1. 得到模型文件

方法一：可以通过访问阿加犀模型广场：<https://aiot.aidlux.com/zh/models>

寻找需要的算法并进行下载即可，注意平台为Qualcomm QCS6490 ，这里举例使用yolov5模型格式转换



下载后通过可以通过ssh发送给开发板，可以使用xftp、MobaXterm等工具。

模型运行支持c++和python，这里我们使用python：首先更新aidlite，然后在解压后的一级目录中运行run\_test.py文件

sudo aid-pkg update

sudo aid-pkg install aidlite-sdk

# 确认当前可下载版本

aid-pkg list

# 安装 AidLite (latest QNN version)

sudo aid-pkg install aidlite-qnn231

# 运行yolo python示例

python3 python/run\_test.py

可以看到python文件夹下生成检测结果图



方法二：将自定义模型或者从模型官网下载的pt模型文件转化。

下载的pt格式一般是在英伟达环境加载，需要将其转为高通DSP硬件专属bin模型文件

1. 更新pip版本为25.1.1

python3.10 -m pip install --upgrade pip

pip -V

1. 安装ultralytics和onnx

pip install ultralytics onnx

1. 设置yolo命令的环境变量

echo 'export PATH="$PATH:$HOME/.local/bin"' >> ~/.bashrc

source ~/.bashrc # 使修改立即生效

验证：执行 yolo --version，若输出版本号（如 8.3.152），则说明命令已生效。

1. 将pt转换为onnx

新建python脚本，内容如下，并执行，可以在本文件夹生成onnx文件

#!/usr/bin/env python

from ultralytics import YOLO

# 加载同级目录下的.pt模型文件

model = YOLO('./yolov5s.pt')  # 替换为实际模型文件名

# 导出ONNX配置参数

export\_params = {

    'format': 'onnx',

    'opset': 12,          # 推荐算子集版本

    'simplify': True,     # 启用模型简化

    'dynamic': False,     # 固定输入尺寸

    'imgsz': 640,         # 标准输入尺寸

    'half': False         # 保持FP32精度

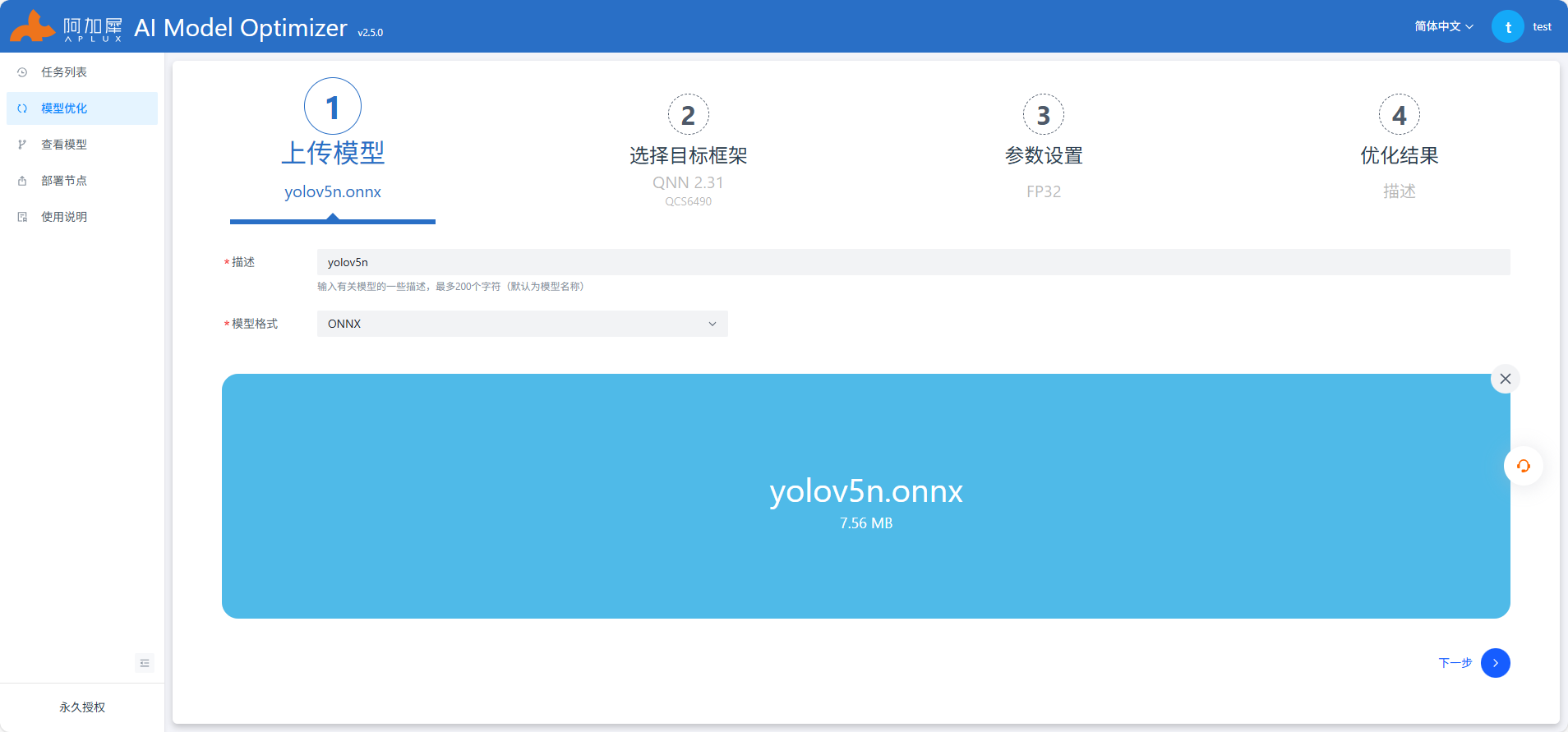
}

# 执行转换并保存到同级目录

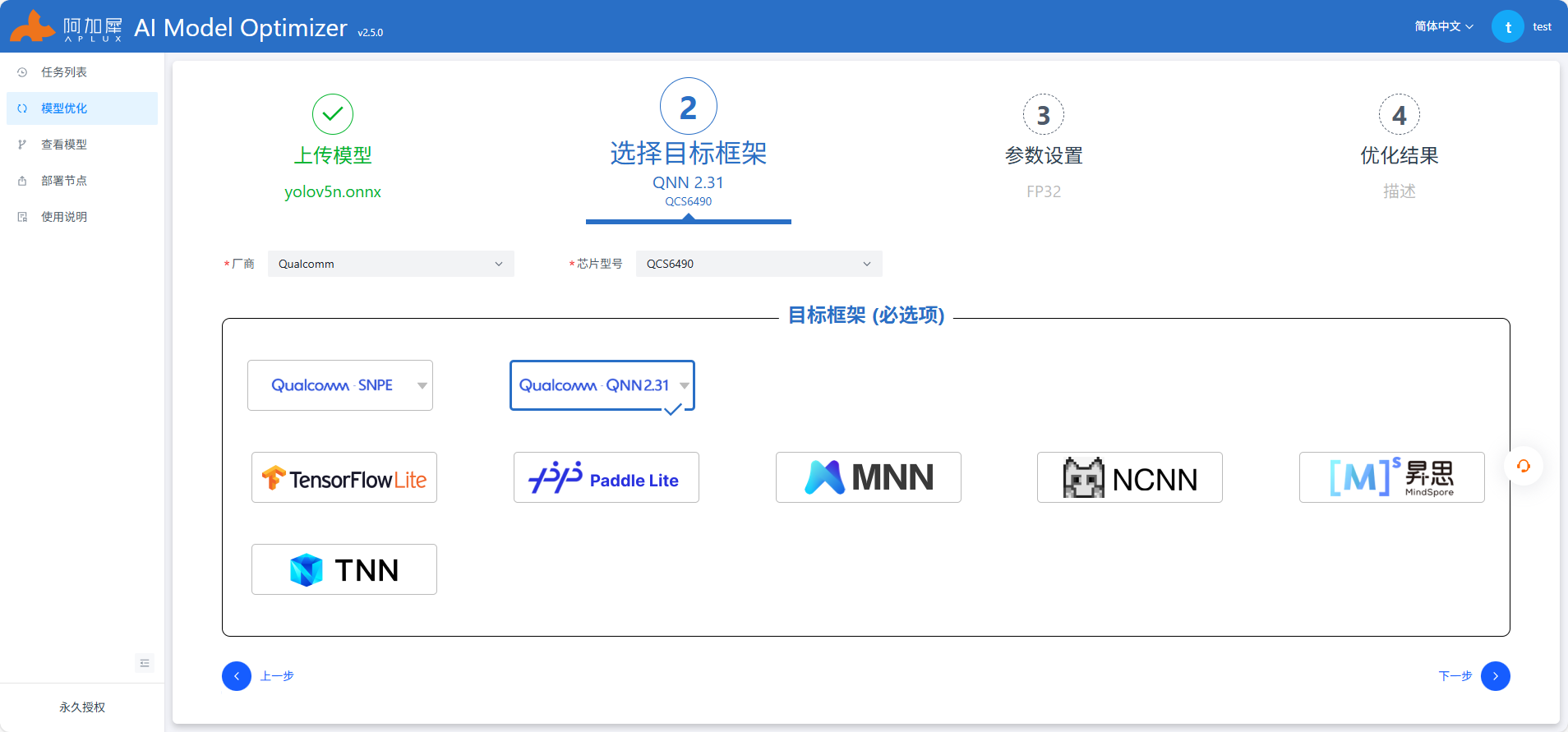
model.export(\*\*export\_params)

