valid = px[mask_valid]

    # Normalizar
    mean_b = px_valid.mean(axis=0)
    std_b = px_valid.std(axis=0) + 1e-10
    px_norm = (px_valid - mean_b) / std_b

    return {
        'px_norm': px_norm,
        'px_valid': px_valid,
        'mask_valid': mask_valid,
        'H': H, 'W': W, 'B': B,
        'total_valid': len(px_valid),
        'total_pixels': H * W,
        'img_png': img_png_rgb
    }

def processar_kmeans(dados, k=4):
    """Executa K-Means e mapeamento semântico similar ao script anterior."""
    px_norm = dados['px_norm']

```

```

H, W = dados['H'], dados['W']
mask_valid = dados['mask_valid']
B = dados['B']
px_valid = dados['px_valid']

# K-Means
km = KMeans(n_clusters=k, random_state=42, n_init=10)
km.fit(px_norm)
labels_valid = km.labels_

# Mapa 2D
labels_map = np.full(H * W, -1, int)
labels_map[mask_valid] = labels_valid
labels_map = labels_map.reshape(H, W)

# IDENTIFICAÇÃO SEMÂNTICA DAS CLASSES (igual ao script anterior)
# Calcular propriedades médias por classe
class_properties = []
for cls in range(k):
    idx = (labels_valid == cls)
    if idx.sum() == 0:
        class_properties.append([0, 0, 0, 0])
        continue

    px_cls = px_valid[idx]

    # Médias por banda (assumindo bandas Sentinel-2)
    mean_b2 = px_cls[:, 0].mean() if B > 0 else 0 # Blue
    mean_b3 = px_cls[:, 1].mean() if B > 1 else 0 # Green
    mean_b4 = px_cls[:, 2].mean() if B > 2 else 0 # Red
    mean_b8 = px_cls[:, 3].mean() if B > 3 else 0 # NIR

    # Calcular NDVI e NDWI
    ndvi = (mean_b8 - mean_b4) / (mean_b8 + mean_b4 + 1e-10) if (mean_b8 + mean_b4 + 1e-10) != 0 else 0
    ndwi = (mean_b3 - mean_b8) / (mean_b3 + mean_b8 + 1e-10) if (mean_b3 + mean_b8 + 1e-10) != 0 else 0

    class_properties.append([ndvi, ndwi, mean_b8, idx.sum()])

# Criar pontuação para cada classe
scores = []
for i, props in enumerate(class_properties):
    ndvi, ndwi, nir, count = props

    score_agua = ndwi - ndvi
    score_veg_densa = ndvi + (nir / 10000 if nir > 0 else 0)
    score_veg_rasteira = ndvi * 0.7
    score_solo = -ndvi - ndwi

    scores.append({
        'original_idx': i,
        'agua': score_agua,
        'veg_densa': score_veg_densa,
        'veg_rasteira': score_veg_rasteira,
        'solo': score_solo,
        'count': count
    })

# Ordenar e mapear
# Água
scores_sorted = sorted(scores, key=lambda x: x['agua'], reverse=True)

```

```

agua_idx = scores_sorted[0]['original_idx']

# Vegetação densa
scores_sorted = sorted(scores, key=lambda x: x['veg_densa'], reverse=True)
veg_densa_idx = scores_sorted[0]['original_idx']
if veg_densa_idx == agua_idx:
    veg_densa_idx = scores_sorted[1]['original_idx']

# Solo
scores_sorted = sorted(scores, key=lambda x: x['solo'], reverse=True)
solo_idx = scores_sorted[0]['original_idx']
if solo_idx in [agua_idx, veg_densa_idx]:
    solo_idx = scores_sorted[1]['original_idx']

# Vegetação rasteira (restante)
restantes = [i for i in range(k) if i not in [agua_idx, veg_densa_idx, solo_idx]]
veg_rasteira_idx = restantes[0] if restantes else 2

# Criar mapeamento de cores
color_mapping = {
    agua_idx: 0,           # Azul
    veg_densa_idx: 1,      # Verde
    veg_rasteira_idx: 2,   # Amarelo
    solo_idx: 3            # Marrom
}

# Reordenar Labels para cores corretas
labels_map_corrigido = np.full_like(labels_map, -1)
for cls_original, cls_corrigido in color_mapping.items():
    labels_map_corrigido[labels_map == cls_original] = cls_corrigido

labels_map = labels_map_corrigido

# Atualizar labels_valid também
labels_valid_corrigido = np.full_like(labels_valid, -1)
for cls_original, cls_corrigido in color_mapping.items():
    labels_valid_corrigido[labels_valid == cls_original] = cls_corrigido
labels_valid = labels_valid_corrigido

# CALCULAR ESTATÍSTICAS DAS CLASSESS
class_stats = []
for cls in range(k):
    n_class = (labels_valid == cls).sum()
    perc_class = n_class / dados['total_valid'] * 100
    class_stats.append((cls, n_class, perc_class))

return {
    'labels_map': labels_map,
    'labels_valid': labels_valid,
    'class_stats': class_stats, # [(cls, n_pixels, %), ...]
    'total_valid': dados['total_valid']
}

def processar_mahalanobis(dados_kmeans, dados_imagem, percentil=95):
    """Calcula anomalias de Mahalanobis similar ao script anterior."""
    px_norm = dados_imagem['px_norm']
    labels_map = dados_kmeans['labels_map']
    labels_valid = dados_kmeans['labels_valid']
    H, W = dados_imagem['H'], dados_imagem['W']
    B = dados_imagem['B']

```

```

k = len(dados_kmeans['class_stats'])

# CÁLCULO DE ESTATÍSTICAS MAHALANOBIS POR CLASSE
stats = {}
for cls in range(k):
    idx = (labels_valid == cls)
    px_cls = px_norm[idx]
    N_cls = len(px_cls)

    mu = px_cls.mean(axis=0)

    if N_cls > 1:
        cov = np.cov(px_cls, rowvar=False)
        cov_reg = cov + np.eye(B) * 1e-6
        try:
            cov_inv = inv(cov_reg)
        except:
            cov_inv = np.linalg.pinv(cov_reg)
    else:
        cov_inv = np.eye(B)

    stats[cls] = (mu, cov_inv, N_cls)

# IDENTIFICAÇÃO DE ANOMALIAS
anom_map = np.zeros((H, W), int)
anom_details = []

for cls in range(k):
    if cls not in stats:
        continue

    mask = (labels_map == cls)
    mask_valid_cls = (labels_valid == cls)

    if mask_valid_cls.sum() == 0:
        continue

    px_cls = px_norm[mask_valid_cls]
    mu, cov_inv, N_cls = stats[cls]

    diff = px_cls - mu
    temp = diff @ cov_inv
    md_vals = np.sqrt(np.sum(temp * diff, axis=1))

    if len(md_vals) > 5:
        limiar = np.percentile(md_vals, percentil)
        anom_local = md_vals > limiar
        n_anom = anom_local.sum()
        perc_anom = n_anom / N_cls * 100

        anom_full = np.zeros(H * W, bool)
        idx_flat = np.where(mask.flatten())[0]
        anom_full[idx_flat] = anom_local
        anom_full = anom_full.reshape(H, W)

        anom_map = anom_map | anom_full
        anom_details.append((cls, N_cls, md_vals.mean(), md_vals.std(), limiar))

return {
    'anom_map': anom_map,
}

```

```

        'anom_details': anom_details,
        'total_anomalias': anom_map.sum()
    }

# ETAPA 5: FUNÇÃO PRINCIPAL DE PROCESSAMENTO POR IMAGEM
#
def processar_imagem_completa(caminho_tiff, caminho_png=None):
    """
    Processa uma única imagem com todo o pipeline K-Means + Mahalanobis.
    Retorna dicionário com todas as métricas.
    """
    # Carregar dados
    dados_imagem = carregar_processar_imagem(caminho_tiff, caminho_png)

    # K-Means
    dados_kmeans = processar_kmeans(dados_imagem, k=K_CLUSTERS)

    # Mahalanobis
    dados_mahal = processar_mahalanobis(dados_kmeans, dados_imagem, percentil=MAHALONOBIS_PERCENTIL)

    # Compilar métricas
    metricas = {
        'arquivo': os.path.basename(caminho_tiff),
        'dimensoes': f'{dados_imagem['H']}x{dados_imagem['W']}',
        'total_pixels': dados_imagem['total_pixels'],
        'pixels_validos': dados_imagem['total_valid'],
        'percentual_validos': (dados_imagem['total_valid'] / dados_imagem['total'])
    }

    # Estatísticas por classe
    for cls, (_, n_pixels, perc) in enumerate(dados_kmeans['class_stats']):
        metricas[f'classe_{cls}_pixels'] = n_pixels
        metricas[f'classe_{cls}_percentual'] = perc

    # Anomalias
    metricas['anomalias_totais'] = dados_mahal['total_anomalias']
    metricas['anomalias_percentual'] = (dados_mahal['total_anomalias'] / dados_imagem['total'])

    # Detalhes Mahalanobis por classe
    for detalhe in dados_mahal['anom_details']:
        cls, N_cls, md_mean, md_std, limiar, n_anom, perc_anom = detalhe
        metricas[f'classe_{cls}_md_media'] = md_mean
        metricas[f'classe_{cls}_md_desvio'] = md_std
        metricas[f'classe_{cls}_md_limiar'] = limiar
        metricas[f'classe_{cls}_anomalias'] = n_anom
        metricas[f'classe_{cls}_anomalias_%'] = perc_anom

    return metricas

# ETAPA 6: VARRER DIRETÓRIOS
#
def varrer_e_processar_pastas(base_path):
    """
    Varre a estrutura de pastas e processa todas as imagens.
    Retorna lista de dicionários com métricas.
    """
    todos_resultados = []

    # Contar total de arquivos para calcular progresso
    total_arquivos = 0

```

```

arquivos_por_estacao = {}

for ano in sorted(os.listdir(base_path)):
    ano_path = os.path.join(base_path, ano)
    if not os.path.isdir(ano_path):
        continue

    for estacao in sorted(os.listdir(ano_path)):
        estacao_path = os.path.join(ano_path, estacao)
        if not os.path.isdir(estacao_path):
            continue

        arquivos_tiff = [f for f in os.listdir(estacao_path) if f.lower().endswith('.tif')]
        total_arquivos += len(arquivos_tiff)
        arquivos_por_estacao[(ano, estacao)] = len(arquivos_tiff)

if total_arquivos == 0:
    atualizar_progresso(100, "Nenhum arquivo encontrado")
    return []

# Iniciar processamento
arquivos_processados = 0
atualizar_progresso(0, "Iniciando varredura de pastas")

# Percorrer anos
for ano_idx, ano in enumerate(sorted(os.listdir(base_path))):
    ano_path = os.path.join(base_path, ano)
    if not os.path.isdir(ano_path):
        continue

    # Percorrer estações
    for estacao_idx, estacao in enumerate(sorted(os.listdir(ano_path))):
        estacao_path = os.path.join(ano_path, estacao)
        if not os.path.isdir(estacao_path):
            continue

