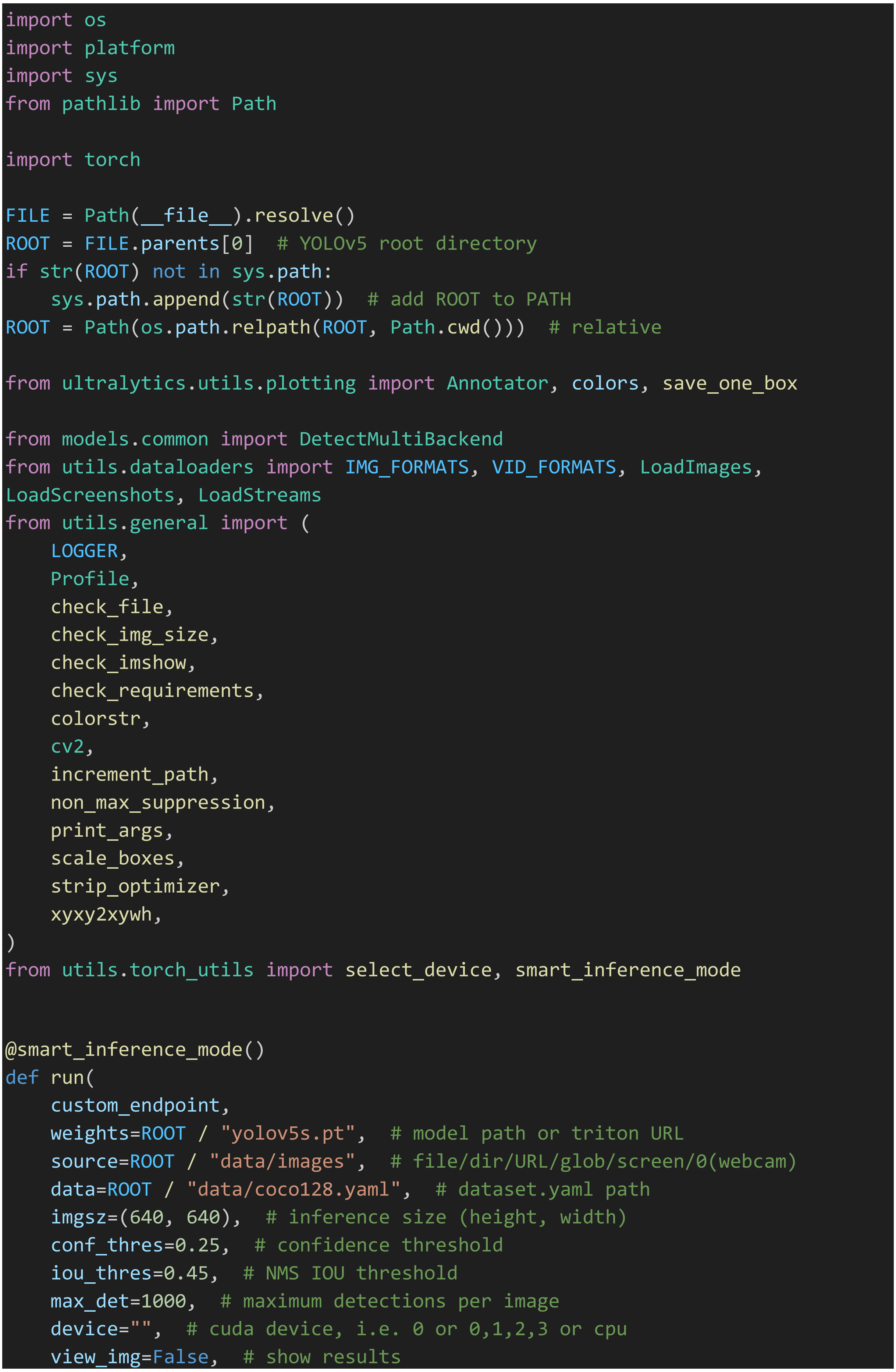
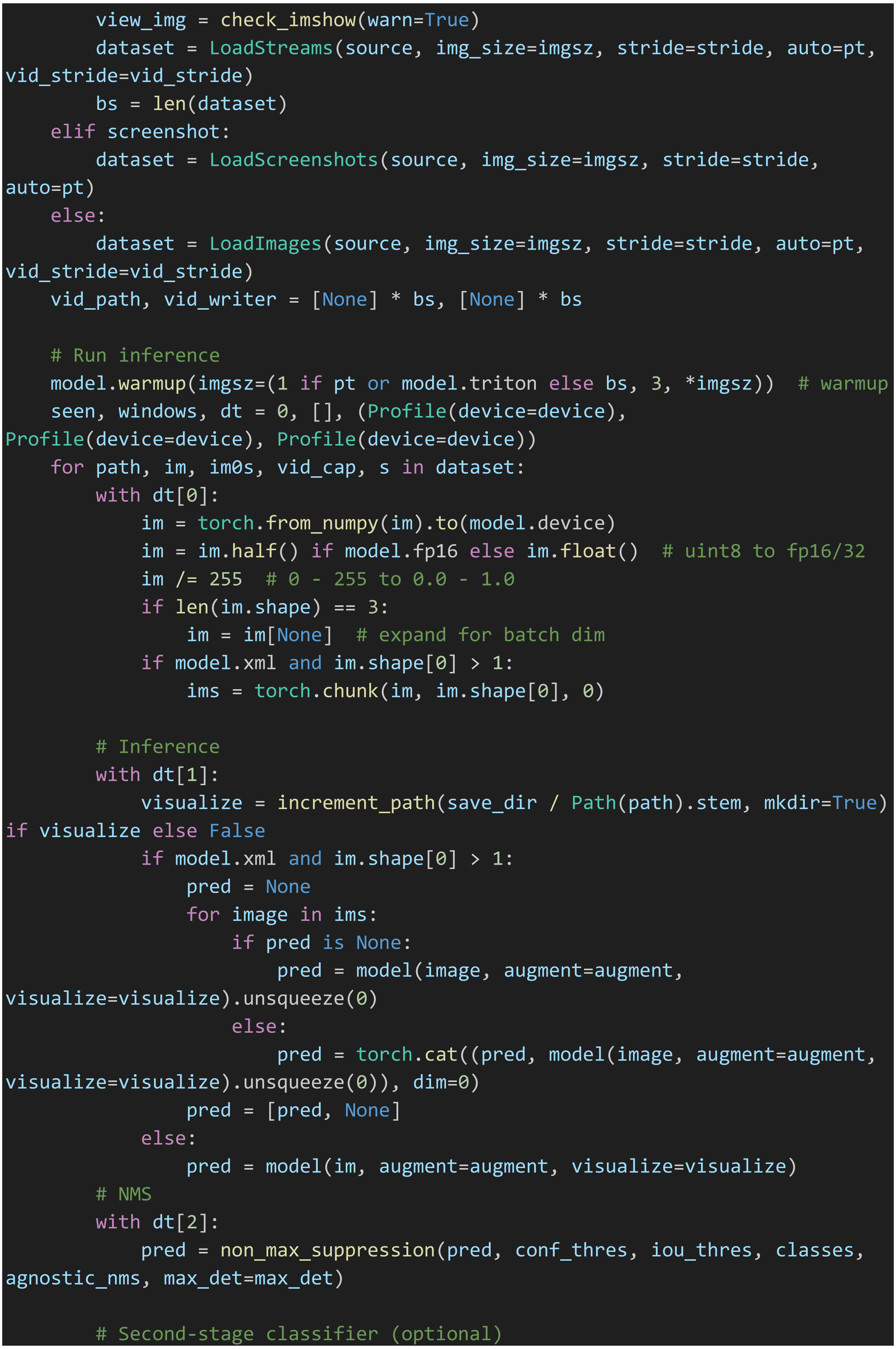
```
# YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
"""
Run YOLOv5 detection inference on images, videos, directories, globs, YouTube,
webcam, streams, etc.

Usage - sources:
    $ python detect.py --weights yolov5s.pt --source 0
      # webcam
                                                     img.jpg
      # image
                                                     vid.mp4
      # video
                                                     screen
       # screenshot
                                                     path/
      # directory
                                                     list.txt
       # list of images
                                                     list.streams
       # list of streams
                                                     'path/*.jpg'
       # glob

 'https://youtu.be/LNwODJXcvt4'  # YouTube

 'rtsp://example.com/media.mp4'  # RTSP, RTMP, HTTP stream

Usage - formats:
    $ python detect.py --weights yolov5s.pt                 # PyTorch
                                 yolov5s.torchscript        # TorchScript
                                 yolov5s.onnx               # ONNX Runtime or
OpenCV DNN with --dnn
                                 yolov5s_openvino_model     # OpenVINO
                                 yolov5s.engine             # TensorRT
                                 yolov5s.mlmodel            # CoreML (macOS-
only)
                                 yolov5s_saved_model        # TensorFlow
SavedModel
                                 yolov5s.pb                 # TensorFlow
GraphDef
                                 yolov5s.tflite             # TensorFlow Lite
                                 yolov5s_edgetpu.tflite     # TensorFlow Edge
TPU
                                 yolov5s_paddle_model       # PaddlePaddle
"""

import argparse
import csv
```



```
import os
import platform
import sys
from pathlib import Path

import torch

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

from ultralytics.utils.plotting import Annotator, colors, save_one_box

from models.common import DetectMultiBackend
from utils.dataloaders import IMG_FORMATS, VID_FORMATS, LoadImages,
LoadScreenshots, LoadStreams
from utils.general import (
    LOGGER,
    Profile,
    check_file,
    check_img_size,
    check_imshow,
    check_requirements,
    colorstr,
    cv2,
    increment_path,
    non_max_suppression,
    print_args,
    scale_boxes,
    strip_optimizer,
    xyxy2xywh,
)
from utils.torch_utils import select_device, smart_inference_mode


@smart_inference_mode()
def run(
    custom_endpoint,
    weights=ROOT / "yolov5s.pt",  # model path or triton URL
    source=ROOT / "data/images",  # file/dir/URL/glob/screen/0(webcam)
    data=ROOT / "data/coco128.yaml",  # dataset.yaml path
    imgsz=(640, 640),  # inference size (height, width)
    conf_thres=0.25,  # confidence threshold
    iou_thres=0.45,  # NMS IOU threshold
    max_det=1000,  # maximum detections per image
    device="",  # cuda device, i.e. 0 or 0,1,2,3 or cpu
    view_img=False,  # show results
    save_txt=False,  # save results to *.txt
    save_csv=False,  # save results in CSV format