1. 进入[AI Model Optimizer](https://aimo.aidlux.com/" \l "/model-convert)，进行ONNX格式转换为QNN支持的格式

Step1：选择模型优化，模型格式选择onnx格式上传模型



Step2：选择芯片型号以及目标框架，这里我们选择QCS6490+Qnn2.31

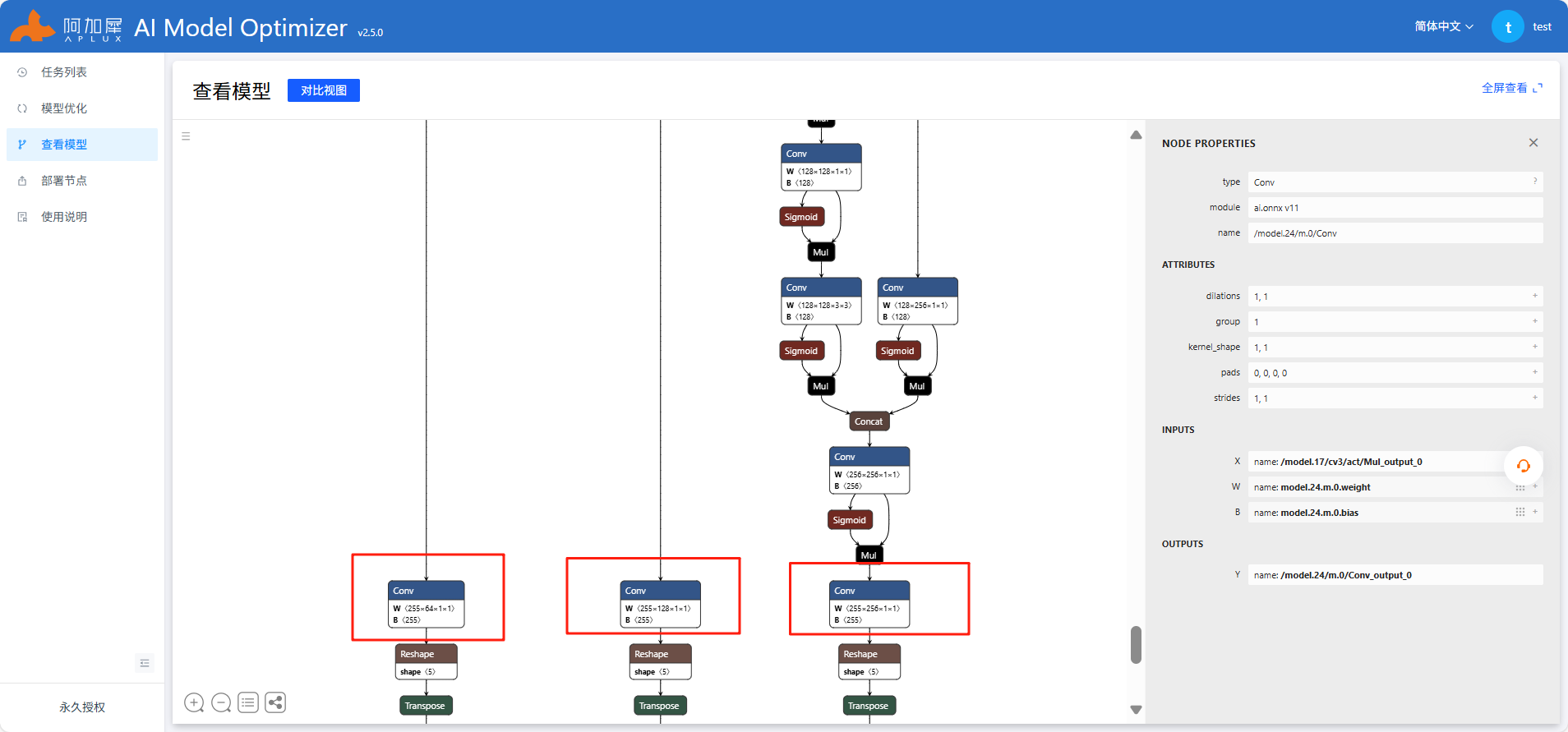


Step3：点击查看模型，使用Netron查看模型结构，进行输入输出的填写，使用Netron工具查看onnx模型结构，选择剪枝位置

/model.24/m.0/Conv\_output\_0

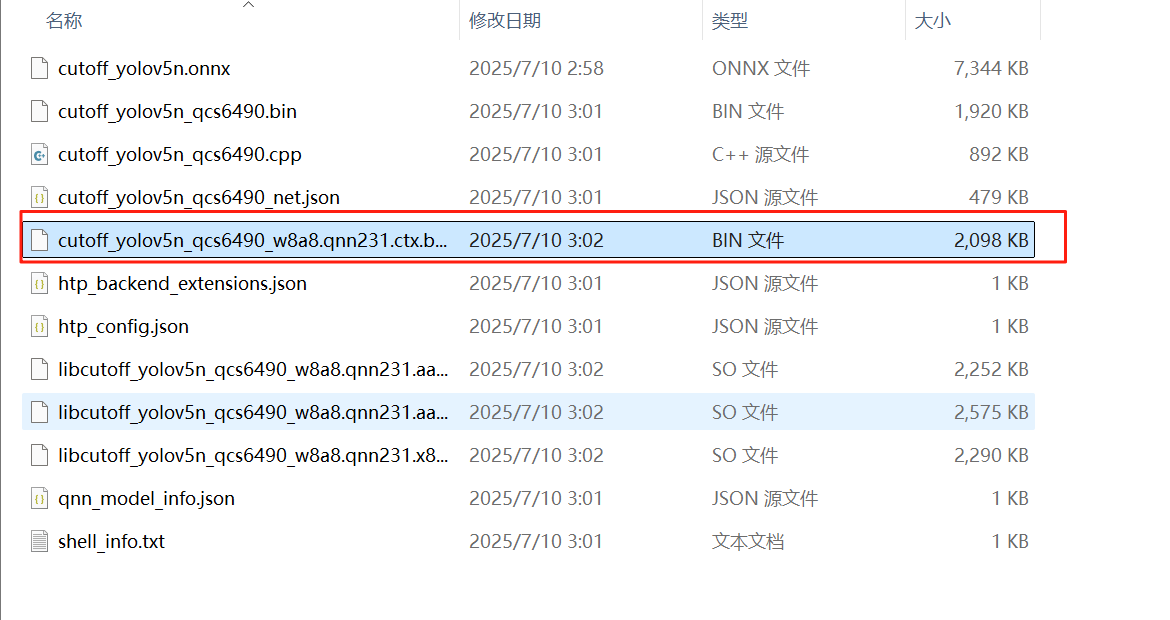
/model.24/m.1/Conv\_output\_0

/model.24/m.2/Conv\_output\_0



参考上图中红色框部分填写，其他不变，注意开启自动量化功能，AIMO更多操作查看使用说明或参考[AIMO平台](https://docs.aidlux.com/guide/software/ai-platform-portal-aimo)

Step4：接下来进行提交即可，转换完成后将目标模型文件下载，解压缩后其中的.bin.aidem文件即为模型文件



QNN测试代码

该代码实现了基于aidlite库的 YOLOv5 目标检测流程，主要功能包括：

命令行参数解析（模型路径、图像路径、推理次数等）；

图像预处理（等比例缩放、填充、归一化）；

使用aidlite加载模型并执行推理（支持 QNN/SNPE 框架，DSP 加速）；

推理性能统计（耗时均值、最大值、最小值、方差）；

检测结果后处理（置信度筛选、非极大值抑制 NMS、坐标映射）；

可视化检测结果（绘制边界框和类别标签，并保存图像）。

#!/usr/bin/env python3

# import sys

import os   #os模块提供了与操作系统进行交互的功能。

import sys  #sys模块提供了与Python解释器进行交互的函数。

import time

import aidlite

import cv2

import numpy as np

OBJ\_CLASS\_NUM = 80  #COCO数据集的类别数

NMS\_THRESH = 0.45   #非极大值抑制的阈值

BOX\_THRESH = 0.5    #检测框的置信度阈值

