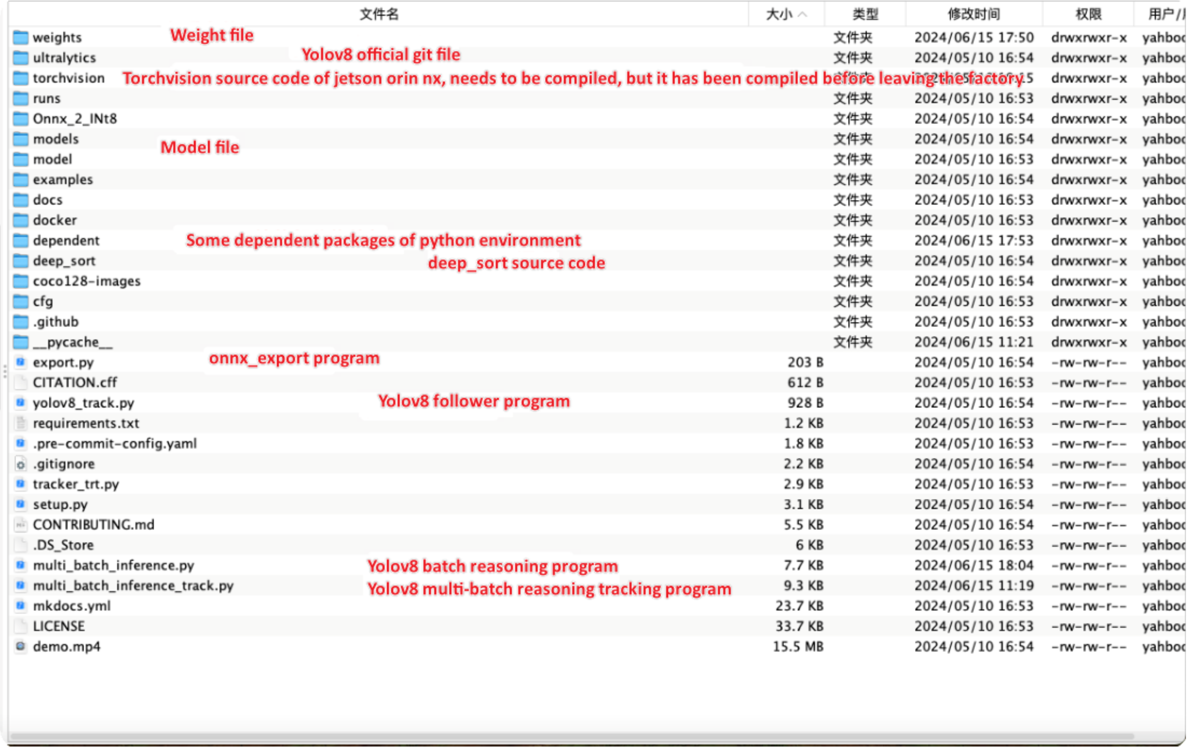
4.yolov8 target detection

Code Path for Yolov8

```
/home/yahboom/YBAMR-COBOT-EDU-00001/soft/yolov8
```

The following figure shows the code directory file structure and folder functions.



