

📖 Comparação de Segmentação: YOLOv12-seg (fallbacks) vs RT-DETR-Seg (Transformers)

Fluxo: 1) GPU/ambiente → 2) `CONFIG` → 3) Montar Drive e validar datasets (YOLO/COCO) → 4) Instalar libs e seeds → 5) Treinar **YOLO-seg** (ordem: `yolo12l-seg.pt` → `yolo11l-seg.pt` → `yolov8l-seg.pt`) → 6) Treinar **RT-DETR-Seg** no **COCO** (fallback para **Mask2Former**) → 7) Avaliar (AP bbox/segm), salvar amostras → 8) Comparar métricas (tabela + CSV) → 9) Visualizações lado a lado → 10) Exportáveis (ONNX, .zip) → 11) `report.json` → 12) Notas/Ajustes.

Decisões de design:

- `IMG_SIZE=1024` por equilíbrio entre detalhe e VRAM numa T4.
- `rect=True` no YOLO preserva proporção por batch.
- RT-DETR-Seg via Transformers: tenta checkpoint *instance seg*; se indisponível, cai para **Mask2Former** mantendo a comparação **transformer + mask head**.

Como mudar hiperparâmetros: edite o bloco **CONFIG** (`IMG_SIZE`, `EPOCHS`, `BATCH`, `MODEL_SIZE`, `CONF`, `IOU_THRES`). Early stopping configurado nas chamadas de treino.

1) Verificação de GPU e ambiente (com detecção de compatibilidade CUDA)

```
# @title 1) Verificação de GPU e ambiente (com detecção de compatibilid
import sys, subprocess, json, platform, os, re

print("Python:", sys.version)
print("Platform:", platform.platform())

# GPU / driver
nvsmi = ""
try:
    nvsmi = subprocess.check_output(["nvidia-smi"], text=True)
    print(nvsmi)
except Exception as e:
    print("nvidia-smi not available or error:", e)

# Extrai versão "CUDA Version: 12.x" do nvidia-smi, se possível
driver_cuda = None
m = re.search(r"CUDA Version:\s*([0-9]+)\.([0-9]+)", nvsmi)
```

```

if m:
    driver_cuda = (int(m.group(1)), int(m.group(2))) # (major, minor)

# PyTorch
try:
    import torch
    print("PyTorch:", torch.__version__)
    print("CUDA available:", torch.cuda.is_available())
    if torch.cuda.is_available():
        print("GPU:", torch.cuda.get_device_name(0))
        print("CUDA capability:", torch.cuda.get_device_capability(0))
        runtime = torch.version.cuda # ex: '12.6'
        print("Torch CUDA runtime:", runtime)
        rt = tuple(int(x) for x in runtime.split('.')) if runtime else

        # Se runtime (ex: 12.6) for maior que o suportado pelo driver (
        if driver_cuda and rt and (rt[0] == driver_cuda[0]) and (rt[1]
            os.environ["TORCH_NEED_CU124"] = "1"
            print(">> Aviso: Driver mostra CUDA", f"{driver_cuda[0]}.{d
                "mas o PyTorch está com CUDA", runtime, "→ rebaixar p
        else:
            os.environ["TORCH_NEED_CU124"] = "0"
    else:
        # Sem CUDA? prossegue, mas não vamos tentar cu124.
        os.environ["TORCH_NEED_CU124"] = "0"
except Exception as e:
    print("Torch info error:", e)
    os.environ["TORCH_NEED_CU124"] = "0"

# Snapshot rápido de libs já presentes
def get_ver(module):
    try:
        m = __import__(module)
        return getattr(m, "__version__", "unknown")
    except:
        return "not installed"

print(json.dumps({
    "ultralytics": get_ver("ultralytics"),
    "transformers": get_ver("transformers"),
    "datasets": get_ver("datasets"),
    "evaluate": get_ver("evaluate"),
    "accelerate": get_ver("accelerate"),
    "pycocotools": get_ver("pycocotools"),
    "torchvision": get_ver("torchvision"),
}, indent=2))

```

Python: 3.12.12 (main, Oct 10 2025, 08:52:57) [GCC 11.4.0]

Platform: Linux-6.6.105+-x86_64-with-glibc2.35

Sat Oct 25 22:10:04 2025

NVIDIA-SMI 550.54.15			Driver Version: 550.54.15		CUDA	
GPU	Name		Persistence-M	Bus-Id	Disp.A	Vol
Fan	Temp	Perf	Pwr:Usage/Cap		Memory-Usage	GPU
=====						
0	Tesla T4		Off	00000000:00:04.0	Off	
N/A	69C	P8	11W / 70W	0MiB / 15360MiB		

Processes:						
GPU	GI	CI	PID	Type	Process name	
	ID	ID				
No running processes found						

PyTorch: 2.8.0+cu126

CUDA available: True

GPU: Tesla T4

CUDA capability: (7, 5)

Torch CUDA runtime: 12.6

>> Aviso: Driver mostra CUDA 12.4 mas o PyTorch está com CUDA 12.6 → reb
{

```
{
  "ultralytics": "not installed",
  "transformers": "4.57.1",
  "datasets": "4.0.0",
  "evaluate": "not installed",
  "accelerate": "1.11.0",
  "pycocotools": "unknown",
  "torchvision": "0.23.0+cu126"
}
```