MODEL\_SIZE = 640    #模型输入尺寸

OBJ\_NUMB\_MAX\_SIZE = 64 #最大目标检测数

PROP\_BOX\_SIZE = 5 + OBJ\_CLASS\_NUM #每个预测框的输出维度：5 表示 (x, y, w, h, confidence)，加上 80 个类别的概率

STRIDE8\_SIZE = MODEL\_SIZE / 8 #特征图下采样步长为 8 的特征图大小（即 640/8=80）

STRIDE16\_SIZE = MODEL\_SIZE / 16

STRIDE32\_SIZE = MODEL\_SIZE / 32

#定义了三组不同尺度的锚框（anchor boxes），每组包含 3 个锚框。这些锚框用于生成候选框。

anchors = [

    [10, 13, 16, 30, 33, 23],

    [30, 61, 62, 45, 59, 119],

    [116, 90, 156, 198, 373, 326],

]

#COCO 数据集中所有目标类别的名称列表，共 80 个类别。

coco\_class = [

    "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",

]

#将边界框从中心坐标 (cx, cy, w, h) 转换为左上角和右下角坐标 (x1, y1, x2, y2)

def eqprocess(image, size1, size2):

    h, w, \_ = image.shape

    mask = np.zeros((size1, size2, 3), dtype=np.float32)

    scale1 = h / size1

    scale2 = w / size2

    if scale1 > scale2:

        scale = scale1

    else:

        scale = scale2

    img = cv2.resize(image, (int(w / scale), int(h / scale)))

    mask[: int(h / scale), : int(w / scale), :] = img

    return mask, scale

#同上，描述该函数的作用是将边界框格式转换。

def xywh2xyxy(x):