        # Atualizar status
        atualizar_progresso(
            (arquivos_processados / total_arquivos * 100),
            f"Processando {ano}/{estacao}"
        )

    # Listar arquivos TIFF na pasta da estação
    arquivos_tiff = [f for f in os.listdir(estacao_path) if f.lower().endswith('.tif')]

    if not arquivos_tiff:
        continue

    # Processar cada TIFF
    for tiff_idx, tiff_file in enumerate(sorted(arquivos_tiff)):
        tiff_path = os.path.join(estacao_path, tiff_file)

        # Atualizar status detalhado
        atualizar_progresso(
            (arquivos_processados / total_arquivos * 100),
            f"Processando {ano}/{estacao}: {tiff_file}"
        )

    # Encontrar PNG correspondente
    nome_base = os.path.splitext(tiff_file)[0]

```

```

        png_file = nome_base + '.png'
        png_path = os.path.join(estacao_path, png_file) if os.path.exists(png_path)

    try:
        # Processar imagem (SEM PRINT DESNECESSÁRIO)
        metricas = processar_imagem_completa(tiff_path, png_path)

        # Adicionar metadados
        metricas['ano'] = ano
        metricas['estacao'] = estacao
        metricas['caminho_tiff'] = tiff_path
        metricas['caminho_png'] = png_path if png_path else 'N/A'

        todos_resultados.append(metricas)

    except Exception as e:
        # Apenas registra erro sem parar o processamento
        print(f"    X ERRO em {tiff_file}: {str(e)}")

    arquivo_processados += 1

# Finalizar
atualizar_progresso(100, "Processamento concluído!")

return todos_resultados

# ETAPA 7: FORMATAÇÃO DA PLANILHA 1 (NOME: DETALHADO)
# -----
def formatar_planilha_detalhado(writer, df_detalhado):
    HEADER_FILL = "D9D9D9" # Cinza claro para cabeçalho
    FONT_NAME = "Calibri"
    FONT_SIZE = 12

    # Formato: (nome_coluna, largura, alinhamento, formato_numero, nome_amigavel)
    config_detalhado = [
        # Metadados
        ("ano", 6, "center", "0", "Ano"),
        ("estacao", 12, "center", "@", "Estação"),
        ("arquivo", 25, "left", "@", "Arquivo"),
        ("dimensoes", 12, "center", "@", "Dimensões"),
        ("total_pixels", 12, "right", "#,##0", "Total Pixels"),
        ("pixels_validos", 12, "right", "#,##0", "Pixels Válidos"),
        ("percentual_validos", 14, "right", "0.0%", "% Válidos"),

        # Anomalias gerais
        ("anomalias_totais", 16, "right", "#,##0", "Anomalias Totais"),
        ("anomalias_percentual", 16, "right", "0.0%", "% Anomalias"),

        # Classes - Água (0)
        ("classe_0_pixels", 12, "right", "#,##0", "Água (pixels)"),
        ("classe_0_percentual", 14, "right", "0.0%", "Água (%)"),
        ("classe_0_anomalias", 14, "right", "#,##0", "Água Anomalias"),
        ("classe_0_anomalias_%", 14, "right", "0.0%", "Água % Anomalias"),
        ("classe_0_md_media", 12, "right", "0.000", "Água MD Média"),
        ("classe_0_md_desvio", 12, "right", "0.000", "Água MD Desvio"),
        ("classe_0_md_limiar", 12, "right", "0.000", "Água MD Limiar"),

        # Classes - Vegetação Densa (1)
        ("classe_1_pixels", 12, "right", "#,##0", "Veg. Densa (pixels)"),
        ("classe_1_percentual", 14, "right", "0.0%", "Veg. Densa (%"))
    ]

```

```

        ("classe_1_anomalias", 14, "right", "#,##0", "Veg. Densa Anomalias"),
        ("classe_1_anomalias_%", 14, "right", "0.0%", "Veg. Densa % Anomalias"),
        ("classe_1_md_media", 12, "right", "0.000", "Veg. Densa MD Média"),
        ("classe_1_md_desvio", 12, "right", "0.000", "Veg. Densa MD Desvio"),
        ("classe_1_md_limiar", 12, "right", "0.000", "Veg. Densa MD Limiar"),

        # Classes - Vegetação Rasteira (2)
        ("classe_2_pixels", 12, "right", "#,##0", "Veg. Rasteira (pixels)"),
        ("classe_2_percentual", 14, "right", "0.0%", "Veg. Rasteira (%)"),
        ("classe_2_anomalias", 14, "right", "#,##0", "Veg. Rasteira Anomalias"),
        ("classe_2_anomalias_%", 14, "right", "0.0%", "Veg. Rasteira % Anomalias")
        ("classe_2_md_media", 12, "right", "0.000", "Veg. Rasteira MD Média"),
        ("classe_2_md_desvio", 12, "right", "0.000", "Veg. Rasteira MD Desvio"),
        ("classe_2_md_limiar", 12, "right", "0.000", "Veg. Rasteira MD Limiar"),

        # Classes - Solo Exposto (3)
        ("classe_3_pixels", 12, "right", "#,##0", "Solo (pixels)"),
        ("classe_3_percentual", 14, "right", "0.0%", "Solo (%)"),
        ("classe_3_anomalias", 14, "right", "#,##0", "Solo Anomalias"),
        ("classe_3_anomalias_%", 14, "right", "0.0%", "Solo % Anomalias"),
        ("classe_3_md_media", 12, "right", "0.000", "Solo MD Média"),
        ("classe_3_md_desvio", 12, "right", "0.000", "Solo MD Desvio"),
        ("classe_3_md_limiar", 12, "right", "0.000", "Solo MD Limiar"),

        # Caminhos (últimas colunas)
        ("caminho_tiff", 40, "left", "@", "Caminho TIFF"),
        ("caminho_png", 40, "left", "@", "Caminho PNG")
    ]

worksheet = writer.sheets['Detalhado']

# Configurar estilos
header_font = Font(name=FONT_NAME, size=FONT_SIZE, bold=True)
header_fill = PatternFill(start_color=HEADER_FILL, end_color=HEADER_FILL, fill_type='solid')
header_alignment = Alignment(horizontal="center", vertical="center", wrap_text=True)

cell_font = Font(name=FONT_NAME, size=FONT_SIZE)
center_alignment = Alignment(horizontal="center", vertical="center")
right_alignment = Alignment(horizontal="right", vertical="center")
left_alignment = Alignment(horizontal="left", vertical="center")

# Filtrar apenas colunas que existem no DataFrame
config_existente = [cfg for cfg in config_detalhado if cfg[0] in df_detalhado]

# Reordenar DataFrame
df_detalhado = df_detalhado[[cfg[0] for cfg in config_existente]]

# Aplicar formatação
for col_idx, (col_name, width, align, num_format, friendly_name) in enumerate(df_detalhado.columns):
    col_letter = get_column_letter(col_idx)

    # Configurar largura da coluna
    worksheet.column_dimensions[col_letter].width = width

    # Configurar célula do cabeçalho (usa nome amigável)
    cell = worksheet.cell(row=1, column=col_idx)
    cell.value = friendly_name
    cell.font = header_font
    cell.fill = header_fill
    cell.alignment = header_alignment

```

```

# Configurar alinhamento para coluna
align_obj = center_alignment if align == "center" else (right_alignment

# Aplicar a todas as células da coluna
for row_idx in range(2, len(df_detalhado) + 2):
    cell = worksheet.cell(row=row_idx, column=col_idx)
    cell.font = cell_font
    cell.alignment = align_obj

# Aplicar formato numérico
if num_format != "@":
    cell.number_format = num_format

# Congelar painel e adicionar auto-filtro
worksheet.freeze_panes = "A2"
worksheet.auto_filter.ref = worksheet.dimensions

# ETAPA 8: FORMATAÇÃO DA PLANILHA 2 (NOME: RESUMO_ESTAÇÃO)
# -----
def formatar_planilha_resumo(writer, df_resumo):
    HEADER_FILL = "D9D9D9" # Cinza claro para cabeçalho
    FONT_NAME = "Calibri"
    FONT_SIZE = 12

    # Formato: (nome_coluna, Largura, alinhamento, formato_numero, nome_amigavel
    config_resumo = [
        ("ano", 6, "center", "0", "Ano"),
        ("estacao", 12, "center", "@", "Estação"),
        ("qtd_imagens", 10, "right", "#,#0", "Qtd. Imagens"),
        ("pixels_validos_medio", 15, "right", "#,#0", "Pixels Válidos (Média)"),
        ("anomalias_%_medio", 14, "right", "0.0%", "Anomalias % (Média)"),
        ("agua_%_medio", 12, "right", "0.0%", "Água % (Média)"),
        ("veg_densa_%_medio", 15, "right", "0.0%", "Veg. Densa % (Média)"),
        ("veg_rasteira_%_medio", 17, "right", "0.0%", "Veg. Rasteira % (Média)"),
        ("solo_%_medio", 13, "right", "0.0%", "Solo % (Média)")
    ]

    worksheet = writer.sheets['Resumo_Estação']

    # Configurar estilos (mesmo do detalhado)
    header_font = Font(name=FONT_NAME, size=FONT_SIZE, bold=True)
    header_fill = PatternFill(start_color=HEADER_FILL, end_color=HEADER_FILL, fi
    header_alignment = Alignment(horizontal="center", vertical="center", wrap_te

    cell_font = Font(name=FONT_NAME, size=FONT_SIZE)
    center_alignment = Alignment(horizontal="center", vertical="center")
    right_alignment = Alignment(horizontal="right", vertical="center")
    left_alignment = Alignment(horizontal="left", vertical="center")

    # Filtrar apenas colunas que existem no DataFrame
    config_existente = [cfg for cfg in config_resumo if cfg[0] in df_resumo.colu

    # Reordenar DataFrame
    df_resumo = df_resumo[[cfg[0] for cfg in config_existente]]]

    # Aplicar formatação
    for col_idx, (col_name, width, align, num_format, friendly_name) in enumerat
        col_letter = get_column_letter(col_idx)

```

```

# Configurar largura da coluna
worksheet.column_dimensions[col_letter].width = width

# Configurar célula do cabeçalho
cell = worksheet.cell(row=1, column=col_idx)
cell.value = friendly_name
cell.font = header_font
cell.fill = header_fill
cell.alignment = header_alignment

# Configurar alinhamento
align_obj = center_alignment if align == "center" else (right_alignment

# Aplicar a todas as células da coluna
for row_idx in range(2, len(df_resumo) + 2):
    cell = worksheet.cell(row=row_idx, column=col_idx)
    cell.font = cell_font
    cell.alignment = align_obj

# Aplicar formato numérico
if num_format != "@":
    cell.number_format = num_format

# Congelar painel e adicionar auto-filtro
worksheet.freeze_panes = "A2"
worksheet.auto_filter.ref = worksheet.dimensions

# ETAPA 9: FUNÇÃO ATUALIZADA PARA EXPORTAR EXCEL (USANDO AS DUAS FUNÇÕES)
# -----
def exportar_excel(resultados, output_path="resultados_classificacao.xlsx"):
    """Exporta resultados para Excel com formatação profissional."""