文件名	大小	类型	修改时间	权限	用户/
weights		文件夹	2024/06/15 17:50	drwxrwxr-x	yahboo
ultralytics		文件夹	2024/05/10 16:54	drwxrwxr-x	yahboo
torchvision		[illegible]	[illegible]	drwxrwxr-x	yahboo
runs		文件夹	2024/05/10 16:53	drwxrwxr-x	yahboo
Onnx_2_INt8		文件夹	2024/05/10 16:54	drwxrwxr-x	yahboo
models		文件夹	2024/05/10 16:54	drwxrwxr-x	yahboo
model		文件夹	2024/05/10 16:53	drwxrwxr-x	yahboo
examples		文件夹	2024/05/10 16:54	drwxrwxr-x	yahboo
docs		文件夹	2024/05/10 16:53	drwxrwxr-x	yahboo
docker		文件夹	2024/05/10 16:53	drwxrwxr-x	yahboo
dependent		文件夹	2024/06/15 17:53	drwxrwxr-x	yahboo
deep_sort		文件夹	2024/05/10 16:54	drwxrwxr-x	yahboo
coco128-images		文件夹	2024/05/10 16:54	drwxrwxr-x	yahboo
cfg		文件夹	2024/05/10 16:53	drwxrwxr-x	yahboo
.github		文件夹	2024/05/10 16:53	drwxrwxr-x	yahboo
__pycache__		文件夹	2024/06/15 11:21	drwxrwxr-x	yahboo
export.py	203 B		2024/05/10 16:54	-rw-rw-r--	yahboo
CITATION.cff	612 B		2024/05/10 16:53	-rw-rw-r--	yahboo
yolov8_track.py	928 B		2024/05/10 16:54	-rw-rw-r--	yahboo
requirements.txt	1.2 KB		2024/05/10 16:53	-rw-rw-r--	yahboo
.pre-commit-config.yaml	1.8 KB		2024/05/10 16:53	-rw-rw-r--	yahboo
.gitignore	2.2 KB		2024/05/10 16:54	-rw-rw-r--	yahboo
tracker_trt.py	2.9 KB		2024/05/10 16:53	-rw-rw-r--	yahboo
setup.py	3.1 KB		2024/05/10 16:54	-rw-rw-r--	yahboo
CONTRIBUTING.md	5.5 KB		2024/05/10 16:54	-rw-rw-r--	yahboo
.DS_Store	6 KB		2024/05/10 16:53	-rw-rw-r--	yahboo
multi_batch_inference.py	7.7 KB		2024/06/15 18:04	-rw-rw-r--	yahboo
multi_batch_inference_track.py	9.3 KB		2024/06/15 11:19	-rw-rw-r--	yahboo
mkdocs.yml	23.7 KB		2024/05/10 16:53	-rw-rw-r--	yahboo
LICENSE	33.7 KB		2024/05/10 16:53	-rw-rw-r--	yahboo
demo.mp4	15.5 MB		2024/05/10 16:54	-rw-rw-r--	yahboo

Weight file

Yolov8 official git file

Torchvision source code of jetson orin nx, needs to be compiled, but it has been compiled before leaving the factory