    """

    Box (center x, center y, width, height) to (x1, y1, x2, y2)

    """

    y = np.copy(x)

    y[:, 0] = x[:, 0] - x[:, 2] / 2  # top left x

    y[:, 1] = x[:, 1] - x[:, 3] / 2  # top left y

    y[:, 2] = x[:, 0] + x[:, 2] / 2  # bottom right x

    y[:, 3] = x[:, 1] + x[:, 3] / 2  # bottom right y

    return y

#描述该函数将边界框从 (x1, y1, x2, y2) 格式转换为 (x1, y1, w, h) 格式

def xyxy2xywh(box):

    """

    Box (left\_top x, left\_top y, right\_bottom x, right\_bottom y) to (left\_top x, left\_top y, width, height)

    """

    box[:, 2:] = box[:, 2:] - box[:, :2]

    return box

#描述这是一个针对单类的 NMS（非极大值抑制） 算法，用于去除冗余的边界框。

def NMS(dets, scores, thresh):

    """

    单类NMS算法

    dets.shape = (N, 5), (left\_top x, left\_top y, right\_bottom x, right\_bottom y, Scores)

    """

    x1 = dets[:, 0]

    y1 = dets[:, 1]

    x2 = dets[:, 2]

    y2 = dets[:, 3]

    areas = (y2 - y1 + 1) \* (x2 - x1 + 1)#计算面积

    keep = []

    index = scores.argsort()[::-1]  # 按置信度从高到低排序

    while index.size > 0:

        i = index[0]  # every time the first is the biggst, and add it directly

        keep.append(i)

        x11 = np.maximum(x1[i], x1[index[1:]])  # calculate the points of overlap

        y11 = np.maximum(y1[i], y1[index[1:]])

        x22 = np.minimum(x2[i], x2[index[1:]])

        y22 = np.minimum(y2[i], y2[index[1:]])

        w = np.maximum(0, x22 - x11 + 1)  # the weights of overlap

        h = np.maximum(0, y22 - y11 + 1)  # the height of overlap

        overlaps = w \* h

        ious = overlaps / (areas[i] + areas[index[1:]] - overlaps)

        idx = np.where(ious <= thresh)[0]

        index = index[idx + 1]  # because index start from 1

    return keep

#描述该函数将边界框裁剪到图像范围内，防止越界。

def clip\_coords(boxes, img\_shape):

    # Clip bounding xyxy bounding boxes to image shape (height, width)

    boxes[:, 0].clip(0, img\_shape[1], out=boxes[:, 0])  # x1

    boxes[:, 1].clip(0, img\_shape[0], out=boxes[:, 1])  # y1

    boxes[:, 2].clip(0, img\_shape[1], out=boxes[:, 2])  # x2

    boxes[:, 3].clip(0, img\_shape[0], out=boxes[:, 3])  # y2

#描述该函数对模型输出进行后处理，包括筛选高置信度框、NMS 去重等操作。

def detect\_postprocess(

    prediction, img0shape, img1shape, conf\_thres=0.25, iou\_thres=0.45

):

    """

    检测输出后处理

    prediction: aidlite模型预测输出

    img0shape: 原始图片shape

    img1shape: 输入图片shape

    conf\_thres: 置信度阈值

    iou\_thres: IOU阈值

    return: list[np.ndarray(N, 5)], 对应类别的坐标框信息, xywh、conf

    """

    h, w, \_ = img1shape

    valid\_condidates = prediction[prediction[..., 4] > conf\_thres]

    valid\_condidates[:, 5:] \*= valid\_condidates[:, 4:5]

    valid\_condidates[:, :4] = xywh2xyxy(valid\_condidates[:, :4])

    max\_det = 300

    max\_wh = 7680

    max\_nms = 30000

    valid\_condidates[:, 4] = valid\_condidates[:, 5:].max(1)

    valid\_condidates[:, 5] = valid\_condidates[:, 5:].argmax(1)

    sort\_id = np.argsort(valid\_condidates[:, 4])[::-1]

    valid\_condidates = valid\_condidates[sort\_id[:max\_nms]]

    boxes, scores = (

        valid\_condidates[:, :4] + valid\_condidates[:, 5:6] \* max\_wh,

        valid\_condidates[:, 4],

    )

    index = NMS(boxes, scores, iou\_thres)[:max\_det]

    out\_boxes = valid\_condidates[index]

    clip\_coords(out\_boxes[:, :4], img0shape)

    out\_boxes[:, :4] = xyxy2xywh(out\_boxes[:, :4])

    print("检测到{}个区域".format(len(out\_boxes)))

    return out\_boxes

#描述该函数用于在图像上绘制检测结果，包括边界框和类别标签。

def draw\_detect\_res(img, det\_pred):