    if not resultados:
        print("Nenhum resultado para exportar!")
        return

    # Criar DataFrame principal (DETALHADO)
df_principal = pd.DataFrame(resultados)

    # Converter percentuais para formato Excel (0.05 para 5%)
percent_cols = [col for col in df_principal.columns if 'percentual' in col]
for col in percent_cols:
    if col in df_principal.columns:
        df_principal[col] = df_principal[col] / 100

    # Criar DataFrame resumido (RESUMO_ESTACAO)
df_resumo = df_principal.groupby(['ano', 'estacao']).agg({
    'arquivo': 'count',
    'pixels_validos': 'mean',
    'anomalias_percentual': 'mean',
    'classe_0_percentual': 'mean',
    'classe_1_percentual': 'mean',
    'classe_2_percentual': 'mean',
    'classe_3_percentual': 'mean'
}).round(4)

    df_resumo = df_resumo.rename(columns={
        'arquivo': 'qtd_imagens',
        'pixels_validos': 'pixels_validos_medio',
        'anomalias_percentual': 'anomalias_%_medio',
    })

```

```

'classe_0_percentual': 'agua_%_medio',
'classe_1_percentual': 'veg_densa_%_medio',
'classe_2_percentual': 'veg_rasteira_%_medio',
'classe_3_percentual': 'solo_%_medio'
}).reset_index()

# Exportar para Excel
with pd.ExcelWriter(output_path, engine='openpyxl') as writer:
    # Salvar DataFrames (sem formatação ainda)
    df_principal.to_excel(writer, sheet_name='Detalhado', index=False)
    df_resumo.to_excel(writer, sheet_name='Resumo_Estação', index=False)

    # Aplicar formatação personalizada a cada planilha
    formatar_planilha_detalhado(writer, df_principal)
    formatar_planilha_resumo(writer, df_resumo)

# ETAPA 10: EXECUÇÃO PRINCIPAL
# -----
# Funções auxiliares para formatação no console
def formatar_numero_br(num):
    """Formata número com separador de milhar como PONTO e decimal como VÍRGULA"""
    formato_us = f"{num:, .0f}"
    return formato_us.replace(",", "X").replace(".", ",").replace("X", ".") 

def formatar_percentual_br(num):
    """Formata percentual com VÍRGULA decimal"""
    return f"{num:.1f}".replace(".", ",")

def criar_barra_visual(percentual, largura=50):
    """Cria uma barra visual ASCII para representar percentuais"""
    blocos_preenchidos = int(percentual * largura / 100)
    return "█" * blocos_preenchidos + "█" * (largura - blocos_preenchidos)

# Bloco principal de execução do script
if __name__ == "__main__":
    # Processar todas as pastas de imagens
    resultados = varrer_e_processar_pastas(BASE_PATH) # Executa a varredura de

    if resultados:
        # Exportar resultados para Excel
        nome_excel = f"resultados_classico_{datetime.now().strftime('%Y%m%d_%H%M')}"
        exportar_excel(resultados, nome_excel) # Exporta dados para arquivo Excel

        # Criar DataFrame para análise estatística
        df = pd.DataFrame(resultados) # Converte lista de dicionários para DataFrame

        # Exibir estatísticas gerais no console
        # Calcular totais gerais
        total_imagens = len(df)
        total_pixels = df['total_pixels'].sum()
        total_anomalias = df['anomalias_totais'].sum()
        percentual_anomalias = (total_anomalias / df['pixels_validos'].sum() * 100)

        print(f"\n{'-'*70}")
        print(" ESTATÍSTICAS GERAIS:")
        print(f"{'-'*70}")
        print(f"  • Imagens processadas: {formatar_numero_br(total_imagens)}")
        print(f"  • Pixels totais: {formatar_numero_br(total_pixels)}")
        print(f"  • Anomalias detectadas: {formatar_numero_br(total_anomalias)}")

```

```

print(f"    • Percentual de anomalias: {formatar_percentual_br(percentual)

# Exibir distribuição média das classes
print(f"\n{'-'*70}")
print(" DISTRIBUIÇÃO MÉDIA DAS CLASSES (Dataset completo):")
print(f"{'-'*70}")

# Configurações para exibição da tabela
area_total_km2 = 10000 # Área total hipotética de 10.000 km²
emojis = ["🔵", "🟢", "🟡", "🟠"] # Emojis para representação visual

print(f"{'Classe':<22} {'Cor':<4} {'Área (km²)':<12} {'%':<8} {'Barra'}"
print("-" * 70)

total_area = 0
total_percent = 0

# Calcular e exibir dados para cada classe
for cls in range(4):
    nome_classe = CLASS_NAMES[cls]
    media_percentual = df[f'classe_{cls}_percentual'].mean()
    area_km2 = area_total_km2 * (media_percentual / 100)

    total_area += area_km2
    total_percent += media_percentual

    # Criar barra visual proporcional ao percentual
    barra = "█" * int(media_percentual / 2) # Cada 2% = um bloco

    print(f"{nome_classe:<22} {emojis[cls]:<4} {area_km2:<11.0f} "
          f"{media_percentual:<7.1f}% {barra}")

# Linha de total consolidado
print("-" * 70)
print(f"{'TOTAL':<22} {'':<4} {total_area:<11.0f} {total_percent:<7.1f}%")

# GRÁFICO DE LINHA TEMPORAL (COM MATPLOTLIB)
# FUNÇÃO ISOLADA para criar gráfico sem poluição
def criar_grafico_temporal(df):
    """Cria gráfico temporal SEM imprimir objetos no console"""
    try:
        import matplotlib.pyplot as plt
        import numpy as np

        # Preparar dados
        anos = sorted(df['ano'].unique())

        dados_grafico = {
            'Água': [df[df['ano'] == ano]['classe_0_percentual'].mean(),
                     'Vegetação Densa': [df[df['ano'] == ano]['classe_1_percentual'].mean(),
                     'Vegetação Rasteira': [df[df['ano'] == ano]['classe_2_percentual'].mean(),
                     'Solo Exposto': [df[df['ano'] == ano]['classe_3_percentual'].mean()]
        }

        cores = ['#1f77b4', '#2ca02c', '#ff7f0e', '#8c564b']

        # CRIAR FIGURA - COM SILENCIO TOTAL
        fig, ax = plt.subplots(figsize=(14, 7))

        # Plotar Linhas (SEM RETORNAR VALOR)
    
```

```

        lines = [] # Lista para armazenar objetos de linha
        for idx, (classe, valores) in enumerate(dados_grafico.items()):
            line = ax.plot(anos, valores,
                            marker='o',
                            linewidth=3.0,
                            markersize=10,
                            color=cores[idx],
                            label=classe)
            lines.append(line[0]) # Armazena, não imprime

        # Configurar (SEM RETORNAR VALOR)
        ax.set_title('EVOLUÇÃO TEMPORAL DA COBERTURA DO SOLO',
                     fontsize=18, fontweight='bold', pad=25)
        ax.set_xlabel('Ano', fontsize=14)
        ax.set_ylabel('Porcentagem (%)', fontsize=14)
        ax.grid(True, linestyle='--', alpha=0.3)

        # Legenda (SEM RETORNAR VALOR)
        legend = ax.legend(loc='upper center',
                            bbox_to_anchor=(0.5, -0.15),
                            ncol=4,
                            fontsize=12,
                            framealpha=0.9)

        ax.set_xticks(anos)
        ax.set_yticks([0, 50])
        ax.tick_params(axis='both', labelsize=12)

        # Anotações (SILENCIOSAS)
        annotations = [] # Lista para armazenar anotações
        for idx, (classe, valores) in enumerate(dados_grafico.items()):
            for x, y in zip(anos, valores):
                ann = ax.annotate(f'{y:.1f}%', 
                                  (x, y),
                                  textcoords="offset points",
                                  xytext=(0, 12),
                                  ha='center',
                                  fontsize=11,
                                  fontweight='bold',
                                  color=cores[idx],
                                  bbox=dict(boxstyle="round", pad=0.3, facecolor='white'))
                annotations.append(ann) # Armazena, não imprime

        plt.tight_layout(rect=[0, 0.1, 1, 0.95])

        # RETORNAR APENAS A FIGURA - NADA MAIS
        return fig

    except ImportError:
        return None

    # EXECUTAR E EXIBIR GRÁFICO
    try:
        # Criar gráfico de forma isolada
        figura = criar_grafico_temporal(df)

        if figura is not None:
            # Exibir gráfico DIRETAMENTE, sem objetos intermediários
            import matplotlib.pyplot as plt
            plt.show()

```

```

        plt.close(figura) # Fechar para liberar memória
    else:
        # Se matplotlib não estiver disponível, mostrar ASCII
        raise ImportError("Matplotlib não disponível")

    except ImportError as e:
        # FALLBACK ASCII (igual ao anterior)
        print(f"⚠️ Matplotlib não disponível. Exibindo versão ASCII:")
        print(f"\n📊 TENDÊNCIA TEMPORAL (ASCII - ANOS: {', '.join(map(str, :
        print("-" * 80)

        anos = sorted(df['ano'].unique()))

        print(f"\n🌐 ÁGUA:")
        for ano in anos:
            media = df[df['ano'] == ano]['classe_0_percentual'].mean()
            barra = "█" * int(media)
            print(f"  {ano}: {media:.1f}% {barra}")

        print(f"\n🟢 VEGETAÇÃO DENSA:")
        for ano in anos:
            media = df[df['ano'] == ano]['classe_1_percentual'].mean()
            barra = "█" * int(media / 2)
            print(f"  {ano}: {media:.1f}% {barra}")

        print(f"\n🟡 VEGETAÇÃO RASTEIRA:")
        for ano in anos:
            media = df[df['ano'] == ano]['classe_2_percentual'].mean()
            barra = "█" * int(media / 2)
            print(f"  {ano}: {media:.1f}% {barra}")

        print(f"\n🟤 SOLO EXPOSTO:")
        for ano in anos:
            media = df[df['ano'] == ano]['classe_3_percentual'].mean()
            barra = "█" * int(media * 2)
            print(f"  {ano}: {media:.1f}% {barra}")

        print(f"\n📌 Legenda: Cada '█' representa ~1-2% (varia por classe)")

    # INSIGHTS E RECOMENDAÇÕES
    print(f"\n" + "-"*110)
    print(" INSIGHTS E RECOMENDAÇÕES")
    print("-"*110)

    # Consistência temporal das imagens processadas
    print(f"\n CONSISTÊNCIA TEMPORAL:")
    for ano, grupo_ano in df.groupby('ano'):
        print(f"  • {ano}: {len(grupo_ano)} imagens processadas")

    # Identificar estações com anomalias excessivas
    media_geral_anomalias = df['anomalias_percentual'].mean()
    estacoes_problematicas = []

    for (ano, estacao), grupo in df.groupby(['ano', 'estacao']):
        if grupo['anomalias_percentual'].mean() > (media_geral_anomalias * 1
            estacoes_problematicas.append((ano, estacao, grupo['anomalias_pe