Model file

Some dependent packages of python environment

deep_sort source code

onnx_export program

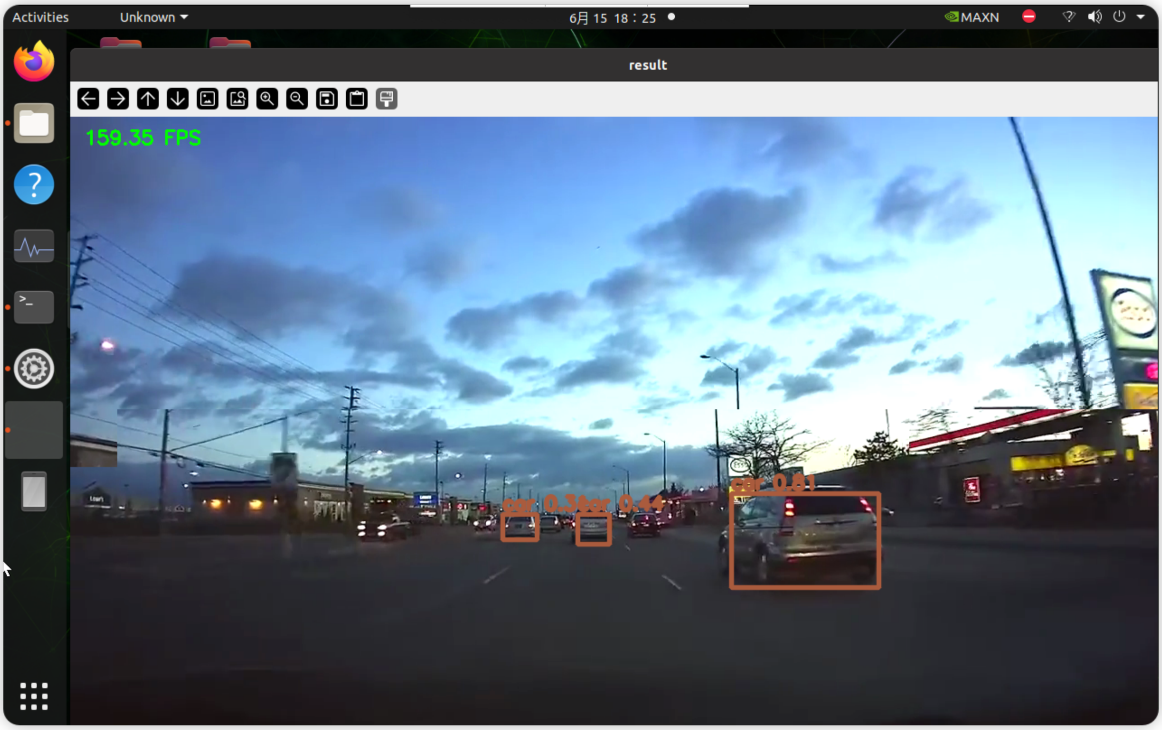
Yolov8 follower program

Yolov8 batch reasoning program

Yolov8 multi-batch reasoning tracking program