✓ 2) Bloco de CONFIG

```
# @title 2) Bloco de CONFIG
CONFIG = {
    "IMG_SIZE": 1024,
    "EPOCHS": 100,
    "BATCH": 8,
    "SEED": 42,
    "MODEL_SIZE": "l",    # 'n','s','m','l','x'
    "CONF": 0.25,
    "IOU_THRES": 0.70,

    # Paths do Drive
    "YOLO_DIR": "/content/drive/MyDrive/Dataset/YOLO_seg_original",
    "COCO_DIR": "/content/drive/MyDrive/Dataset/COCO_original",

    # Saídas no Drive
    "OUT_YOLO": "/content/drive/MyDrive/Experimentos/seg_yolo",
    "OUT_RTDETR": "/content/drive/MyDrive/Experimentos/seg_transformer"

    # YOLO
    "YOLO_WORKERS": 2,
    "YOLO_PATIENCE": 20,

    # Transformers
    "RT_USE_AMP": True,
    "RT_LR": 2e-5,
    "RT_WEIGHT_DECAY": 1e-4,
    "RT_WARMUP_STEPS": 200,
    "RT_GRAD_ACCUM": 2,
    "RT_EVAL_CONF": 0.05,
}
print(CONFIG)
```

```
{'IMG_SIZE': 1024, 'EPOCHS': 100, 'BATCH': 8, 'SEED': 42, 'MODEL_SIZE':
```

✓ 3) Montar Drive e validações (YOLO + COCO) + corrigir data.yaml

```
# @title 3) Montar Drive e validações (YOLO + COCO) + corrigir data.yaml
import os, re, glob, yaml, shutil, pathlib, json
from google.colab import drive

# Monta o Drive
drive.mount('/content/drive', force_remount=True)
```

```

YOLO_DIR = CONFIG["YOLO_DIR"]
COCO_DIR = CONFIG["COCO_DIR"]

assert os.path.isdir(YOLO_DIR), f"YOLO_DIR não encontrado: {YOLO_DIR}"
assert os.path.isdir(COCO_DIR), f"COCO_DIR não encontrado: {COCO_DIR}"

def count_images_labels(split_dir):
    img_dir = os.path.join(split_dir, "images")
    lbl_dir = os.path.join(split_dir, "labels")
    imgs = sorted(glob.glob(os.path.join(img_dir, "**", "*.jpg"), recur
                        glob.glob(os.path.join(img_dir, "**", "*.png"), recur
                        glob.glob(os.path.join(img_dir, "**", "*.jpeg"), recu
    lbls = sorted(glob.glob(os.path.join(lbl_dir, "**", "*.txt"), recur
    # pares por stem
    img_stems = set([os.path.splitext(os.path.basename(p))[0] for p in
    lbl_stems = set([os.path.splitext(os.path.basename(p))[0] for p in
    missing_lbl = img_stems - lbl_stems
    missing_img = lbl_stems - img_stems
    return imgs, lbls, missing_lbl, missing_img

print("== Checando YOLO splits ==")
for split in ["train", "valid", "test"]:
    sd = os.path.join(YOLO_DIR, split)
    assert os.path.isdir(sd), f"Split {split} não encontrado em {YOLO_D
    imgs, lbls, missing_lbl, missing_img = count_images_labels(sd)
    print(f"{split}: imgs={len(imgs)} lbls={len(lbls)}")
    assert len(imgs) > 0 and len(lbls) > 0, f"Split {split} vazio."
    assert len(missing_lbl) == 0, f"{split}: imagens sem label: {list(s
    assert len(missing_img) == 0, f"{split}: labels sem imagem: {list(s

print("\n== Checando/Padronizando COCO ==")
ann_dir = os.path.join(COCO_DIR, "annotations")
os.makedirs(ann_dir, exist_ok=True)

def is_coco_json(p):
    try:
        with open(p, "r") as f: j = json.load(f)
        return all(k in j for k in ["images","annotations","categories"]
    except: return False

# Se não existirem instances_*.json, tentamos copiar de _annotations.co
required = {
    "train": os.path.join(ann_dir, "instances_train.json"),
    "val": os.path.join(ann_dir, "instances_val.json"),
    "test": os.path.join(ann_dir, "instances_test.json"),
}

def try_copy_from_split(split_key, split_folder_name):

```

```

split_dir = os.path.join(COCO_DIR, split_folder_name)
cands = [
    os.path.join(split_dir, "_annotations.coco.json"),
    os.path.join(split_dir, "annotation.json"),
    os.path.join(split_dir, "anotation.json"),
    os.path.join(split_dir, "annotations.json"),
    os.path.join(split_dir, f"instances_{split_key}.json"),
]
for p in cands:
    if os.path.exists(p) and is_coco_json(p):
        shutil.copy2(p, required[split_key])
        return True, p
return False, None

# 'valid' como alias de 'val'
split_map = {"train": "train", "val": "val" if os.path.isdir(os.path.join

for sk, dst in required.items():
    if not os.path.exists(dst):
        ok, src = try_copy_from_split(sk, split_map[sk])
        if ok:
            print(f"[OK] {sk}: copiado {src} -> {dst}")
        else:
            print(f"[WARN] {sk}: não encontrado JSON COCO; será necessá

# Verifica presença final
print()
for name, path in [("instances_train", required["train"]), ("instances_
    print(name, "->", os.path.exists(path), path)

# Corrige data.yaml (caminhos absolutos + chave 'val')
data_yaml_guess = os.path.join(YOLO_DIR, "data.yaml")
assert os.path.exists(data_yaml_guess), f"data.yaml não encontrado em {

with open(data_yaml_guess, "r") as f:
    data = yaml.safe_load(f)

if "valid" in data and "val" not in data:
    data["val"] = data.pop("valid")

def absolutize(p):
    if p is None: return p
    if isinstance(p, list):
        return [absolutize(x) for x in p]
    p = str(p)
    if p.startswith("/"):
        return p
    return os.path.normpath(os.path.join(YOLO_DIR, p))