    # Recomendações para auditoria
    print(f"\n RECOMENDAÇÕES PARA AUDITORIA:")
    print(f"  1. Verificar imagens com anomalias > {media_geral_anomalias * 1

```

```

if estacoes_problematicas:
    print(f" 2. Investigar estações com anomalias elevadas:")
    for ano, estacao, media in estacoes_problematicas[:3]: # Mostra até
        print(f"      • {ano}/{estacao}: {media:.1f}%")
else:
    print(f" 2. Nenhuma estação com anomalias excessivas detectada")

print(f" 3. Analisar variações sazonais na distribuição das classes")
print(f" 4. Validar manualmente {max(1, int(total_imagens * 0.05))} im"

# Insights ambientais baseados nas médias
print(f"\n INSIGHTS AMBIENTAIS:")
media_veg_densa = df['classe_1_percentual'].mean()
media_solo = df['classe_3_percentual'].mean()
media_agua = df['classe_0_percentual'].mean()

if media_veg_densa > 35:
    print(f"    Vegetação densa adequada: {media_veg_densa:.1f}%")
else:
    print(f"    ⚠️ Vegetação densa abaixo do ideal: {media_veg_densa:.1f}%")

if media_solo < 20:
    print(f"    Solo exposto dentro do esperado: {media_solo:.1f}%")
else:
    print(f"    ⚠️ Solo exposto acima do esperado: {media_solo:.1f}%")

if media_agua > 5:
    print(f"    Presença hídrica adequada: {media_agua:.1f}%")
else:
    print(f"    ⚠️ Presença hídrica abaixo do esperado: {media_agua:.1f}%")

# Informar Localização do arquivo de resultados
print(f" 📁 Resultados salvos em: {nome_excel}")

# ETAPA 11: FINALIZAÇÃO
# -----
tempo_total = time.time() - inicio # tempo total em seg

# Converte para o formato HH:MM:SS
horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

print("\n" + "="*110)
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=*110")
=====
```

PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS

Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais
Calculo K-Means (k=4) + Distância de Mahalanobis no Dataset

VBox(children=(HTML(value='Iniciando processamento do dataset de imagens...'), FloatProgress(value=0.0,...

Runtimewarning: Mean of empty slice.
 Runtimewarning: invalid value encountered in divide

ESTATÍSTICAS GERAIS:

- Imagens processadas: 139
- Pixels totais: 22.240.000
- Anomalias detectadas: 1.109.804
- Percentual de anomalias: 5,0%

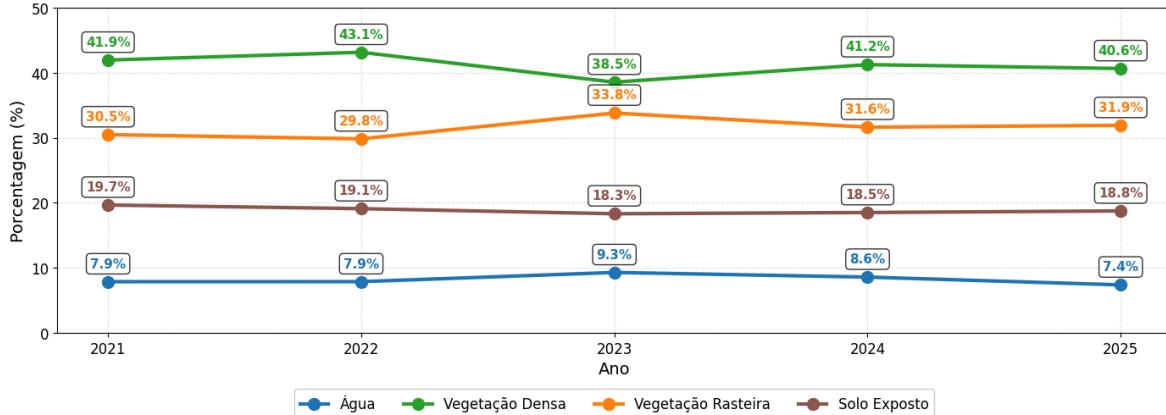
DISTRIBUIÇÃO MÉDIA DAS CLASSES (Dataset completo):

Classe	Cor	Área (km ²)	%	Barra
--------	-----	-------------------------	---	-------

Água	●	821	8.2	%
Vegetação Densa	●	4125	41.3	%
Vegetação Rasteira	●	3142	31.4	%
Solo Exposto	●	1889	18.9	%

TOTAL	9978	99.8	%
-------	------	------	---

EVOLUÇÃO TEMPORAL DA COBERTURA DO SOLO



 INSIGHTS E RECOMENDAÇÕES

CONSISTÊNCIA TEMPORAL:

- 2021: 27 imagens processadas
- 2022: 32 imagens processadas
- 2023: 24 imagens processadas
- 2024: 32 imagens processadas
- 2025: 24 imagens processadas

RECOMENDAÇÕES PARA AUDITORIA:

1. Verificar imagens com anomalias > 6.0%
2. Nenhuma estação com anomalias excessivas detectada
3. Analisar variações sazonais na distribuição das classes
4. Validar manualmente 6 imagens aleatórias

INSIGHTS AMBIENTAIS:

Vegetação densa adequada: 41.3%
 Solo exposto dentro do esperado: 18.9%
 Presença hídrica adequada: 8.2%

Resultados salvos em: resultados_classico_20251206_194314.xlsx

Tempo de execução → 00:01:52.85 (hh:mm:ss)
 Finalizado em → 06/12/2025 às 19:43:14

```
In [21]: # -----
# BLOCO 2: EXECUÇÃO - LETRA C
#           Aprimoramento de Rede Neural com Yolo 8
#
# OBJETIVO: Gerar anotações pseudo-labels e realizar fine-tuning no modelo Yolo
# -----
# Importações Complementares
import yaml      # Biblioteca para ler e criar arquivos de configuração .yaml (e
import shutil    # Biblioteca para operações de alto nível em arquivos (copiar,
import random   # Biblioteca para gerar números aleatórios (usada para embaralhar
# ETAPA 1: CABEÇALHO INSTITUCIONAL
# -----
print("=" * 110)                                     # Imprime Linha
print("PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS") # Título principal
print("          Asseguração Independente de Relatórios ESG via Dados Exógenos Sa") # Especificação
print("          Aprendizado Profundo Fine-Tuning Yolo 8")      # Especificação
print("=" * 110)                                     # Imprime Linha

inicio = time.time()                                # Registra o tempo

# ETAPA 2: BARRA DE PROGRESSO
# -----
try:
    import ipywidgets as widgets # Importa biblioteca para widgets interativos
    progress = widgets.FloatProgress( # Cria widget de progresso flutuante
        value=0, min=0, max=100, description='Progresso:', # Configurações iniciais
        style={'bar_color': '#0066cc'}, layout=widgets.Layout(width='85%', height='25px')
)
```

```

        )
status = widgets.HTML(value="Iniciando configurações...") # Widget h
display(widgets.VBox([status, progress])) # Exibe caixa vertical com status
use_widgets = True # Flag para indicar que widgets estão disponíveis
except ImportError:
    use_widgets = False # Flag para indicar falha na importação (ex: ambiente s
    print("Widgets não instalados.") # Mensagem informativa se widgets não esti

# ETAPA 3: CONFIGURAÇÕES E PARÂMETROS
# -----
# Atualização da Barra de Progresso
if use_widgets:
    status.value = "Definindo caminhos e parâmetros do sistema..."
    progress.value = 10

# Definição de Caminhos do Sistema de Arquivos
BASE_DIR = Path(r"C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c")
IMG_ORIGINAL = Path(r"C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\202
MASK_KMEANS = Path(r"C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025

# Configuração de Diretórios de Saída
ID_EXECUCAO = f"Yolo_treino"
DATASET_DIR = BASE_DIR / f"Yolo_dataset_{ID_EXECUCAO}"
OUTPUT_DIR = BASE_DIR / "Yolo_resultados"

# Definições Semânticas (Classes e Cores)
CLASS_NAMES = {0: 'Agua', 1: 'Veg_Densa', 2: 'Veg_Rasteira', 3: 'Solo'}
COLORS_BGR = {0: (171, 71, 0), 1: (69, 139, 35), 2: (0, 215, 255), 3: (0, 75, 15

# ETAPA 4: PREPARAÇÃO DO DATASET (EXTRAÇÃO DE POLÍGONOS E SPLIT)
# -----
def preparar_dados():
    # Atualização da Barra de Progresso
    if use_widgets:
        status.value = "Processando imagens e gerando anotações (Polígonos).."
        progress.value = 20

    # Criação da estrutura de pastas para o YOLO
    for split in ['train', 'val']:
        os.makedirs(DATASET_DIR / split / "images", exist_ok=True)
        os.makedirs(DATASET_DIR / split / "labels", exist_ok=True)

    # Leitura das Imagens Base
    img = cv2.imread(str(IMG_ORIGINAL))
    mask = cv2.imread(str(MASK_KMEANS))
    H, W, _ = img.shape
    todos_poligonos = []

    # Loop de Extração de Polígonos por Classe
    for cls_id, color_bgr in COLORS_BGR.items():
        # Definição de Limites de cor com tolerância (para lidar com compressão)
        lower = np.array([max(0, c-15) for c in color_bgr])
        upper = np.array([min(255, c+15) for c in color_bgr])
        binary_mask = cv2.inRange(mask, lower, upper)

        # Encontra contornos na máscara binária
        contours, _ = cv2.findContours(binary_mask, cv2.RETR_EXTERNAL, cv2.CHAIN

        conta_classe = 0
        for cnt in contours:

```

```

# Filtragem de Ruído e Simplificação
if cv2.contourArea(cnt) > 50:
    cnt = cv2.approxPolyDP(cnt, 0.005 * cv2.arcLength(cnt, True), True)

# Normalização das Coordenadas (0 a 1) exigida pelo YOLO
points = [f"{pt[0][0]}/{W:.6f} {pt[0][1]}/{H:.6f}" for pt in cnt]

if len(points) > 2:
    todos_poligonos.append(f"{cls_id} " + ".join(points))
    conta_classe += 1

print(f"    -> Classe {CLASS_NAMES[cls_id]}: {conta_classe} áreas encontradas")

# Divisão de Dados (Train/Val Split)
random.seed(42)
random.shuffle(todos_poligonos)
split_idx = int(len(todos_poligonos) * 0.8)
train_labels = todos_poligonos[:split_idx]
val_labels = todos_poligonos[split_idx:]

# Salvamento dos Arquivos
nome_txt = IMG_ORIGINAL.stem + ".txt"

# Conjunto de Treino
shutil.copy2(IMG_ORIGINAL, DATASET_DIR / "train" / "images" / IMG_ORIGINAL.name)
with open(DATASET_DIR / "train" / "labels" / nome_txt, "w") as f:
    f.write("\n".join(train_labels))

# Conjunto de Validação
shutil.copy2(IMG_ORIGINAL, DATASET_DIR / "val" / "images" / IMG_ORIGINAL.name)
with open(DATASET_DIR / "val" / "labels" / nome_txt, "w") as f:
    f.write("\n".join(val_labels))