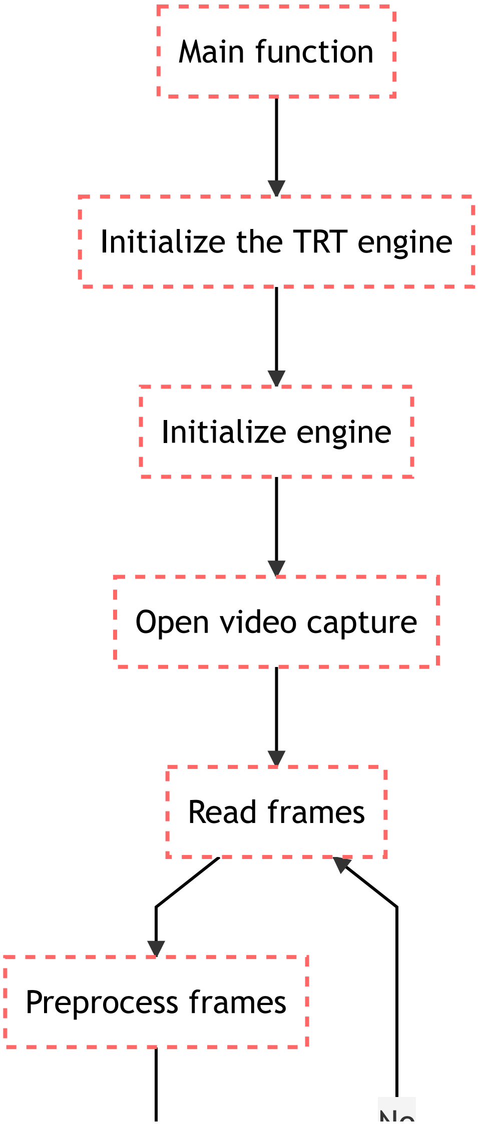
Run the demo command of target detection and execute the following code in the terminal

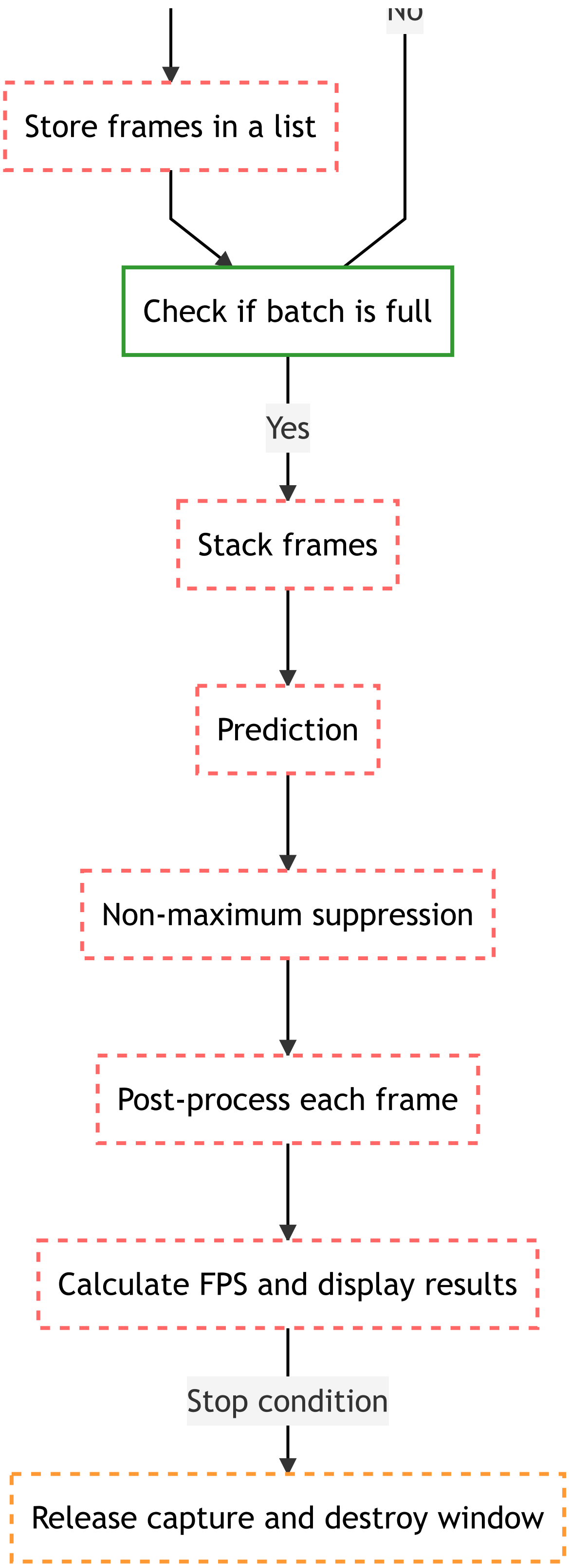
```
cd ~/YBAMR-COBOT-EDU-00001/soft/yolov8 && python multi_batch_inference.py
```