```

```

for key in ["path", "train", "val", "test"]:
    if key in data:
        data[key] = absolutize(data[key])

data.pop("path", None)

os.makedirs(CONFIG["OUT_YOLO"], exist_ok=True)
fixed_yaml = os.path.join(CONFIG["OUT_YOLO"], "data_fixed.yaml")
with open(fixed_yaml, "w") as f:
    yaml.safe_dump(data, f)
print("\nSalvo data.yaml corrigido em:", fixed_yaml)

```

```

Mounted at /content/drive
== Checando YOLO splits ==
train: imgs=25 lbls=25
valid: imgs=6 lbls=6
test: imgs=3 lbls=3
\n== Checando/Padronizando COCO ==

instances_train -> True /content/drive/MyDrive/Dataset/COCO_original/annot
instances_val -> True /content/drive/MyDrive/Dataset/COCO_original/annot
instances_test -> True /content/drive/MyDrive/Dataset/COCO_original/annot
\nSalvo data.yaml corrigido em: /content/drive/MyDrive/Experimentos/seg_

```

4) Instalação de dependências e seeds (fix cu124 + versão do pycocotools)

```

# @title 4) Instalação de dependências e seeds (fix cu124 + versão do p
import os, random, numpy as np

need_cu124 = os.environ.get("TORCH_NEED_CU124", "0") == "1"

if need_cu124:
    print(">> Ajustando PyTorch para cu124 (driver CUDA 12.4 detectado)
    # Instala versões compatíveis com cu124 disponíveis oficialmente
    !pip install -q --upgrade --force-reinstall torch==2.6.0+cu
else:
    print("Mantendo PyTorch atual.")

# Demais dependências (idempotente)
!pip install -q -U ultralytics pycocotools transformers datasets evalua

# Imports pós-instalação
import torch, torchvision, transformers, datasets, evaluate, ultralytic
from importlib.metadata import version, PackageNotFoundError
try:

```

```

coco_ver = version("pycocotools")
except PackageNotFoundError:
    coco_ver = "not installed"

print({
    "torch": torch.__version__,
    "torchvision": torchvision.__version__,
    "torch_cuda_runtime": torch.version.cuda,
    "ultralytics": ultralytics.__version__,
    "transformers": transformers.__version__,
    "datasets": datasets.__version__,
    "evaluate": evaluate.__version__,
    "pycocotools": coco_ver,
})

# Seeds determinísticas
def set_all_seeds(seed=42):
    random.seed(seed); np.random.seed(seed); torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

set_all_seeds(CONFIG["SEED"])

```

```

>> Ajustando PyTorch para cu124 (driver CUDA 12.4 detectado) → torch 2.6
_____ 24.6/24.6 MB 85.6 MB/s eta
_____ 883.7/883.7 kB 71.0 MB/s eta
_____ 13.8/13.8 MB 130.9 MB/s eta
_____ 664.8/664.8 MB 1.3 MB/s eta
_____ 363.4/363.4 MB 1.5 MB/s eta
_____ 211.5/211.5 MB 6.2 MB/s eta
_____ 56.3/56.3 MB 15.0 MB/s eta
_____ 127.9/127.9 MB 8.0 MB/s eta
_____ 207.5/207.5 MB 5.6 MB/s eta
_____ 188.7/188.7 MB 6.3 MB/s eta
_____ 99.1/99.1 kB 8.2 MB/s eta
_____ 21.1/21.1 MB 82.2 MB/s eta
_____ 6.2/6.2 MB 129.3 MB/s eta
_____ 62.1/62.1 kB 6.5 MB/s eta
_____ 536.2/536.2 kB 49.1 MB/s eta
_____ 768.4/768.4 MB 1.5 MB/s eta
_____ 7.3/7.3 MB 38.4 MB/s eta 0:0
_____ 150.1/150.1 MB 6.9 MB/s eta
_____ 166.7/166.7 MB 6.4 MB/s eta
_____ 6.6/6.6 MB 111.7 MB/s eta 0:
_____ 44.6/44.6 kB 4.3 MB/s eta 0:
_____ 199.3/199.3 kB 20.4 MB/s eta
_____ 134.9/134.9 kB 13.0 MB/s eta
_____ 2.0/2.0 MB 82.2 MB/s eta 0:0
_____ 16.6/16.6 MB 57.4 MB/s eta 0



```



```

930.8/930.8 kB 55.4 MB/s eta
ERROR: pip's dependency resolver does not currently take into account all
ipypthon 7.34.0 requires jedi>=0.16, which is not installed.
gcsfs 2025.3.0 requires fsspec==2025.3.0, but you have fsspec 2025.9.0 w
tensorflow 2.19.0 requires numpy<2.2.0,>=1.26.0, but you have numpy 2.3.
opencv-contrib-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version
torchaudio 2.8.0+cu126 requires torch==2.8.0, but you have torch 2.6.0+cu
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.3.3 which i
cupy-cuda12x 13.3.0 requires numpy<2.3,>=1.22, but you have numpy 2.3.3
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2; python_version
datasets 4.0.0 requires fsspec[http]<=2025.3.0,>=2023.1.0, but you have
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9
62.0/62.0 kB 6.3 MB/s eta
1.1/1.1 MB 37.2 MB/s eta 0:0
506.8/506.8 kB 44.5 MB/s eta
84.1/84.1 kB 9.5 MB/s eta 0:
16.5/16.5 MB 129.8 MB/s eta
47.7/47.7 MB 13.7 MB/s eta 0