# Criação do Arquivo de Configuração (data.yaml)
yaml_data = {
    'path': str(DATASET_DIR),
    'train': 'train/images',
    'val': 'val/images',
    'names': CLASS_NAMES
}

with open(DATASET_DIR / "data.yaml", "w") as f:
    yaml.dump(yaml_data, f)

print(f">> Dataset Pronto! {len(train_labels)} polígonos de treino / {len(val_labels)} de validação")

# ETAPA 5: MONITOR GRÁFICO (COM ATUALIZAÇÃO DA BARRA DE PROGRESSO)
# -----
from IPython.display import display, clear_output

class LivePlotter:
    def __init__(self):
        self.epochs = []
        self.losses = []
        self.maps = []

    # Inicialização da Figura Gráfica
    self.fig, (self.ax1, self.ax2) = plt.subplots(1, 2, figsize=(12, 5))
    plt.close(self.fig)

```

```

# Cria um "Handle" (Identificador) para atualizar o gráfico dinamicamente
self.display_handle = display(self.fig, display_id=True)

# Impressão do Cabeçalho da Tabela no Console
print("=*95")
print(f'{ "TREINAMENTO INTENSIVO (MONITORAMENTO EM TEMPO REAL)":^95}')
print("=*95")
print(f'| {'EPOCA':^7} | {'SEG LOSS':^15} | {'ACURÁCIA (mAP)':^18} | {'P')
print("-" * 95)

def update(self, trainer):
    # --- ATUALIZAÇÃO DA BARRA DE PROGRESSO (WIDGETS) ---
    if use_widgets:
        total = trainer.args.epochs
        atual = trainer.epoch + 1
        progress.value = (atual / total) * 100
        status.value = f"<b>Treinando Rede Neural... Época {atual}/{total}</b>"

    # Coleta de Métricas do Treinador YOLO
    epoch = trainer.epoch + 1
    metrics = trainer.metrics

    # Extração Segura dos Valores (Loss, mAP, Precision, Recall)
    loss = trainer.loss_items[1].item() if len(trainer.loss_items) > 1 else None
    map50 = metrics.get("metrics/mAP50(M)", metrics.get("metrics/mAP50(m)", None))
    prec = metrics.get("metrics/Precision(M)", metrics.get("metrics/Precision(m)", None))
    rec = metrics.get("metrics/Recall(M)", metrics.get("metrics/Recall(m)", None))

    # Atualização dos Históricos para o Gráfico
    self.epochs.append(epoch)
    self.losses.append(loss)
    self.maps.append(map50)

    # Redesenho dos Gráficos
    self.ax1.clear()
    self.ax2.clear()

    # Gráfico 1: Curva de Erro (Vermelho)
    self.ax1.plot(self.epochs, self.losses, 'r-o', linewidth=2)
    self.ax1.set_title("ERRO (Loss) - Caindo?", color='red', fontweight='bold')
    self.ax1.set_xlabel("Época")
    self.ax1.grid(True, alpha=0.3)

    # Gráfico 2: Curva de Acurácia (Azul)
    self.ax2.plot(self.epochs, self.maps, 'b-o', linewidth=2)
    self.ax2.set_title("ACURÁCIA (mAP) - Subindo?", color='blue', fontweight='bold')
    self.ax2.set_xlabel("Época")
    self.ax2.grid(True, alpha=0.3)

    # Atualiza a imagem na tela sem piscar (usando o Handle criado no init)
    self.display_handle.update(self.fig)

    # Impressão da Linha de Dados na Tabela do Console
    seta = "..."
    if len(self.maps) > 1:
        if map50 > self.maps[-2]: seta = "Sobe ↑"
        elif loss < self.losses[-2]: seta = "Cai ↓"

    print(f'| {epoch:^7} | {loss:^15.4f} | {map50:^18.2%} | {prec:^12.2%} | {seta}' )

```

```

# Instância Global e Funções de Callback (Ganchos para o YOLO)
plotter = None

def on_train_epoch_end(trainer):
    if plotter: plotter.update(trainer)

def on_train_start(trainer):
    global plotter
    plotter = LivePlotter()

# ETAPA 6: EXECUÇÃO DO TREINAMENTO
# -----
def executar_pipeline():
    # 1. Preparação
    if use_widgets:
        status.value = "<b>1/3: Organizando Dataset e Polígonos...</b>"

    preparar_dados()

    # 2. Configuração do Modelo
    if use_widgets:
        status.value = "<b>2/3: Carregando YOLOv8-Seg e Inicializando Gráficos.."

    print(f"\n--- 2. INICIANDO MOTOR (YOLOv8-SEG) ---")
    model = YOLO('yolov8n-seg.pt')

    # Conecta o nosso Monitor Gráfico (ETAPA 5) ao Modelo
    model.add_callback("on_train_start", on_train_start)
    model.add_callback("on_train_epoch_end", on_train_epoch_end)

    # 3. Treinamento (A Validação acontece aqui)
    if use_widgets:
        status.value = "<b>3/3: TREINANDO... (Acompanhe o Gráfico abaixo)</b>"

    model.train(
        data=str(DATASET_DIR / "data.yaml"),
        epochs=100,
        imgsz=640,
        batch=4,
        patience=0,
        project=str(OUTPUT_DIR),
        name=ID_EXECUCAO,
        device='cpu',
        plots=True,
        verbose=False,
        val=True
    )

    # 4. Conclusão
    if use_widgets:
        status.value = "<b>✓ PROCESSO CONCLUÍDO! Verifique os resultados.</b>"
        progress.value = 100
        progress.style.bar_color = '#28a745'

    print("-" * 95)
    print("\n✓ FIM! O gráfico deve mostrar a linha de erro no chão.")
    print(f"📁 Resultados salvos em: {OUTPUT_DIR / ID_EXECUCAO}")

    plt.ioff()
    plt.show()

```

```
# Comando para rodar tudo se for o script principal
if __name__ == "__main__":
    executar_pipeline()

# ETAPA 7: FINALIZAÇÃO
# -----
tempo_total = time.time() - inicio # tempo total em seg

# Converte para o formato HH:MM:SS
horas = int(tempo_total // 3600)
minutos = int((tempo_total % 3600) // 60)
segundos = tempo_total % 60
tempo_formatado = f"{horas:02d}:{minutos:02d}:{segundos:05.2f}"

print("\n" + "=" * 110)
print(f"Tempo de execução → {tempo_formatado} (hh:mm:ss)")
print(f"Finalizado em → {datetime.now().strftime('%d/%m/%Y às %H:%M:%S')}")
print("=" * 110)
```

```
=====
=====
PROJETO: AUDITORIA AMBIENTAL EM USINAS HIDRELÉTRICAS
    Asseguração Independente de Relatórios ESG via Dados Exógenos Satelitais
        Aprendizado Profundo Fine-Tuning Yolo 8
=====
```

```
=====
=====
VBox(children=(HTML(value='<b>Iniciando configurações...</b>'), FloatProgress(value=0.0, description='Progress...'))
```

```

-> Classe Agua: 3 áreas encontradas.
-> Classe Veg_Densa: 39 áreas encontradas.
-> Classe Veg_Rasteira: 32 áreas encontradas.
-> Classe Solo: 55 áreas encontradas.
>> Dataset Pronto! 103 polígonos de treino / 26 de validação.

```

--- 2. INICIANDO MOTOR (YOLOv8-SEG) ---

Ultralytics 8.3.235 Python-3.13.7 torch-2.9.1+cpu CPU (AMD Ryzen 5 5625U with Radeon Graphics)

```

engine\trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=4, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, compile=False, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Viso Computacional\VC_TF\Yolo_dataset_Yolo_treino\data.yaml, degrees=0.0, deterministic=True, device=cpu, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=100, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=640, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=yolov8n-seg.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=Yolo_treino3, nbs=64, nms=False, opset=None, optimize=False, optimizer=auto, overlap_mask=True, patience=0, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Viso Computacional\VC_TF\Yolo_resultados, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Viso Computacional\VC_TF\Yolo_resultados\Yolo_treino3, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=segment, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=False, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None
Overriding model.yaml nc=80 with nc=4

```

	from	n	params	module
arguments				
0		-1	1	464 ultralytics.nn.modules.conv.Conv
[3, 16, 3, 2]		-1	1	4672 ultralytics.nn.modules.conv.Conv
1		-1	1	7360 ultralytics.nn.modules.block.C2f
[16, 32, 3, 2]		-1	1	18560 ultralytics.nn.modules.conv.Conv
2		-1	2	49664 ultralytics.nn.modules.block.C2f
[32, 32, 1, True]		-1	1	73984 ultralytics.nn.modules.conv.Conv
3		-1	2	197632 ultralytics.nn.modules.block.C2f
[32, 64, 3, 2]		-1	1	295424 ultralytics.nn.modules.conv.Conv
4		-1	2	460288 ultralytics.nn.modules.block.C2f
[64, 64, 2, True]		-1	1	164608 ultralytics.nn.modules.block.SPPF
5		-1	1	0 torch.nn.modules.upsampling.Upsample
[64, 128, 3, 2]		-1	2	[None, 2, 'nearest']
6		-1	1	0 ultralytics.nn.modules.conv.Concat
[128, 128, 2, True]		-1	1	
7		-1	1	
[128, 256, 3, 2]		-1	1	
8		-1	1	
[256, 256, 1, True]		-1	1	
9		-1	1	
[256, 256, 5]		-1	1	
10		-1	1	
[None, 2, 'nearest']		-1	6	
11		-1	1	