    """

    检测结果绘制

    """

    img = img.astype(np.uint8)

    color\_step = int(255 / len(coco\_class))

    for i in range(len(det\_pred)):

        x1, y1, x2, y2 = [int(t) for t in det\_pred[i][:4]]

        score = det\_pred[i][4]

        cls\_id = int(det\_pred[i][5])

        print(i + 1, [x1, y1, x2 + x1, y2 + y1], score, coco\_class[cls\_id])

        cv2.putText(

            img,

            f"{coco\_class[cls\_id]}",

            (x1, y1 - 6),

            cv2.FONT\_HERSHEY\_SIMPLEX,

            0.5,

            (255, 255, 255),

            1,

        )

        cv2.rectangle(

            img,

            (x1, y1),

            (x2 + x1, y2 + y1),

            (0, int(cls\_id \* color\_step), int(255 - cls\_id \* color\_step)),

            thickness=10,

        )

    return img

#描述该类是 YOLOv5 的检测头，用于处理模型输出并生成最终的检测结果。

class Detect:

    # YOLOv5 Detect head for detection models

    def \_\_init\_\_(self, nc=80, anchors=(), stride=[], image\_size=640):  # detection layer

        super().\_\_init\_\_()

        self.nc = nc  # number of classes

        self.no = nc + 5  # number of outputs per anchor

        self.stride = stride

        self.nl = len(anchors)  # number of detection layers

        self.na = len(anchors[0]) // 2  # number of anchors

        self.grid, self.anchor\_grid = [0] \* self.nl, [0] \* self.nl

        self.anchors = np.array(anchors, dtype=np.float32).reshape(self.nl, -1, 2)

        base\_scale = image\_size // 8

        for i in range(self.nl):

            self.grid[i], self.anchor\_grid[i] = self.\_make\_grid(

                base\_scale // (2\*\*i), base\_scale // (2\*\*i), i

            )

    def \_make\_grid(self, nx=20, ny=20, i=0):

        y, x = np.arange(ny, dtype=np.float32), np.arange(nx, dtype=np.float32)

        yv, xv = np.meshgrid(y, x)

        yv, xv = yv.T, xv.T

        # add grid offset, i.e. y = 2.0 \* x - 0.5

        grid = np.stack((xv, yv), 2)

        grid = grid[np.newaxis, np.newaxis, ...]

        grid = np.repeat(grid, self.na, axis=1) - 0.5

        anchor\_grid = self.anchors[i].reshape((1, self.na, 1, 1, 2))

        anchor\_grid = np.repeat(anchor\_grid, repeats=ny, axis=2)

        anchor\_grid = np.repeat(anchor\_grid, repeats=nx, axis=3)

        return grid, anchor\_grid

    def sigmoid(self, arr):

        return 1 / (1 + np.exp(-arr))

    def \_\_call\_\_(self, x):

        z = []  # inference output

        for i in range(self.nl):

            bs, \_, ny, nx = x[i].shape

            x[i] = x[i].reshape(bs, self.na, self.no, ny, nx).transpose(0, 1, 3, 4, 2)

            y = self.sigmoid(x[i])

            y[..., 0:2] = (y[..., 0:2] \* 2.0 + self.grid[i]) \* self.stride[i]  # xy

            y[..., 2:4] = (y[..., 2:4] \* 2) \*\* 2 \* self.anchor\_grid[i]  # wh

            z.append(y.reshape(bs, self.na \* nx \* ny, self.no))

        return np.concatenate(z, 1)

def show\_help\_prompt():

    print("python3 ./qnn\_yolov5\_multi.py 3 Run on the DSP")

    return False

def main():

    argvs = len(sys.argv)

    if argvs < 2:

        return show\_help\_prompt()

    acc\_type = int(sys.argv[1])

    aidlite.set\_log\_level(aidlite.LogLevel.INFO)

    aidlite.log\_to\_stderr()

    print(f"Aidlite library version : {aidlite.get\_library\_version()}")

    print(f"Aidlite python library version : {aidlite.get\_py\_library\_version()}")

    config = aidlite.Config.create\_instance()

    if config is None:

        print("Create config failed !")

        return False

    model\_name = "cutoff\_yolov5s\_sigmoid\_w8a8.qnn229.ctx.bin"