```

ERROR: pip's dependency resolver does not currently take into account all
pylibcudf-cu12 25.6.0 requires pyarrow<20.0.0a0,>=14.0.0; platform_machi
tensorflow 2.19.0 requires numpy<2.2.0,>=1.26.0, but you have numpy 2.2.
cudf-cu12 25.6.0 requires pyarrow<20.0.0a0,>=14.0.0; platform_machine ==
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.2.6 which i
WARNING  torchvision==0.21 is incompatible with torch==2.8.
Run 'pip install torchvision==0.23' to fix torchvision or 'pip install -
For a full compatibility table see [https://github.com/pytorch/vision#ins](https://github.com/pytorch/vision#install)
Creating new Ultralytics Settings v0.0.6 file 
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultr
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs
{'torch': '2.8.0+cu126', 'torchvision': '0.23.0+cu126', 'torch_cuda_runt

✓ 5) Treino YOLO-seg (v12→v11→v8) + Validação + Amostras

```

# @title 5) Treino YOLO-seg (v12→v11→v8) + Validação + Amostras
import os, time, json, glob
from datetime import datetime
from ultralytics import YOLO

os.makedirs(CONFIG["OUT_YOLO"], exist_ok=True)

ckpts = [
    f"yolo12{CONFIG['MODEL_SIZE']}-seg.pt",
    f"yolo11{CONFIG['MODEL_SIZE']}-seg.pt",
    f"yolov8{CONFIG['MODEL_SIZE']}-seg.pt",
]

chosen_ckpt = None
model = None
errors = []

```

```

--

for ck in ckpts:
    try:
        print(f"Tentando carregar: {ck}")
        model = YOLO(ck)
        chosen_ckpt = ck
        print("Carregado:", ck)
        break
    except Exception as e:
        print(f"Falhou: {ck} -> {e}")
        errors.append((ck, str(e)))

assert model is not None, f"Não foi possível carregar nenhum checkpoint

RUN_NAME = f"yolo_seg_{os.path.splitext(os.path.basename(chosen_ckpt))[

results = model.train(
    data=os.path.join(CONFIG["OUT_YOLO"], "data_fixed.yaml"),
    imgsz=CONFIG["IMG_SIZE"],
    epochs=CONFIG["EPOCHS"],
    batch=CONFIG["BATCH"],
    device=0,
    workers=CONFIG["YOLO_WORKERS"],
    pretrained=True,
    patience=CONFIG["YOLO_PATIENCE"],
    rect=True,
    project=CONFIG["OUT_YOLO"],
    name=RUN_NAME,
    seed=CONFIG["SEED"],
    exist_ok=True,
)

val_res = model.val(
    data=os.path.join(CONFIG["OUT_YOLO"], "data_fixed.yaml"),
    imgsz=CONFIG["IMG_SIZE"],
    device=0,
    split="val",
    conf=CONFIG["CONF"],
    iou=CONFIG["IOU_THRES"],
    project=CONFIG["OUT_YOLO"],
    name=RUN_NAME+"_val",
    save_json=True,
)

print("YOLO val metrics:", getattr(val_res, "results_dict", val_res))

# Predições de amostra (primeiras 8 imagens do split val)
import yaml, glob, os
with open(os.path.join(CONFIG["OUT_YOLO"], "data_fixed.yaml"), "r") as

```

```

d_yaml = yaml.safe_load(f)

val_img_dir = os.path.join(d_yaml["val"], "images") if isinstance(d_yaml["val"], dict) else d_yaml["val"]
sample_imgs = sorted(glob.glob(os.path.join(val_img_dir, "**", "*.jpg")))
pred_out_dir = os.path.join(CONFIG["OUT_YOLO"], RUN_NAME, "sample_preds")
os.makedirs(pred_out_dir, exist_ok=True)

preds = model.predict(
    source=sample_imgs,
    conf=CONFIG["CONF"],
    iou=CONFIG["IOU_THRES"],
    imgsz=CONFIG["IMG_SIZE"],
    device=0,
    save=True,
    project=pred_out_dir,
    name="yolo_samples",
    exist_ok=True,
)

print("Amostras YOLO salvas em:", os.path.join(pred_out_dir, "yolo_samp

```

6) RT-DETR-Seg (Transformers) — Dataset COCO + Treino + Avaliação

```

# @title 6) RT-DETR-Seg (Transformers) — Dataset COCO + Treino + Avaliação
import os, json, math, time, numpy as np, torch
from PIL import Image
from pycocotools.coco import COCO
from pycocotools import mask as mask_util
from torch.utils.data import Dataset, DataLoader
from tqdm.auto import tqdm

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
os.makedirs(CONFIG["OUT_RTDETR"], exist_ok=True)

# === Dataset COCO para instance segmentation ===
class COCOSegDataset(Dataset):
    def __init__(self, img_root, ann_json, processor, img_size=1024):
        self.coco = COCO(ann_json)
        self.img_root = img_root
        self.processor = processor
        self.img_ids = list(self.coco.imgs.keys())
        self.img_size = img_size

    # filtra imagens com ao menos uma anotação
    def __len__(self):
        keep = []
        for img_id in self.img_ids:

```

```

        anns = self.coco.getAnnIds(imgIds=[img_id], iscrowd=None)
        if len(anns) > 0:
            keep.append(img_id)
        self.img_ids = keep

def __len__(self):
    return len(self.img_ids)

def __getitem__(self, idx):
    img_id = self.img_ids[idx]
    info = self.coco.loadImgs([img_id])[0]
    file_name = info["file_name"]
    path = os.path.join(self.img_root, file_name)
    image = Image.open(path).convert("RGB")

    ann_ids = self.coco.getAnnIds(imgIds=[img_id], iscrowd=None)
    anns = self.coco.loadAnns(ann_ids)
    anns = [a for a in anns if "bbox" in a and "category_id" in a]

    encoded = self.processor(
        images=image,
        annotations={"image_id": img_id, "annotations": anns},
        return_tensors="pt"
    )
    # remove dim de batch artificial
    for k in encoded:
        if isinstance(encoded[k], torch.Tensor):
            encoded[k] = encoded[k].squeeze(0)
        elif isinstance(encoded[k], list) and len(encoded[k]) == 1:
            encoded[k] = encoded[k][0]
    encoded["image_id"] = img_id
    encoded["orig_size"] = (image.height, image.width)
    return encoded

def collate_fn(batch):
    pixel_values = torch.stack([b["pixel_values"] for b in batch])
    pixel_mask = torch.stack([b["pixel_mask"] for b in batch]) if "pixel_mask" in batch[0] else None
    labels = [b["labels"] for b in batch]
    image_ids = [b["image_id"] for b in batch]
    orig_sizes = [b["orig_size"] for b in batch]
    res = {"pixel_values": pixel_values, "labels": labels, "image_ids": image_ids}
    if pixel_mask is not None:
        res["pixel_mask"] = pixel_mask
    return res

# == Escolha do modelo/processador (RT-DETR-Seg → fallback Mask2Former)
from transformers import AutoImageProcessor, AutoModelForInstanceSegmentation

```

```

chosen_transformer = None
chosen_processor = None
model_candidates = [
    # tentativa ideal (pode não existir no HF)
    "PaddlePaddle/rt-detr-hf-l-seg",
    # alternativas Mask2Former (instance)
    "facebook/mask2former-swin-large-coco-instance",
    "facebook/mask2former-swin-base-coco-instance",
    "facebook/mask2former-swin-small-coco-instance",
]

load_errors = []
for name in model_candidates:
    try:
        print(f"Tentando carregar: {name}")
        chosen_processor = AutoImageProcessor.from_pretrained(name)
        chosen_transformer = AutoModelForInstanceSegmentation.from_pretrained(name)
        print("Carregado:", name)
        break
    except Exception as e:
        print(f"Falhou: {name} -> {e}")
        load_errors.append((name, str(e)))

assert chosen_transformer is not None, f"Não foi possível carregar modelo"
chosen_transformer.to(device)

# === Datasets/DataLoaders (COCO) ===
train_img_dir = os.path.join(CONFIG["COCO_DIR"], "train")
val_img_dir = os.path.join(CONFIG["COCO_DIR"], "val")
ann_dir = os.path.join(CONFIG["COCO_DIR"], "annotations")

train_json = os.path.join(ann_dir, "instances_train.json")
val_json = os.path.join(ann_dir, "instances_val.json")
assert os.path.exists(train_json) and os.path.exists(val_json), "JSONs não encontrados"

train_ds = COCOSegDataset(train_img_dir, train_json, chosen_processor,
                           val_ds = COCOSegDataset(val_img_dir, val_json, chosen_processor,

rt_batch = max(1, min(2, CONFIG["BATCH"])) # batch pequeno por VRAM
from torch.utils.data import DataLoader
train_loader = DataLoader(train_ds, batch_size=rt_batch, shuffle=True,
val_loader = DataLoader(val_ds, batch_size=1, shuffle=False,

# === Otimizador e agendamento ===
from torch.optim import AdamW
total_steps = math.ceil(len(train_loader) * CONFIG["EPOCHS"] / CONFIG["RT_BATCH_SIZE"])
optimizer = AdamW(chosen_transformer.parameters(), lr=CONFIG["RT_LR"],

```

```

def lr_lambda(step):
    w = CONFIG["RT_WARMUP_STEPS"]
    if step < w:
        return float(step) / max(1, w)
    return 1.0

scheduler = torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda)
scaler = torch.cuda.amp.GradScaler(enabled=CONFIG["RT_USE_AMP"])

# === Loop de treino ===
chosen_transformer.train()
global_step = 0
print(f"Iniciando treino Transformers: epochs={CONFIG['EPOCHS']} batch=

for epoch in range(CONFIG["EPOCHS"]):
    pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{CONFIG['EPOCHS']}")
    optimizer.zero_grad(set_to_none=True)
    for step, batch in enumerate(pbar):
        pixel_values = batch["pixel_values"].to(device)
        labels = [{k: (v.to(device) if isinstance(v, torch.Tensor) else
        pixel_mask = batch.get("pixel_mask")
        if pixel_mask is not None:
            pixel_mask = pixel_mask.to(device)

        with torch.cuda.amp.autocast(enabled=CONFIG["RT_USE_AMP"]):
            outputs = chosen_transformer(pixel_values=pixel_values, pix
            loss = outputs.loss / CONFIG["RT_GRAD_ACCUM"]

        scaler.scale(loss).backward()

        if (step + 1) % CONFIG["RT_GRAD_ACCUM"] == 0:
            scaler.step(optimizer)
            scaler.update()
            optimizer.zero_grad(set_to_none=True)
            scheduler.step()
            global_step += 1

    pbar.set_postfix({"loss": float(loss.item())})

# === Avaliação COCO (bbox e segm) ===
from pycocotools.cocoeval import COCOeval

chosen_transformer.eval()
coco_gt = COCO(val_json)
bbox_dets, segm_dets = [], []
infer_times = []

with torch.no_grad():