```
[1]
12           -1  1    148224 ultralytics.nn.modules.block.C2f
[384, 128, 1]
13           -1  1      0 torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
14      [-1, 4]  1      0 ultralytics.nn.modules.conv.Concat
[1]
15           -1  1    37248 ultralytics.nn.modules.block.C2f
[192, 64, 1]
16           -1  1    36992 ultralytics.nn.modules.conv.Conv
[64, 64, 3, 2]
17      [-1, 12]  1      0 ultralytics.nn.modules.conv.Concat
[1]
18           -1  1   123648 ultralytics.nn.modules.block.C2f
[192, 128, 1]
19           -1  1   147712 ultralytics.nn.modules.conv.Conv
[128, 128, 3, 2]
20      [-1, 9]  1      0 ultralytics.nn.modules.conv.Concat
[1]
21           -1  1   493056 ultralytics.nn.modules.block.C2f
[384, 256, 1]
22      [15, 18, 21]  1  1004860 ultralytics.nn.modules.head.Segment
[4, 32, 64, [64, 128, 256]]
YOLOv8n-seg summary: 151 layers, 3,264,396 parameters, 3,264,380 gradients, 11.5
GFLOPs
```

Transferred 381/417 items from pretrained weights
Freezing layer 'model.22.dfl.conv.weight'
train: Fast image access (ping: 0.00.0 ms, read: 35.10.0 MB/s, size: 379.6 KB)
train: Scanning C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Visão Computacional\VC_TF\Yolo_dataset_Yolo_treino\train\labels.cache... 1 images, 0 backgrounds, 0 corrupt: 100% ————— 1/1 1.4Kit/s 0.0s0s
val: Fast image access (ping: 0.00.0 ms, read: 23.30.0 MB/s, size: 379.6 KB)
val: Scanning C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Visão Computacional\VC_TF\Yolo_dataset_Yolo_treino\val\labels.cache... 1 images, 0 backgrounds, 0 corrupt: 100% ————— 1/1 1.1Kit/s 0.0s0s
Plotting labels to C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Visão Computacional\VC_TF\Yolo_resultados\Yolo_treino3\labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.00125, momentum=0.9) with parameter groups 66 weight(decay=0.0), 77 weight(decay=0.0005), 76 bias(decay=0.0)
<Figure size 1200x500 with 2 Axes>

TREINAMENTO INTENSIVO (MONITORAMENTO EM TEMPO REAL)							
ÉPOCA STATUS	SEG LOSS	ACURÁCIA (mAP)	PRECISÃO	RECALL			
<hr/>							
Image sizes 640 train, 640 val							
Using 0 dataloader workers							
Logging results to C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Visão Computacional\VC_TF\Yolo_resultados\Yolo_treino3							
Starting training for 100 epochs...							
<hr/>							
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
640:	1/100	0G	3.007	5.864	3.906	2.199	183
	1	5.8639	0.00%	0.00%	0.00%	0.00%	
	...						
P50-95) s 0.4s	Class	Images	Instances	Box(P	R	mAP50	mA
	Mask(P	R	mAP50	mAP50-95): 100%	—	1/1	2.8it/
0.00369	all	1	26	0.0105	0.139	0.014	
0.00358	0.0278	0.00208	0.000208				
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
640:	2/100	0G	2.54	5.881	3.833	2.159	119
	2	5.8807	0.21%	0.00%	0.00%	0.00%	
Sobe ↑							
P50-95) s 0.4s	Class	Images	Instances	Box(P	R	mAP50	mA
	Mask(P	R	mAP50	mAP50-95): 100%	—	1/1	2.6it/
0.0037	all	1	26	0.0105	0.139	0.014	
0.00355	0.0278	0.00206	0.000412				
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
640:	3/100	0G	2.953	5.884	3.893	2.23	264
	3	5.8843	0.21%	0.00%	0.00%	0.00%	
	...						
P50-95) s 0.3s	Class	Images	Instances	Box(P	R	mAP50	mA
	Mask(P	R	mAP50	mAP50-95): 100%	—	1/1	2.9it/
0.00328	all	1	26	0.0105	0.139	0.0133	
0.00355	0.0278	0.00214	0.000429				
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
640:	4/100	0G	3.018	5.694	3.954	2.191	271
	4	5.6935	0.21%	0.00%	0.00%	0.00%	
Sobe ↑							

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.0027	all 0.00358	1 0.0278	26 0.0022	0.0106 0.00022	0.139	0.0114	
<hr/>							
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
640: 100%	5/100	0G	2.557	5.694	3.849	2.134	188
Sobe	5	5.6937	1/1 1.4s/it	1.4s 0.22%	0.00%	0.00%	0.00%
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.0027	all 0.00358	1 0.0278	26 0.0022	0.0106 0.00022	0.139	0.0114	
<hr/>							
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
640: 100%	6/100	0G	2.652	5.708	3.913	2.082	212
Sobe	6	5.7080	1/1 1.5s/it	1.5s 0.22%	0.00%	0.00%	0.00%
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00344	all 0.0037	1 0.0278	26 0.00224	0.0109 0.000856	0.139	0.0123	
<hr/>							
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
640: 100%	7/100	0G	2.438	5.482	3.786	2.15	93
Sobe	7	5.4824	1/1 1.0it/s	1.0s 0.22%	0.00%	0.00%	0.00%
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00344	all 0.0037	1 0.0278	26 0.00224	0.0109 0.000856	0.139	0.0123	
<hr/>							
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
640: 100%	8/100	0G	2.479	5.46	3.87	2.141	83
Cai	8	5.4598	1/1 1.0it/s	1.0s 0.22%	0.00%	0.00%	0.00%
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00552	all 0.00355	1 0.0278	26 0.00228	0.0105 0.00113	0.139	0.015	
<hr/>							
Size	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
640: 100%	9/100	0G	2.347	5.658	3.767	2.09	84
Sobe	9	5.6575	1/1 1.1it/s	0.9s 0.23%	0.00%	0.00%	0.00%

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50 95%): 100%	R	mAP50 mA
			mAP50	mAP50-95:			
0.00552	all	1	26	0.0105	0.139	0.015	
<hr/>							
640: 100%	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
10/100	0G	2.315	5.701	3.814	2.06	89	
10	5.7006	1/1 1.0s/it 1.0s	0.23%	0.00%	0.00%		
...							
P50-95) s 0.4s	Class Mask(P)	Images R	Instances	Box(P mAP50 95%): 100%	R	mAP50 mA	
0.00922	all	1	26	0.0107	0.139	0.0248	
	0.00366	0.0278	0.0111	0.00778			
<hr/>							
640: 100%	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
11/100	0G	2.5	5.188	3.754	2.177	69	
11	5.1880	1/1 1.1it/s 0.9s	1.11%	0.00%	0.00%		
Sobe							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50 95%): 100%	R	mAP50 mA	
0.00922	all	1	26	0.0107	0.139	0.0248	
	0.00366	0.0278	0.0111	0.00778			
<hr/>							
640: 100%	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
12/100	0G	2.532	5.245	3.827	2.063	177	
12	5.2446	1/1 1.4s/it 1.4s	1.11%	0.00%	0.00%		
...							
P50-95) s 0.4s	Class Mask(P)	Images R	Instances	Box(P mAP50 95%): 100%	R	mAP50 mA	
0.00922	all	1	26	0.0107	0.139	0.0248	
	0.00366	0.0278	0.0111	0.00778			
<hr/>							
640: 100%	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
13/100	0G	2.527	5.27	3.866	2.003	181	
13	5.2697	1/1 1.5s/it 1.5s	1.11%	0.00%	0.00%		
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50 95%): 100%	R	mAP50 mA	
0.0227	all	1	26	0.0107	0.139	0.047	
	0.00355	0.0278	0.03	0.021			
<hr/>							
640: 100%	Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
14/100	0G	2.427	5.266	3.795	2.13	77	
14	5.2661	1/1 1.1it/s 0.9s	3.00%	0.00%	0.00%		
Sobe							

P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	3.1it/	
0.0227	all	1	26	0.0107	0.139	0.047	
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
15/100	0G	2.636	5.105	3.856	2.178	173	640: 100% <hr/>
15	5.1055	1.5s/it	1.5s	0.00%	0.00%	0.00%	Cai ↓
<hr/>							
P50-95) s 0.4s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	2.5it/	
0.0227	all	1	26	0.0107	0.139	0.047	
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
16/100	0G	2.415	5.115	3.748	2.074	110	640: 100% <hr/>
16	5.1154	1.2s/it	1.2s	0.00%	0.00%	0.00%	...
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	3.2it/	
0.0238	all	1	26	0.00758	0.111	0.0519	
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
17/100	0G	2.441	4.996	3.868	1.987	183	640: 100% <hr/>
17	4.9959	1.3s/it	1.3s	0.00%	0.00%	0.00%	Sobe ↑
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	3.1it/	
0.0238	all	1	26	0.00758	0.111	0.0519	
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
18/100	0G	2.245	4.85	3.811	1.953	107	640: 100% <hr/>
18	4.8498	1.1s/it	1.1s	0.00%	0.00%	0.00%	Cai ↓
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	3.1it/	
0.0238	all	1	26	0.00758	0.111	0.0519	
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
19/100	0G	2.619	4.955	3.867	2.092	314	640: 100% <hr/>
19	4.9554	1.8s/it	1.8s	0.00%	0.00%	0.00%	...

P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	3.2it/	
0.0238	all	1	26	0.00758	0.111	0.0519	0.00388 0.0278 0.0301 0.0211
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
20/100	0G	2.378	4.773	3.866	1.981	195	640: 100% 1/1 1.6s/it 1.6s
20 Cai ↓	4.7726	3.01%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.4s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	2.8it/	
0.0224	all	1	26	0.0114	0.139	0.05	0.00392 0.0278 0.0301 0.0211
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
21/100	0G	2.506	4.662	3.822	2.06	159	640: 100% 1/1 1.4s/it 1.4s
21 Sobe ↑	4.6623	3.01%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.4s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	2.5it/	
0.0224	all	1	26	0.0114	0.139	0.05	0.00392 0.0278 0.0301 0.0211
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
22/100	0G	2.778	4.87	3.869	2.016	307	640: 100% 1/1 1.7s/it 1.7s
22 ...	4.8699	3.01%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	3.0it/	
0.0224	all	1	26	0.0114	0.139	0.05	0.00392 0.0278 0.0301 0.0211
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
23/100	0G	2.434	4.738	3.778	1.933	129	640: 100% 1/1 1.1s/it 1.1s
23 Cai ↓	4.7383	3.01%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.5s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R		mAP50 mA
					1/1	2.1it/	
0.0224	all	1	26	0.0114	0.139	0.05	0.00392 0.0278 0.0301 0.0211
<hr/>							
Epoch	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	Size
24/100	0G	2.339	5.047	3.789	2.031	100	640: 100% 1/1 1.1s/it 1.1s
24 ...	5.0469	3.01%	0.00%	0.00%	0.00%	0.00%	

P50-95) s 0.4s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R	mAP50 mA
0.00333	0.00362	0.0278	1 0.03	26 0.00706 0.009	0.111	0.015
25/100 640: 100% Cai ↓	0G 1/1 4.7028	2.108 1.1s/it 	4.703 1.1s 3.00%	3.676	1.836	131 0.00%
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R	mAP50 mA
0.00333	0.00362	0.0278	1 0.03	26 0.00706 0.009	0.111	0.015
26/100 640: 100% Cai ↓	0G 1/1 4.6333	2.484 1.1it/s 	4.633 0.9s 3.00%	3.675	2.136	75 0.00%
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R	mAP50 mA
0.00333	0.00362	0.0278	1 0.03	26 0.00706 0.009	0.111	0.015
27/100 640: 100% ...	0G 1/1 4.6644	2.197 1.1s/it 	4.664 1.1s 3.00%	3.695	1.855	120 0.00%
P50-95) s 0.4s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R	mAP50 mA
0.00333	0.00362	0.0278	1 0.03	26 0.00706 0.009	0.111	0.015
28/100 640: 100% Cai ↓	0G 1/1 4.4989	2.181 1.3s/it 	4.499 1.3s 3.00%	3.742	1.786	180 0.00%
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 100%)	R	mAP50 mA
0.00333	0.00362	0.0278	1 0.03	26 0.00706 0.009	0.111	0.015
29/100 640: 100% ...	0G 1/1 4.7822	2.516 1.2it/s 	4.782 0.8s 3.00%	3.721	2.165	60 0.00%

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.0012	all 0	1 0	26 0	0.00366 0	0.0833	0.00593	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
30/100	0G	2.109	4.504	3.649	1.737		142
640: 100%	1/1	1.3s/it	1.3s				
30 Cai ↓	4.5036	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.0012	all 0	1 0	26 0	0.00366 0	0.0833	0.00593	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
31/100	0G	2.12	4.604	3.775	1.773		240
640: 100%	1/1	1.5s/it	1.5s				
31	4.6044	0.00%	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.0012	all 0	1 0	26 0	0.00366 0	0.0833	0.00593	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
32/100	0G	2.154	4.647	3.73	1.771		165
640: 100%	1/1	1.2s/it	1.2s				
32	4.6471	0.00%	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.0012	all 0	1 0	26 0	0.00366 0	0.0833	0.00593	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
33/100	0G	2.037	4.625	3.613	1.851		61
640: 100%	1/1	1.3it/s	0.8s				
33 Cai ↓	4.6251	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.0012	all 0	1 0	26 0	0.00366 0	0.0833	0.00593	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
34/100	0G	2.19	4.634	3.655	1.843		109
640: 100%	1/1	1.2s/it	1.2s				
34	4.6343	0.00%	0.00%	0.00%	0.00%	0.00%	
...							

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.0012	all 0	1 0	26 0	0.00366 0	0.0833	0.00593	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
35/100	0G	2.116	4.583	3.667	1.787		137
640: 100%	1/1	1.2s/it	1.2s				
35 Cai	4.5834	0.00%	0.00%	0.00%	0.00%	0.00%	
	P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA
0.000471	all 0	1 0	26 0	0.00351 0	0.0833	0.00235	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
36/100	0G	2.183	4.238	3.577	1.827		90
640: 100%	1/1	1.1it/s	0.9s				
36 Cai	4.2377	0.00%	0.00%	0.00%	0.00%	0.00%	
	P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA
0.000471	all 0	1 0	26 0	0.00351 0	0.0833	0.00235	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
37/100	0G	2.078	4.605	3.689	1.704		215
640: 100%	1/1	1.5s/it	1.5s				
37 ... Cai	4.6053	0.00%	0.00%	0.00%	0.00%	0.00%	
	P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA
0.000471	all 0	1 0	26 0	0.00351 0	0.0833	0.00235	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
38/100	0G	1.758	4.569	3.534	1.626		74
640: 100%	1/1	1.1it/s	0.9s				
38 Cai	4.5686	0.00%	0.00%	0.00%	0.00%	0.00%	
	P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA
0.000471	all 0	1 0	26 0	0.00351 0	0.0833	0.00235	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
39/100	0G	2.01	4.467	3.666	1.861		70
640: 100%	1/1	1.2it/s	0.8s				
39 Cai	4.4668	0.00%	0.00%	0.00%	0.00%	0.00%	

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000471	all 0	1 0	26 0	0.00351 0	0.0833	0.00235	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
40/100	0G	2.033	4.442	3.683	1.771		103
640: 100%	1/1	1.0s/it	1.0s				
40 Cai ↓	4.4418	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.4s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000471	all 0	1 0	26 0	0.00351 0	0.0833	0.00235	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
41/100	0G	2.073	4.439	3.538	1.873		66
640: 100%	1/1	1.1it/s	0.9s				