Code analysis

Code running flow chart:





Below is the code for our multi-batch object detection reasoning. The model accelerated by yolov8s' tenserrt is used.

```
import cv2
import numpy as np
from collections import OrderedDict, namedtuple
import time
import torch
```

```
from ultralytics.utils.ops import non_max_suppression, scale_boxes
import tensorrt as trt

names = ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light',
         'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
         'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
         'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard',
         'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
         'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',
         'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
         'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear',
         'hair drier', 'toothbrush']  # coco80 Category
colors = [[np.random.randint(0, 255) for _ in range(3)] for _ in names] ##Set random color


def letterbox(im, new_shape=(640, 640), color=(114, 114, 114), auto=False, scaleup=True, stride=32):
    # Resize and pad an image while satisfying stride multiple constraints
    shape = im.shape[:2]  # current shape [height, width]
    if isinstance(new_shape, int):
        new_shape = (new_shape, new_shape)

    # Scale ratio (new / old)
    r = min(new_shape[0] / shape[0], new_shape[1] / shape[1])
    if not scaleup:  # only scale down, do not scale up (for better val mAP)
        r = min(r, 1.0)

    # Compute padding
    new_unpad = int(round(shape[1] * r)), int(round(shape[0] * r))
    dw, dh = new_shape[1] - new_unpad[0], new_shape[0] - new_unpad[1]  # wh padding

    if auto:  # minimum rectangle
        dw, dh = np.mod(dw, stride), np.mod(dh, stride)  # wh padding

    dw /= 2  # divide padding into 2 sides
    dh /= 2

    if shape[::-1] != new_unpad:  # resize
        im = cv2.resize(im, new_unpad, interpolation=cv2.INTER_LINEAR)
    top, bottom = int(round(dh - 0.1)), int(round(dh + 0.1))
    left, right = int(round(dw - 0.1)), int(round(dw + 0.1))
    im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER_CONSTANT, value=color)  # add border
    # print(dw,dh)
    return im, r, (dw, dh)


class TRT_engine():
    def __init__(self, weight, thres=0.60, size=640, video_path='', batch_size=3) -> None:
        self.video_path = video_path
        self.imgsz = size
        self.weight = weight
        self.iou_thres = thres
        self.batch_size = batch_size
        self.device = torch.device('cuda:0')
        self.init_engine()

    def init_engine(self):
        # Infer TensorRT Engine
        self.Binding = namedtuple('Binding', ('name', 'dtype', 'shape', 'data', 'ptr'))
        self.logger = trt.Logger(trt.Logger.INFO)
        trt.init_libnvinfer_plugins(self.logger, namespace="")
        with open(self.weight, 'rb') as self.f, trt.Runtime(self.logger) as self.runtime:
            self.model = self.runtime.deserialize_cuda_engine(self.f.read())
        self.bindings = OrderedDict()
        print(f"num binding = {self.model.num_bindings}")
        for index in range(self.model.num_bindings):
            self.name = self.model.get_binding_name(index)
            print(f"name = {self.name}")
            self.dtype = trt.nptype(self.model.get_binding_dtype(index))
            self.shape = tuple(self.model.get_binding_shape(index))
            self.data = torch.from_numpy(np.empty(self.shape, dtype=np.dtype(self.dtype))).to(self.device)
            self.bindings[self.name] = self.Binding(self.name, self.dtype, self.shape, self.data,
                                                    int(self.data.data_ptr()))
        self.binding_addrs = OrderedDict((n, d.ptr) for n, d in self.bindings.items())
        self.context = self.model.create_execution_context()

    def predict(self, imgs):
        self.binding_addrs['images'] = int(imgs.data_ptr())

        self.context.execute_async_v2(list(self.binding_addrs.values()), torch.cuda.current_stream().cuda_stream)
        outputs = self.bindings['output0'].data  # Compare with the results output by onnx 393,409,425,441
        # print(outputs.shape)
        return outputs

    def process(self):
        cap = cv2.VideoCapture(self.video_path)
        # cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
        # cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
        img_list = []
        square_frame_list = []
        stop = False
        while cap.isOpened() and not stop:
            ret, frame = cap.read()
            if ret:
                frame_tensor, _, dw, dh = preprocess(frame, imgsz=self.imgsz)
                img_list.append(frame_tensor)
                square_frame_list.append(frame)
                if len(img_list) == self.batch_size:  # Used to store processed
frame images. When the queue is full, the images in the queue are concatenated
into a tensor.
                    frames = torch.stack(img_list, 0)
                    t1 = time.perf_counter()
                    outputs = self.predict(frames)
                    t2 = time.perf_counter()
                    infer_time = (t2 - t1) / self.batch_size
                    # print(outputs.shape)
                    outputs = non_max_suppression(outputs, 0.25, self.iou_thres,
classes=None, agnostic=False)
                    for i in range(self.batch_size):
                        t3 = time.perf_counter()
                        result = post_process(square_frame_list[i], outputs[i],
frames)
                        t4 = time.perf_counter()
                        fps = 1 / (infer_time + t4 - t3)
                        cv2.putText(result, f"{fps:.2f} FPS", (15, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
                        cv2.imshow("result", result)
                        if cv2.waitKey(1) & 0xFF == ord("q"):
                            stop = True
                            break
                    img_list = []
                    square_frame_list = []
            else:
                break
        cap.release()
        cv2.destroyAllWindows()


def preprocess(image, imgsz=640):
    img, ratio, (dw, dh) = letterbox(image, imgsz, stride=32,
                                     auto=False)  # When auto is FALSE, the
output image is 960*960, and when it is TRUE, it is resized proportionally.
    # The maximum length and width of the output image is 960
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # BGR to RGB
    img = img.transpose((2, 0, 1))
    img = np.ascontiguousarray(img)
    img = torch.from_numpy(img).to(torch.device('cuda:0')).float()
    img /= 255.0
    return img, image, dw, dh  # (3,640,640)


def post_process(img, det, frames):
    """
    Draw bounding boxes on the input image.
    """
    if len(det):
        # Rescale boxes from img_size to im0 size
        det[:, :4] = scale_boxes(frames.shape[2:], det[:, :4],
img.shape).round()
        for *xyxy, conf, cls in reversed(det):
            label = f'{names[int(cls)]} {conf:.2f}'
            p1, p2 = (int(xyxy[0]), int(xyxy[1])), (int(xyxy[2]), int(xyxy[3]))
            cv2.rectangle(img, p1, p2, colors[int(cls)], thickness=4,
lineType=cv2.LINE_AA)
```

```
            cv2.putText(img, label, (p1[0], p1[1] - 5),
cv2.FONT_HERSHEY_SIMPLEX,
                        0.7, colors[int(cls)], thickness=3,
lineType=cv2.LINE_AA)
    return img


if __name__ == '__main__':
    batch_size = 4 #Set the batch size to 4
    #trt_path = f"./weights/yolov8n-cal-batch{batch_size}-int8.trt"
    trt_path = f"./weights/yolov8s.trt" ##Model Path
    trt_engine = TRT_engine(trt_path, batch_size=batch_size, thres=0.45,
size=640, video_path="demo.mp4") ##Load the video used for inference
    trt_engine.process()
```