    #model\_name = "cutoff\_yolov8x\_qcs6490\_w8a8.qnn231.ctx.bin"       #//!模型选择

    if acc\_type == 1:

        config.accelerate\_type = aidlite.AccelerateType.TYPE\_CPU

    elif acc\_type == 2:

        config.accelerate\_type = aidlite.AccelerateType.TYPE\_GPU

        config.is\_quantify\_model = 0

    elif acc\_type == 3:

        config.accelerate\_type = aidlite.AccelerateType.TYPE\_DSP    #//!选择DSP方式

        config.is\_quantify\_model = 1

    else:

        return show\_help\_prompt()

    config.implement\_type = aidlite.ImplementType.TYPE\_LOCAL

    config.framework\_type = aidlite.FrameworkType.TYPE\_QNN229       #//!选择QNN229方式

    # model\_path = os.path.join(

    #     resource\_dir, "cutoff\_yolov5s\_640\_sigmoid\_int8.serialized.bin"

    # )

    resource\_dir = '/home/aidlux/yolov5/data/qnn\_yolov5\_multi/'

    model\_path = os.path.join(resource\_dir, model\_name) #加载模型

    model = aidlite.Model.create\_instance(model\_path)

    if model is None:

        print("Create model failed !")

        return False

    #输入张量，[每次推理图像张数,图像高，图像宽，图像通道数]

    input\_shapes = [[1, MODEL\_SIZE, MODEL\_SIZE, 3]]

    #三个输出层（feature map），分别对应不同尺度的目标检测[每次推理图像张数,特征图高，特征图宽，输出纬度]

    #每个锚框预测 (4 + 1 + 80) = 85 个值（xywh + confidence + class probs），每层有 3 个 anchor box → 3 × 85 = 255

    #三种输出，分别是检测小目标（深层特征）、中目标（中等深度特征）、大目标（浅层特征）

    output\_shapes = [[1, 20, 20, 255], [1, 40, 40, 255], [1, 80, 80, 255]]

    model.set\_model\_properties(

        input\_shapes,

        aidlite.DataType.TYPE\_FLOAT32,

        output\_shapes,

        aidlite.DataType.TYPE\_FLOAT32,

    )

    interpreter = aidlite.InterpreterBuilder.build\_interpretper\_from\_model\_and\_config(

        model, config

    )

    if interpreter is None:

        print("build\_interpretper\_from\_model\_and\_config failed !")

        return None

    result = interpreter.init()

    if result != 0:

        print("interpreter init failed !")

        return False

    result = interpreter.load\_model()

    if result != 0:

        print("interpreter load model failed !")

        return False

    print("detect model load success!")

    input\_tensor\_info = interpreter.get\_input\_tensor\_info()

    if len(input\_tensor\_info) == 0 :

        printf("interpreter get\_input\_tensor\_info() failed !\n")

        return False

    for gi, graph\_tensor\_info in enumerate(input\_tensor\_info):

        for ti, tensor\_info in enumerate(graph\_tensor\_info):

            print(  f"Input  tensor : Graph[{gi}]-Tensor[{ti}]-name[{tensor\_info.name}]"

                    f"-element\_count[{tensor\_info.element\_count}]-element\_type[{tensor\_info.element\_type}]"

                    f"-dimensions[{tensor\_info.dimensions}]-shape{tensor\_info.shape}")

    output\_tensor\_info = interpreter.get\_output\_tensor\_info()

    if len(output\_tensor\_info) == 0 :

        printf("interpreter get\_output\_tensor\_info() failed !\n")

        return False

    for gi, graph\_tensor\_info in enumerate(output\_tensor\_info):

        for ti, tensor\_info in enumerate(graph\_tensor\_info):

            print(  f"Output tensor : Graph[{gi}]-Tensor[{ti}]-name[{tensor\_info.name}]"

                    f"-element\_count[{tensor\_info.element\_count}]-element\_type[{tensor\_info.element\_type}]"

                    f"-dimensions[{tensor\_info.dimensions}]-shape{tensor\_info.shape}")

    stride8 = stride16 = stride32 = None

    image\_path = os.path.join(resource\_dir, "image.jpg")

    frame = cv2.imread(image\_path)

    frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    img\_input, scale = eqprocess(frame, MODEL\_SIZE, MODEL\_SIZE)

    img\_input = img\_input / 255

    img\_input = img\_input.astype(np.float32)

    sum\_time\_0 = 0.0

    sum\_time\_1 = 0.0

    sum\_time\_2 = 0.0

    \_counter = 10

    for idx in range(\_counter):

        st0 = time.time()

        input\_tensor\_data = img\_input.data

        # result = interpreter.set\_input\_tensor(0, input\_tensor\_data)

        result = interpreter.set\_input\_tensor("images", input\_tensor\_data)

        if result != 0:

            print("interpreter set\_input\_tensor() failed")

            return False

        et0 = time.time()

        dur0 = et0 - st0

        sum\_time\_0 += dur0

        print(f"current [{idx}] set\_input\_tensor cost time :{dur0} ms")

        st1 = time.time()

        result = interpreter.invoke()   #执行推理

        if result != 0:

            print("interpreter set\_input\_tensor() failed")

            return False

        et1 = time.time()

        dur1 = et1 - st1

        sum\_time\_1 += dur1

        print(f"current [{idx}] invoke cost time :{dur1} ms")

        st2 = time.time()