41 Cai ↓	4.4393	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000471	all 0	1 0	26 0	0.00351 0	0.0833	0.00235	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
42/100	0G	2.101	4.501	3.661	1.745		123
640: 100%	1/1	1.2s/it	1.2s				
42 ...	4.5012	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00025	all 0	1 0	26 0	0.00351 0	0.0833	0.0025	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
43/100	0G	2.194	4.562	3.582	1.937		66
640: 100%	1/1	1.2it/s	0.8s				
43 ...	4.5623	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.4s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00025	all 0	1 0	26 0	0.00351 0	0.0833	0.0025	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
44/100	0G	2.155	4.616	3.621	1.73		211
640: 100%	1/1	1.4s/it	1.4s				
44 ...	4.6164	0.00%	0.00%	0.00%	0.00%	0.00%	

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00025	all 0	1 0	26 0	0.00351 0	0.0833	0.0025	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
45/100	0G	1.997	4.239	3.588	1.744		90
640: 100%	1/1	1.1it/s 0.9s					
45 Cai ↓	4.2390	0.00%	0.00%	0.00%	0.00%		
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00025	all 0	1 0	26 0	0.00351 0	0.0833	0.0025	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
46/100	0G	2.29	4.698	3.691	1.811		312
640: 100%	1/1	1.7s/it 1.7s					
46 ...	4.6980	0.00%	0.00%	0.00%	0.00%		
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00025	all 0	1 0	26 0	0.00351 0	0.0833	0.0025	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
47/100	0G	2.033	4.45	3.624	1.717		158
640: 100%	1/1	1.2s/it 1.2s					
47 Cai ↓	4.4502	0.00%	0.00%	0.00%	0.00%		
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00025	all 0	1 0	26 0	0.00351 0	0.0833	0.0025	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
48/100	0G	2.123	4.387	3.594	1.908		71
640: 100%	1/1	1.2it/s 0.9s					
48 Cai ↓	4.3870	0.00%	0.00%	0.00%	0.00%		
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00025	all 0	1 0	26 0	0.00351 0	0.0833	0.0025	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
49/100	0G	1.837	4.363	3.605	1.688		68
640: 100%	1/1	1.2it/s 0.8s					
49 Cai ↓	4.3633	0.00%	0.00%	0.00%	0.00%		

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00025	all 0	1 0	26 0	0.00351 0	0.0833	0.0025	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
50/100	0G	1.979	4.285	3.568	1.723	132	
640: 100%	1/1	1.2s/it	1.2s				
50 Cai ↓	4.2851	0.00%	0.00%	0.00%	0.00%	0.00%	
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
51/100	0G	1.974	4.35	3.529	1.684	111	
640: 100%	1/1	1.0it/s	1.0s				
51	4.3500	0.00%	0.00%	0.00%	0.00%	0.00%	
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
52/100	0G	1.828	4.277	3.513	1.66	83	
640: 100%	1/1	1.1it/s	0.9s				
52 Cai ↓	4.2772	0.00%	0.00%	0.00%	0.00%	0.00%	
<hr/>							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
53/100	0G	1.868	4.167	3.523	1.593	210	
640: 100%	1/1	1.5s/it	1.5s				
53 Cai ↓	4.1673	0.00%	0.00%	0.00%	0.00%	0.00%	
<hr/>							
P50-95) s 0.4s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
54/100	0G	1.92	4.278	3.513	1.619	162	
640: 100%	1/1	1.3s/it	1.3s				
54 ...	4.2776	0.00%	0.00%	0.00%	0.00%	0.00%	

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
55/100	0G	2.19	4.423	3.685	1.736	305	
640: 100%	1/1	1.7s/it	1.7s				
55	4.4234	0.00%	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
56/100	0G	2.026	4.428	3.616	1.659	248	
640: 100%	1/1	1.5s/it	1.5s				
56	4.4279	0.00%	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.4s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
57/100	0G	1.913	4.294	3.446	1.716	86	
640: 100%	1/1	1.1lit/s	0.9s				
57	4.2938	0.00%	0.00%	0.00%	0.00%	0.00%	
Cai ↓							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
58/100	0G	1.735	4.24	3.436	1.588	69	
640: 100%	1/1	1.2lit/s	0.8s				
58	4.2395	0.00%	0.00%	0.00%	0.00%	0.00%	
Cai ↓							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
59/100	0G	2.059	4.378	3.604	1.71	177	
640: 100%	1/1	1.3s/it	1.3s				
59	4.3775	0.00%	0.00%	0.00%	0.00%	0.00%	
...							

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000425	all 0	1 0	26 0	0.00351 0	0.0833	0.00425	1/1 3.2it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
60/100	0G	2.143	4.442	3.628	1.77	240	
640: 100%	1/1	1.6s/it 4.4424	1.6s 0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	1/1 3.2it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
61/100	0G	1.881	4.306	3.399	1.763	59	
640: 100%	1/1	1.3it/s 4.3063	0.8s 0.00%	0.00%	0.00%	0.00%	
Cai ↓							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	1/1 3.3it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
62/100	0G	1.908	4.141	3.313	1.728	55	
640: 100%	1/1	1.3it/s 4.1409	0.8s 0.00%	0.00%	0.00%	0.00%	
Cai ↓							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	1/1 3.3it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
63/100	0G	1.748	4.081	3.427	1.598	91	
640: 100%	1/1	1.1it/s 4.0815	0.9s 0.00%	0.00%	0.00%	0.00%	
Cai ↓							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	1/1 2.9it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
64/100	0G	2.146	4.351	3.582	1.719	234	
640: 100%	1/1	1.6s/it 4.3512	1.6s 0.00%	0.00%	0.00%	0.00%	
...							

P50-95) s 0.4s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
65/100	0G	1.908	4.089	3.432	1.721	68	
640: 100%	1/1 4.0888	1.2it/s 0.9s	0.00%	0.00%	0.00%	0.00%	
Cai ↓							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
66/100	0G	1.961	4.352	3.519	1.739	75	
640: 100%	1/1 4.3522	1.2it/s 0.9s	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
67/100	0G	1.866	4.248	3.463	1.518	168	
640: 100%	1/1 4.2480	1.3s/it 1.3s	0.00%	0.00%	0.00%	0.00%	
Cai ↓							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
68/100	0G	1.99	4.316	3.488	1.635	179	
640: 100%	1/1 4.3155	1.3s/it 1.3s	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	
<hr/>							
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
69/100	0G	1.859	4.047	3.514	1.588	134	
640: 100%	1/1 4.0470	1.2s/it 1.2s	0.00%	0.00%	0.00%	0.00%	
Cai ↓							

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
70/100	0G	2.136	4.453	3.573	1.673	297	
640: 100%	1/1	1.6s/it	1.6s				
70	4.4533	0.00%	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
71/100	0G	2.057	4.373	3.503	1.648	229	
640: 100%	1/1	1.5s/it	1.5s				
71	4.3729	0.00%	0.00%	0.00%	0.00%	0.00%	
Cai 🔍							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.00032	all 0	1 0	26 0	0.00375 0	0.0833	0.0032	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
72/100	0G	1.853	4.272	3.468	1.561	189	
640: 100%	1/1	1.4s/it	1.4s				
72	4.2719	0.00%	0.00%	0.00%	0.00%	0.00%	
Cai 🔍							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
73/100	0G	1.983	4.292	3.463	1.525	259	
640: 100%	1/1	1.5s/it	1.5s				
73	4.2915	0.00%	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
74/100	0G	1.877	4.177	3.381	1.518	170	
640: 100%	1/1	1.2s/it	1.2s				
74	4.1766	0.00%	0.00%	0.00%	0.00%	0.00%	
Cai 🔍							

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
75/100	0G	2.069	4.426	3.508	1.648	275	
640: 100%	1/1	1.6s/it	1.6s				
75	4.4256	0.00%	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
76/100	0G	1.968	4.43	3.453	1.57	196	
640: 100%	1/1	1.4s/it	1.4s				
76	4.4300	0.00%	0.00%	0.00%	0.00%	0.00%	
...							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
77/100	0G	1.962	4.223	3.458	1.718	81	
640: 100%	1/1	1.1it/s	0.9s				
77	4.2225	0.00%	0.00%	0.00%	0.00%	0.00%	
Cai ↓							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
78/100	0G	1.831	4.098	3.342	1.559	136	
640: 100%	1/1	1.1s/it	1.1s				
78	4.0977	0.00%	0.00%	0.00%	0.00%	0.00%	
Cai ↓							
P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
79/100	0G	2.161	4.332	3.542	1.647	283	
640: 100%	1/1	1.6s/it	1.6s				
79	4.3324	0.00%	0.00%	0.00%	0.00%	0.00%	
...							

P50-95) s 0.3s	Class Mask(P)	Images R	Instances		Box(P mAP50-95): 100%	R	mAP50 mA
			mAP50	mAP50-95)			
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	1/1 3.3it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
80/100	0G	1.713	4.046	3.341	1.538	125	
640: 100%	1/1	1.1s/it	1.1s				
80 Cai ↓	4.0464	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	1/1 3.2it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
81/100	0G	2.043	4.357	3.542	1.651	255	
640: 100%	1/1	1.5s/it	1.5s				
81 ...	4.3567	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	1/1 3.0it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
82/100	0G	1.838	4.157	3.399	1.543	172	
640: 100%	1/1	1.3s/it	1.3s				
82 Cai ↓	4.1566	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	1/1 3.0it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
83/100	0G	2.278	4.436	3.567	1.638	301	
640: 100%	1/1	1.6s/it	1.6s				
83 ...	4.4355	0.00%	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances	Box(P mAP50-95): 100%	R	mAP50 mA	
0.000303	all 0	1 0	26 0	0.00383 0	0.0833	0.00303	1/1 3.0it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
84/100	0G	1.873	4.062	3.38	1.565	178	
640: 100%	1/1	1.3s/it	1.3s				
84 Cai ↓	4.0619	0.00%	0.00%	0.00%	0.00%	0.00%	

P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 26 0.00383 0.0833 0.00303	R	mAP50 mA
0.000303	all 0	1 0	26 0 0	0.00383 0.0833 0.00303		1/1 3.2it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
85/100 640: 100% 85 Cai ↓	0G 1/1 4.0494	1.695 1.1it/s 0.9s 0.00%	4.049 0.00%	3.328 0.00%	1.546 0.00%	94 0.00%
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 26 0.00383 0.0833 0.00303	R	mAP50 mA
0.000303	all 0	1 0	26 0 0	0.00383 0.0833 0.00303		1/1 3.2it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
86/100 640: 100% 86 ...	0G 1/1 4.2454	1.878 1.2s/it 1.2s 0.00%	4.245 0.00%	3.402 0.00%	1.619 0.00%	134 0.00%
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 26 0.00402 0.0833 0.00269	R	mAP50 mA
0.000791	all 0	1 0	26 0 0	0.00402 0.0833 0.00269		1/1 3.0it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
87/100 640: 100% 87 ...	0G 1/1 4.4252	2.036 1.6s/it 1.6s 0.00%	4.425 0.00%	3.395 0.00%	1.578 0.00%	257 0.00%
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 26 0.00402 0.0833 0.00269	R	mAP50 mA
0.000791	all 0	1 0	26 0 0	0.00402 0.0833 0.00269		1/1 3.1it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
88/100 640: 100% 88 Cai ↓	0G 1/1 3.9201	1.886 1.2it/s 0.8s 0.00%	3.92 0.00%	3.307 0.00%	1.643 0.00%	66 0.00%
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P 26 0.00402 0.0833 0.00269	R	mAP50 mA
0.000791	all 0	1 0	26 0 0	0.00402 0.0833 0.00269		1/1 3.3it/
Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances
89/100 640: 100% 89 ...	0G 1/1 4.3710	1.975 1.5s/it 1.5s 0.00%	4.371 0.00%	3.389 0.00%	1.526 0.00%	211 0.00%

P50-95) s 0.4s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P R 1/1 2.8it/	mAP50 mA	
0.000791	all 0	1 0	26 0	0.00402 0	0.0833 0.00269	
	Epoch Size	GPU_mem	box_loss seg_loss	cls_loss df_l_loss	Instances	
90/100	0G	1.763	4.149	3.349	1.614	77
640: 100%	1/1	1.0it/s	1.0s			
90 Cai ↓	4.1492	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.4s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P R 1/1 2.3it/	mAP50 mA	
0.000791	all 0	1 0	26 0	0.00402 0	0.0833 0.00269	
Closing dataloader mosaic						
	Epoch Size	GPU_mem	box_loss seg_loss	cls_loss df_l_loss	Instances	
91/100	0G	2.005	4.012	3.324	1.697	95
640: 100%	1/1	1.0it/s	1.0s			
91 Cai ↓	4.0122	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P R 1/1 3.0it/	mAP50 mA	
0.000791	all 0	1 0	26 0	0.00402 0	0.0833 0.00269	
	Epoch Size	GPU_mem	box_loss seg_loss	cls_loss df_l_loss	Instances	
92/100	0G	1.964	4.089	3.283	1.48	103
640: 100%	1/1	1.1it/s	1.0s			
92 ...	4.0888	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.4s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P R 1/1 2.6it/	mAP50 mA	
0.000791	all 0	1 0	26 0	0.00402 0	0.0833 0.00269	
	Epoch Size	GPU_mem	box_loss seg_loss	cls_loss df_l_loss	Instances	
93/100	0G	1.828	3.986	3.237	1.513	103
640: 100%	1/1	1.0it/s	1.0s			
93 Cai ↓	3.9865	0.00%	0.00%	0.00%	0.00%	
P50-95) s 0.3s	Class Mask(P)	Images R	Instances mAP50 mAP50-95): 100%	Box(P R 1/1 3.1it/	mAP50 mA	
0.000791	all 0	1 0	26 0	0.00402 0	0.0833 0.00269	
	Epoch Size	GPU_mem	box_loss seg_loss	cls_loss df_l_loss	Instances	
94/100	0G	1.841	3.874	3.251	1.548	96
640: 100%	1/1	1.0it/s	1.0s			
94	3.8738	0.00%	0.00%	0.00%	0.00%	

Cai ↓		Class	Images	Instances	Box(P R)	R	mAP50	mA
P50-95)	s 0.3s	Mask(P	R	mAP50 mAP50-95): 100%			1/1	3.0it/
0.000791		all 0	1 0	26 0	0.00402 0	0.0833	0.00269	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
95/100	640: 100%	0G	2.141	4.173	3.292	1.521	103	
95		4.1728	0.00%	0.00%	0.00%	0.00%		
...								
P50-95)	s 0.4s	Class	Images	Instances	Box(P R)	R	mAP50	mA
		Mask(P	R	mAP50 mAP50-95): 100%			1/1	2.7it/
0.000791		all 0	1 0	26 0	0.00402 0	0.0833	0.00269	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
96/100	640: 100%	0G	2.077	4.03	3.28	1.819	78	
96		4.0301	0.00%	0.00%	0.00%	0.00%		
Cai ↓								
P50-95)	s 0.3s	Class	Images	Instances	Box(P R)	R	mAP50	mA
		Mask(P	R	mAP50 mAP50-95): 100%			1/1	2.9it/
0.000791		all 0	1 0	26 0	0.00402 0	0.0833	0.00269	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
97/100	640: 100%	0G	2.431	4.691	3.539	1.619	103	
97		4.6913	0.00%	0.00%	0.00%	0.00%		
...								
P50-95)	s 0.3s	Class	Images	Instances	Box(P R)	R	mAP50	mA
		Mask(P	R	mAP50 mAP50-95): 100%			1/1	3.0it/
0.000791		all 0	1 0	26 0	0.00402 0	0.0833	0.00269	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
98/100	640: 100%	0G	2.487	4.54	3.426	1.653	103	
98		4.5398	0.00%	0.00%	0.00%	0.00%		
Cai ↓								
P50-95)	s 0.3s	Class	Images	Instances	Box(P R)	R	mAP50	mA
		Mask(P	R	mAP50 mAP50-95): 100%			1/1	2.9it/
0.000791		all 0	1 0	26 0	0.00402 0	0.0833	0.00269	
	Epoch Size	GPU_mem	box_loss	seg_loss	cls_loss	dfl_loss	Instances	
99/100	640: 100%	0G	1.925	3.992	3.249	1.668	85	
99		3.9919	0.00%	0.00%	0.00%	0.00%		