```

```

for batch in tqdm(val_loader, desc="Inferência validação"):
    pixel_values = batch["pixel_values"].to(device)
    pixel_mask = batch.get("pixel_mask")
    if pixel_mask is not None:
        pixel_mask = pixel_mask.to(device)
    image_ids = batch["image_ids"]
    orig_sizes = batch["orig_sizes"] # (H, W)

    start = time.time()
    outputs = chosen_transformer(pixel_values=pixel_values, pixel_r
    if torch.cuda.is_available():
        torch.cuda.synchronize()
    infer_times.append((time.time() - start) * 1000.0) # ms

    try:
        post = chosen_processor.post_process_instance_segmentation(
            outputs, threshold=CONFIG["RT_EVAL_CONF"], target_sizes
        )[0]
        scores = post["scores"].cpu().numpy().tolist()
        labels = post["labels"].cpu().numpy().tolist()
        boxes = post["boxes"].cpu().numpy().tolist()
        masks = post["masks"].cpu().numpy()
        img_id = int(image_ids[0])
        for s, c, b, m in zip(scores, labels, boxes, masks):
            x1, y1, x2, y2 = b
            w = max(0.0, x2 - x1); h = max(0.0, y2 - y1)
            coco_bbox = [float(x1), float(y1), float(w), float(h)]
            bbox_dets.append({
                "image_id": img_id,
                "category_id": int(c),
                "bbox": coco_bbox,
                "score": float(s),
            })
            m_bin = (m > 0.5).astype(np.uint8)
            rle = mask_util.encode(np.asfortranarray(m_bin))
            rle["counts"] = rle["counts"].decode("ascii")
            segm_dets.append({
                "image_id": img_id,
                "category_id": int(c),
                "segmentation": rle,
                "score": float(s),
            })
    except Exception:
        post = chosen_processor.post_process_object_detection(
            outputs, threshold=CONFIG["RT_EVAL_CONF"], target_sizes
        )[0]
        scores = post["scores"].cpu().numpy().tolist()
        labels = post["labels"].cpu().numpy().tolist()

```

```

boxes = post["boxes"].cpu().numpy().tolist()
img_id = int(image_ids[0])
for s, c, b in zip(scores, labels, boxes):
    x1,y1,x2,y2 = b
    w = max(0.0, x2 - x1); h = max(0.0, y2 - y1)
    coco_bbox = [float(x1), float(y1), float(w), float(h)]
    bbox_dets.append({
        "image_id": img_id,
        "category_id": int(c),
        "bbox": coco_bbox,
        "score": float(s),
    })

# Salva resultados
from datetime import datetime
rt_out_dir = os.path.join(CONFIG["OUT_RTDETR"], "preds_" + datetime.now)
os.makedirs(rt_out_dir, exist_ok=True)
bbox_json = os.path.join(rt_out_dir, "bbox_results.json")
segm_json = os.path.join(rt_out_dir, "segm_results.json")
with open(bbox_json, "w") as f: json.dump(bbox_dets, f)
with open(segm_json, "w") as f: json.dump(segm_dets, f)
print("Salvos:", bbox_json, segm_json)

# COCOeval
coco_dt_bbox = coco_gt.loadRes(bbox_json) if len(bbox_dets)>0 else None
bbox_metrics = {}
if coco_dt_bbox is not None:
    coco_eval_bbox = COCOeval(coco_gt, coco_dt_bbox, iouType="bbox")
    coco_eval_bbox.evaluate(); coco_eval_bbox.accumulate(); coco_eval_b
    bbox_metrics = {
        "AP@[.5:.95]_bbox": float(coco_eval_bbox.stats[0]),
        "AP50_bbox": float(coco_eval_bbox.stats[1]),
        "AP75_bbox": float(coco_eval_bbox.stats[2]),
    }

segm_metrics = {}
if len(segm_dets) > 0:
    coco_dt_segm = coco_gt.loadRes(segm_json)
    coco_eval_segm = COCOeval(coco_gt, coco_dt_segm, iouType="segm")
    coco_eval_segm.evaluate(); coco_eval_segm.accumulate(); coco_eval_s
    segm_metrics = {
        "AP@[.5:.95]_segm": float(coco_eval_segm.stats[0]),
        "AP50_segm": float(coco_eval_segm.stats[1]),
        "AP75_segm": float(coco_eval_segm.stats[2]),
    }

rt_infer_ms = float(np.mean(infer_times)) if infer_times else None
print("Transformer bbox metrics:", bbox_metrics)

```



```

print("Transformer segm metrics:", segm_metrics)
print("Transformer infer time (ms/img):", rt_infer_ms)

# Amostras salvas
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

sample_save = os.path.join(rt_out_dir, "samples")
os.makedirs(sample_save, exist_ok=True)

val_ids = val_ds.img_ids[:6]
for img_id in val_ids:
    info = val_ds.coco.loadImgs([img_id])[0]
    path = os.path.join(val_img_dir, info["file_name"])
    img = np.array(Image.open(path).convert("RGB"))

    bds = [d for d in bbox_dets if d["image_id"] == img_id and d["score"] > 0.5]
    sds = [d for d in segm_dets if d["image_id"] == img_id and d["score"] > 0.5]

    fig = plt.figure(figsize=(8, 8))
    ax = plt.gca()
    ax.imshow(img)
    for d in bds:
        x,y,w,h = d["bbox"]
        ax.add_patch(Rectangle((x,y), w, h, fill=False, linewidth=1.5))
    for d in sds[:20]:
        m = mask_util.decode(d["segmentation"])
        ax.contour(m, levels=[0.5], linewidths=1.0)
    ax.set_axis_off()
    plt.tight_layout()
    plt.savefig(os.path.join(sample_save, f"{img_id}.png"), dpi=150)
    plt.close()

print("Amostras de RT/Mask2Former salvas em:", sample_save)

```

✓ 7) Coleta de métricas + comparação (DataFrame + CSV)

```

# @title 7) Coleta de métricas + comparação (DataFrame + CSV)
import os, json, pandas as pd, torch, time, numpy as np, glob, yaml

# YOLO metrics
yolo_run_dir = os.path.join(CONFIG["OUT_YOLO"], RUN_NAME)
yolo_results_json = os.path.join(yolo_run_dir, "results.json")
yolo_metrics = {}
if os.path.exists(yolo_results_json):
    with open(yolo_results_json, "r") as f:
        yolo_metrics = json.load(f)