    save_conf=False,  # save confidences in --save-txt labels
    save_crop=False,  # save cropped prediction boxes
    nosave=False,  # do not save images/videos
    classes=None,  # filter by class: --class 0, or --class 0 2 3
    agnostic_nms=False,  # class-agnostic NMS
    augment=False,  # augmented inference
    visualize=False,  # visualize features
    update=False,  # update all models
    project=ROOT / "runs/detect",  # save results to project/name
    name="exp",  # save results to project/name
    exist_ok=False,  # existing project/name ok, do not increment
    line_thickness=3,  # bounding box thickness (pixels)
    hide_labels=False,  # hide labels
    hide_conf=False,  # hide confidences
    half=False,  # use FP16 half-precision inference
    dnn=False,  # use OpenCV DNN for ONNX inference
    vid_stride=1,  # video frame-rate stride
):
    source = str(source)
    save_img = not nosave and not source.endswith(".txt")  # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(("rtsp://", "rtmp://", "http://", "https://"))
    webcam = source.isnumeric() or source.endswith(".streams") or (is_url and not is_file)
    screenshot = source.lower().startswith("screen")
    if is_url and is_file:
        source = check_file(source)  # download

    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment run
    (save_dir / "labels" if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # Load model
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    # Dataloader
    bs = 1  # batch_size
    if webcam:
```



```
        view_img = check_imshow(warn=True)
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt,
vid_stride=vid_stride)
        bs = len(dataset)
    elif screenshot:
        dataset = LoadScreenshots(source, img_size=imgsz, stride=stride,
auto=pt)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt,
vid_stride=vid_stride)
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt or model.triton else bs, 3, *imgsz))  # warmup
    seen, windows, dt = 0, [], (Profile(device=device),
Profile(device=device), Profile(device=device))
    for path, im, im0s, vid_cap, s in dataset:
        with dt[0]:
            im = torch.from_numpy(im).to(model.device)
            im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
            im /= 255  # 0 - 255 to 0.0 - 1.0
            if len(im.shape) == 3:
                im = im[None]  # expand for batch dim
            if model.xml and im.shape[0] > 1:
                ims = torch.chunk(im, im.shape[0], 0)

        # Inference
        with dt[1]:
            visualize = increment_path(save_dir / Path(path).stem, mkdir=True)
if visualize else False
            if model.xml and im.shape[0] > 1:
                pred = None
                for image in ims:
                    if pred is None:
                        pred = model(image, augment=augment,
visualize=visualize).unsqueeze(0)
                    else:
                        pred = torch.cat((pred, model(image, augment=augment,
visualize=visualize).unsqueeze(0)), dim=0)
                pred = [pred, None]
            else:
                pred = model(im, augment=augment, visualize=visualize)
        # NMS
        with dt[2]:
            pred = non_max_suppression(pred, conf_thres, iou_thres, classes,