```
Cai ↓ |  

P50-95) Mask(P Class Images Instances Box(P R mAP50 mA  

s 0.3s R mAP50 mAP50-95): 100% ————— 1/1 3.1it/  

0.000791 all 1 26 0.00402 0.0833 0.00269  

0 0 0 0  

Epoch GPU_mem box_loss seg_loss cls_loss df1_loss Instances  

Size 100/100 0G 2.001 3.988 3.324 1.72 85  

640: 100% ————— 1/1 1.2it/s 0.9s  

| 100 | 3.9879 | 0.00% | 0.00% | 0.00% |  

Cai ↓ |  

P50-95) Mask(P Class Images Instances Box(P R mAP50 mA  

s 0.3s R mAP50 mAP50-95): 100% ————— 1/1 3.1it/  

0.000791 all 1 26 0.00402 0.0833 0.00269  

0 0 0 0
```

100 epochs completed in 0.052 hours.

Optimizer stripped from C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Viso Computacional\VC_TF\Yolo_resultados\Yolo_treino3\weights\last.pt, 6.8M B

Optimizer stripped from C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Viso Computacional\VC_TF\Yolo_resultados\Yolo_treino3\weights\best.pt, 6.8M B

Validating C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Viso Computacional\VC_TF\Yolo_resultados\Yolo_treino3\weights\best.pt...

Ultralytics 8.3.235 Python-3.13.7 torch-2.9.1+cpu CPU (AMD Ryzen 5 5625U with Radeon Graphics)

YOLOv8n-seg summary (fused): 85 layers, 3,258,844 parameters, 0 gradients, 11.3 G FLOPs

```
P50-95) Mask(P Class Images Instances Box(P R mAP50 mA  

s 0.2s R mAP50 mAP50-95): 100% ————— 1/1 4.5it/  

0.0268 0.00392 0.0278 0.0301 0.0211 0.111 0.0519  

0.0268 0.00392 0.0278 0.0301 0.0211
```

Speed: 2.3ms preprocess, 114.2ms inference, 0.0ms loss, 52.7ms postprocess per image

Results saved to C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Viso Computacional\VC_TF\Yolo_resultados\Yolo_treino3

FIM! O gráfico deve mostrar a linha de erro no chão.

Resultados salvos em: C:\Users\renat\OneDrive\Mestrado em Contabilidade\2025\2025.2c Viso Computacional\VC_TF\Yolo_resultados\Yolo_treino

Tempo de execução → 00:03:15.77 (hh:mm:ss)

Finalizado em → 07/12/2025 às 15:51:46