```

```

else:
    try:
        yolo_metrics = getattr(val_res, "results_dict", {})
    except:
        yolo_metrics = {}

def safe_get(d, keys, default=None):
    for k in keys:
        if k in d: return d[k]
    return default

yolo_map50      = safe_get(yolo_metrics, ["metrics/precision(B)", "metri
yolo_map5095_b  = safe_get(yolo_metrics, ["metrics/mAP50-95(B)", "map", "
yolo_map5095_m  = safe_get(yolo_metrics, ["metrics/mAP50-95(M)", "map_ma

# parâmetros YOLO
yolo_params = None
try:
    yolo_params = sum(p.numel() for p in model.model.parameters()) / 1e
except Exception:
    pass

# tempo de inferência YOLO (mediana em ~8 imgs)
with open(os.path.join(CONFIG["OUT_YOLO"], "data_fixed.yaml"), "r") as
    d_yaml = yaml.safe_load(f)
imgs_inf = sorted(glob.glob(os.path.join(d_yaml["val"], "images", "**"),
times = []
with torch.no_grad():
    for p in imgs_inf:
        s = time.time()
        _ = model.predict(source=p, imgsz=CONFIG["IMG_SIZE"], conf=CONF
        if torch.cuda.is_available(): torch.cuda.synchronize()
        times.append((time.time()-s)*1000.0)
yolo_infer_ms = float(np.median(times)) if times else None

# Transformer metrics (do passo anterior)
rt_bbox_ap    = bbox_metrics.get("AP@[.5:.95]_bbox")
rt_segm_ap    = segm_metrics.get("AP@[.5:.95]_segm")
rt_ap50_bbox  = bbox_metrics.get("AP50_bbox")
rt_ap50_segm  = segm_metrics.get("AP50_segm")

rt_params = sum(p.numel() for p in chosen_transformer.parameters()) / 1
rt_gflops = None
rt_infer = rt_infer_ms

df = pd.DataFrame([
    {
        "model": f"YOLO{chosen_ckpt.replace('.pt', '')}",

```

```

        "imgsz": CONFIG["IMG_SIZE"],
        "epochs": CONFIG["EPOCHS"],
        "params(M)": round(yolo_params, 2) if yolo_params else None,
        "GFLOPs": None,
        "mAP50": round(yolo_map50, 4) if yolo_map50 is not None else No
        "mAP50-95 (bbox)": round(yolo_map5095_b, 4) if yolo_map5095_b i
        "mAP50-95 (mask)": round(yolo_map5095_m, 4) if yolo_map5095_m i
        "infer_time_ms/img": round(yolo_infer_ms, 2) if yolo_infer_ms i
    },
    {
        "model": getattr(chosen_transformer.config, "_name_or_path", "t
        "imgsz": CONFIG["IMG_SIZE"],
        "epochs": CONFIG["EPOCHS"],
        "params(M)": round(rt_params, 2),
        "GFLOPs": rt_gflops,
        "mAP50": round(rt_ap50_bbox, 4) if rt_ap50_bbox is not None els
        "mAP50-95 (bbox)": round(rt_bbox_ap, 4) if rt_bbox_ap is not No
        "mAP50-95 (mask)": round(rt_segm_ap, 4) if rt_segm_ap is not No
        "infer_time_ms/img": round(rt_infer, 2) if rt_infer is not None
    }
])

cmp_csv = os.path.join(CONFIG["OUT_RTDETR"], "comparacao_metrics.csv")
os.makedirs(CONFIG["OUT_RTDETR"], exist_ok=True)
df.to_csv(cmp_csv, index=False)
print("CSV salvo em:", cmp_csv)
df

```

8) Visualizações lado a lado (YOLO vs RT/Mask2Former) no split de validação

```

# @title 8) Visualizações lado a lado (YOLO vs RT/Mask2Former) no split
import os, glob, numpy as np, matplotlib.pyplot as plt
from PIL import Image
from matplotlib.patches import Rectangle

with open(os.path.join(CONFIG["OUT_YOLO"], "data_fixed.yaml"), "r") as
    d_yaml = yaml.safe_load(f)

val_imgs = sorted(glob.glob(os.path.join(d_yaml["val"], "images", "**"),
side_by_side_dir = os.path.join(CONFIG["OUT_RTDETR"], "side_by_side")
os.makedirs(side_by_side_dir, exist_ok=True)

def yolo_predict_on_image(img_path):
    res = model.predict(source=img_path, imgsz=CONFIG["IMG_SIZE"], conf
    boxes, masks, scores = [], [], []

```

```

    if len(res) > 0 and hasattr(res[0], "boxes"):
        b = res[0].boxes
        if b is not None:
            for i in range(len(b)):
                boxes.append(b.xyxy[i].cpu().numpy().tolist())
                scores.append(float(b.conf[i].item()))
    if len(res) > 0 and getattr(res[0], "masks", None) is not None and
        for m in res[0].masks.data.cpu().numpy():
            masks.append(m)
    return boxes, masks, scores

def rt_predict_on_image(img_path):
    im = Image.open(img_path).convert("RGB")
    enc = chosen_processor(images=im, return_tensors="pt").to(device)
    with torch.no_grad():
        out = chosen_transformer(**enc)
    H, W = im.height, im.width
    try:
        post = chosen_processor.post_process_instance_segmentation(out,
            boxes = post["boxes"].cpu().numpy().tolist()
            scores = post["scores"].cpu().numpy().tolist()
            masks = post["masks"].cpu().numpy()
    except Exception:
        post = chosen_processor.post_process_object_detection(out, thre
            boxes = post["boxes"].cpu().numpy().tolist()
            scores = post["scores"].cpu().numpy().tolist()
            masks = []
    return boxes, masks, scores

for p in val_imgs:
    img = np.array(Image.open(p).convert("RGB"))
    y_boxes, y_masks, y_scores = yolo_predict_on_image(p)
    r_boxes, r_masks, r_scores = rt_predict_on_image(p)

    fig, axes = plt.subplots(1, 2, figsize=(14, 7))
    axes[0].imshow(img); axes[0].set_title("YOLO-seg")
    for bb in y_boxes:
        x1,y1,x2,y2 = bb
        axes[0].add_patch(Rectangle((x1,y1), x2-x1, y2-y1, fill=False,
    for m in y_masks[:20]:
        axes[0].contour(m, levels=[0.5], linewidths=1.0)
    axes[0].axis("off")

    axes[1].imshow(img); axes[1].set_title("RT-DETR-Seg (fallback Mask2
    for bb in r_boxes:
        x1,y1,x2,y2 = bb
        axes[1].add_patch(Rectangle((x1,y1), x2-x1, y2-y1, fill=False,
    for m in r_masks[:20]:

```

```
axes[1].contour((m>0.5).astype(np.uint8), levels=[0.5], linewidth  
axes[1].axis("off")  
  
import os  
plt.tight_layout()  
base = os.path.splitext(os.path.basename(p))[0]  
outp = os.path.join(side_by_side_dir, f"{base}_compare.png")  
plt.savefig(outp, dpi=150)  
plt.close()  
  
print("Visualizações lado a lado salvas em:", side_by_side_dir)
```

✓ 9) Exportáveis (YOLO → ONNX) e compactação de runs

```
# @title 9) Exportáveis (YOLO → ONNX) e compactação de runs
import os, shutil, zipfile
from ultralytics import YOLO

onnx_path = None
try:
    exp = model.export(format="onnx", imgsz=CONFIG["IMG_SIZE"], opset=1)
    onnx_path = exp
    print("Modelo YOLO exportado para ONNX:", onnx_path)
except Exception as e:
    print("Falha ao exportar ONNX:", e)

def zip_dir(src_dir, zip_path):
    with zipfile.ZipFile(zip_path, "w", zipfile.ZIP_DEFLATED) as zf:
        for root, _, files in os.walk(src_dir):
            for f in files:
                fp = os.path.join(root, f)
                arc = os.path.relpath(fp, src_dir)
                zf.write(fp, arc)

yolo_zip = os.path.join(CONFIG["OUT_YOLO"], f"{RUN_NAME}.zip")
try:
    zip_dir(os.path.join(CONFIG["OUT_YOLO"], RUN_NAME), yolo_zip)
    print("Zip YOLO salvo em:", yolo_zip)
except Exception as e:
    print("Zip YOLO falhou:", e)

rt_zip = os.path.join(CONFIG["OUT_RTDETR"], "transformer_run.zip")
try:
    zip_dir(os.path.join(CONFIG["OUT_RTDETR"]), rt_zip)
    print("Zip Transformers salvo em:", rt_zip)
except Exception as e:
    print("Zip Transformers falhou:", e)
```

✓ 10) Reprodutibilidade: salvar report.json

```
# @title 10) Reprodutibilidade: salvar report.json
import json, os, platform, torch, transformers, ultralytics, datasets,
report = {
    "config": CONFIG,
    "env": {
        "python": platform.python_version(),
        "torch": torch.__version__,
        "torchvision": torchvision.__version__,
        "transformers": transformers.__version__,
        "ultralytics": ultralytics.__version__,
        "datasets": datasets.__version__,
        "evaluate": evaluate.__version__,
        "cuda_available": torch.cuda.is_available(),
        "gpu_name": torch.cuda.get_device_name(0) if torch.cuda.is_avai
        "torch_cuda_runtime": torch.version.cuda,
    },
    "yolo": {
        "checkpoint": chosen_ckpt,
        "run_dir": os.path.join(CONFIG["OUT_YOLO"], RUN_NAME),
        "inference_ms": None, # preenchido na tabela
    },
    "transformer": {
        "name": getattr(chosen_transformer.config, "_name_or_path", Non
    },
    "comparacao_csv": os.path.join(CONFIG["OUT_RTDETR"], "comparacao_me
}
report_path = os.path.join(CONFIG["OUT_RTDETR"], "report.json")
os.makedirs(CONFIG["OUT_RTDETR"], exist_ok=True)
with open(report_path, "w") as f:
    json.dump(report, f, indent=2)
print("Report salvo em:", report_path)
```

Notas / Como ajustar

- **Por que `IMG_SIZE=1024` ?** Radiografias têm estruturas finas; 1024 preserva detalhe sem estourar VRAM na T4 (com batch moderado).
- **`rect=True` no YOLO:** batching retangular reduz padding e preserva proporções.
- **Ajustes comuns:** `BATCH` (VRAM), `EPOCHS` (treino mais longo), `MODEL_SIZE` (`"x"` para mais capacidade; `"m"/"s"` para menos VRAM), `patience` (early stopping do YOLO).
- **Transformers (RT/Mask2Former):** manter `batch=1-2`, usar `gradient_accumulation` e AMP (`autocast`) para caber na T4.
- **Fallback RT-DETR-Seg:** se não existir checkpoint RT-DETR com cabeça de máscara no HF, o notebook usa **Mask2Former** (*transformer + mask head*). O nome carregado aparece no log.
- **COCO vs YOLO:** YOLO treina no **YOLO-format**; o pipeline Transformers usa o **COCO instance segmentation**. Esta notebook centraliza/normaliza `instances_{train,val,test}.json` em `COCO_DIR/annotations` e tenta copiar de `_annotations.coco.json` dos splits, se necessário.