        # stride8 = interpreter.get\_output\_tensor(0)

        stride8 = interpreter.get\_output\_tensor("\_326") #获取不同层级的输出张量（如 stride8、stride16、stride32）

        if stride8 is None:

            print("sample : interpreter->get\_output\_tensor() 0 failed !")

            return False

        print(f"len(stride8 {len(stride8)}")

        # stride16 = interpreter.get\_output\_tensor(1)

        stride16 = interpreter.get\_output\_tensor("\_364")

        if stride16 is None:

            print("sample : interpreter->get\_output\_tensor() 1 failed !")

            return False

        print(f"len(stride16 {len(stride16)}")

        # stride32 = interpreter.get\_output\_tensor(2)

        stride32 = interpreter.get\_output\_tensor("\_402")

        if stride32 is None:

            print("sample : interpreter->get\_output\_tensor() 2 failed !")

            return False

        print(f"len(stride32 {len(stride32)}")

        et2 = time.time()

        dur2 = et2 - st2

        sum\_time\_2 += dur2

        print(f"current [{idx}] get\_output\_tensor cost time :{dur2} ms")

    print(

        f"repeat [{\_counter}] times , input[{sum\_time\_0 \* 1000}]ms --- invoke[{sum\_time\_1 \* 1000}]ms --- output[{sum\_time\_2 \* 1000}]ms --- sum[{(sum\_time\_0 + sum\_time\_1 + sum\_time\_2) \* 1000}]ms"

    )

    stride = [8, 16, 32]

    yolo\_head = Detect(OBJ\_CLASS\_NUM, anchors, stride, MODEL\_SIZE)

    # 后处理部分reshape需要知道模型的output\_shapes

    validCount0 = stride8.reshape(\*output\_shapes[2]).transpose(0, 3, 1, 2)

    validCount1 = stride16.reshape(\*output\_shapes[1]).transpose(0, 3, 1, 2)

    validCount2 = stride32.reshape(\*output\_shapes[0]).transpose(0, 3, 1, 2)

    #使用 Detect 类对模型输出进行解码，得到最终的检测结果。

    pred = yolo\_head([validCount0, validCount1, validCount2])

    #对模型输出进行后处理，得到可用的边界框和类别信息。

    det\_pred = detect\_postprocess(

        pred, frame.shape, [MODEL\_SIZE, MODEL\_SIZE, 3], conf\_thres=0.5, iou\_thres=0.45

    )

    #//?根据置信度最大取出best\_person

    # 取出所有person

    person\_det = det\_pred[det\_pred[:, 5] == 0]

    det\_pred = person\_det

    # 输出置信度最高的中心像素坐标（改为：面积最大的框）

    if len(person\_det) > 0:

         # 先将 xywh 转换为 xyxy

        boxes\_xyxy = xywh2xyxy(person\_det[:, :4])

         # 提取 x1, y1, x2, y2

        x1 = boxes\_xyxy[:, 0]

        y1 = boxes\_xyxy[:, 1]

        x2 = boxes\_xyxy[:, 2]

        y2 = boxes\_xyxy[:, 3]

        areas = (x2 - x1) \* (y2 - y1)

        # 找到面积最大的框的索引

        best\_index = np.argmax(areas)

        # 取出面积最大的person框

        best\_person = person\_det[[best\_index]]  # 使用 [[index]] 保持二维数组结构

        det\_pred = best\_person

    else:

        print("未检测到 person")

    det\_pred[np.isnan(det\_pred)] = 0.0

    det\_pred[:, :4] = det\_pred[:, :4] \* scale

    # 提取坐标并计算中心点

    x, y, w, h = det\_pred[0, :4].astype(int)    #x,y左上角点，w,h宽高

    center\_x = x + w// 2

    center\_y = y + h // 2

    print(f"Person 中心坐标: ({center\_x}, {center\_y})")

    print(f"x, y, w, h: {x, y, w, h}\n\n\n\n\n\n\n")

    #在原始图像上绘制检测结果

    res\_img = draw\_detect\_res(frame, det\_pred)

    #画出中心点画一个点

    cv2.circle(res\_img, (center\_x, center\_y), radius=5, color=(0, 0, 255), thickness=20)

    result = interpreter.destory()

    if result != 0:

        print("interpreter set\_input\_tensor() failed")

        return False

    frame\_bgr = cv2.cvtColor(res\_img, cv2.COLOR\_RGB2BGR)

    result\_img\_path = f"{os.path.splitext(os.path.abspath(\_\_file\_\_))[0]}.jpg"

    cv2.imwrite(result\_img\_path, frame\_bgr)

    print(f"The result image has been saved to : {result\_img\_path}")

    return True

if \_\_name\_\_ == "\_\_main\_\_":

    main()

执行python your\_code.py 3

输出图像：