agnostic_nms, max_det=max_det)

        # Second-stage classifier (optional)
        # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

        # Define the path for the CSV file
        csv_path = save_dir / "predictions.csv"

        # Create or append to the CSV file
        def write_to_csv(image_name, prediction, confidence):
            data = {"Image Name": image_name, "Prediction": prediction, "Confidence": confidence}
            with open(csv_path, mode="a", newline="") as f:
                writer = csv.DictWriter(f, fieldnames=data.keys())
                if not csv_path.is_file():
                    writer.writeheader()
                writer.writerow(data)

        # Process predictions
        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
                s += f"{i}: "
            else:
                p, im0, frame = path, im0s.copy(), getattr(dataset, "frame", 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
            txt_path = str(save_dir / "labels" / p.stem) + ("" if dataset.mode == "image" else f"_{frame}")  # im.txt
            s += "%gx%g " % im.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if save_crop else im0  # for save_crop
            annotator = Annotator(im0, line_width=line_thickness, example=str(names))
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                for c in det[:, 5].unique():
                    n = (det[:, 5] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

                # call custom endpoint on esp32
                custom_endpoint()

                # Write results
                for *xyxy, conf, cls in reversed(det):
                    c = int(cls)  # integer class
                    label = names[c] if hide_conf else f"{names[c]}"
                    confidence = float(conf)
                    confidence_str = f"{confidence:.2f}"

                    if save_csv:
                        write_to_csv(p.name, label, confidence_str)

                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
gn).view(-1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if save_conf else (cls,
*xywh)  # label format
                        with open(f"{txt_path}.txt", "a") as f:
                            f.write(("%g " * len(line)).rstrip() % line +
"\n")

                    if save_img or save_crop or view_img:  # Add bbox to image
                        c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if
hide_conf else f"{names[c]} {conf:.2f}")
                        annotator.box_label(xyxy, label, color=colors(c,
True))
                    if save_crop:
                        save_one_box(xyxy, imc, file=save_dir / "crops" /
names[c] / f"{p.stem}.jpg", BGR=True)

            # Stream results
            im0 = annotator.result()

            yield(b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + cv2.imencode('.jpg',
im0)[1].tobytes() + b'\r\n')


        # Print time (inference-only)
        LOGGER.info(f"{s}{'' if len(det) else '(no detections), '}{dt[1].dt *
1E3:.1f}ms")

    # Print results
    t = tuple(x.t / seen * 1e3 for x in dt)  # speeds per image
    LOGGER.info(f"Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per
image at shape {(1, 3, *imgsz)}" % t)
    if save_txt or save_img:
```

```python
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ""
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    if update:
        strip_optimizer(weights[0])  # update model (to fix SourceChangeWarning)


def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument("--weights", nargs="+", type=str, default=ROOT / "yolov5s.pt", help="model path or triton URL")
    parser.add_argument("--source", type=str, default=ROOT / "data/images", help="file/dir/URL/glob/screen/0(webcam)")
    parser.add_argument("--data", type=str, default=ROOT / "data/coco128.yaml", help="(optional) dataset.yaml path")
    parser.add_argument("--imgsz", "--img", "--img-size", nargs="+", type=int, default=[640], help="inference size h,w")
    parser.add_argument("--conf-thres", type=float, default=0.25, help="confidence threshold")
    parser.add_argument("--iou-thres", type=float, default=0.45, help="NMS IoU threshold")
    parser.add_argument("--max-det", type=int, default=1000, help="maximum detections per image")
    parser.add_argument("--device", default="", help="cuda device, i.e. 0 or 0,1,2,3 or cpu")
    parser.add_argument("--view-img", action="store_true", help="show results")
    parser.add_argument("--save-txt", action="store_true", help="save results to *.txt")
    parser.add_argument("--save-csv", action="store_true", help="save results in CSV format")
    parser.add_argument("--save-conf", action="store_true", help="save confidences in --save-txt labels")
    parser.add_argument("--save-crop", action="store_true", help="save cropped prediction boxes")
    parser.add_argument("--nosave", action="store_true", help="do not save images/videos")
    parser.add_argument("--classes", nargs="+", type=int, help="filter by class: --classes 0, or --classes 0 2 3")
    parser.add_argument("--agnostic-nms", action="store_true", help="class-agnostic NMS")
    parser.add_argument("--augment", action="store_true", help="augmented inference")
    parser.add_argument("--visualize", action="store_true", help="visualize features")
    parser.add_argument("--update", action="store_true", help="update all models")
    parser.add_argument("--project", default=ROOT / "runs/detect", help="save
results to project/name")
    parser.add_argument("--name", default="exp", help="save results to
project/name")
    parser.add_argument("--exist-ok", action="store_true", help="existing
project/name ok, do not increment")
    parser.add_argument("--line-thickness", default=3, type=int,
help="bounding box thickness (pixels)")
    parser.add_argument("--hide-labels", default=False, action="store_true",
help="hide labels")
    parser.add_argument("--hide-conf", default=False, action="store_true",
help="hide confidences")
    parser.add_argument("--half", action="store_true", help="use FP16 half-
precision inference")
    parser.add_argument("--dnn", action="store_true", help="use OpenCV DNN for
ONNX inference")
    parser.add_argument("--vid-stride", type=int, default=1, help="video
frame-rate stride")
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt


def main(opt):
    check_requirements(ROOT / "requirements.txt", exclude=("tensorboard",
"thop"))
    run(**vars(opt))


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