目录

[一、 CPipe框架介绍 1](#_Toc26755)

[1. 框架优势 2](#_Toc22883)

[2. CPipe安装方法: 2](#_Toc27873)

[3. Node节点可视化: 3](#_Toc7996)

[4. 算法结果实时可视化: 3](#_Toc9240)

[5. 视频流节点添加/编辑/删除: 3](#_Toc23145)

[6. 算法检测区域添加: 4](#_Toc24249)

[7. License授权操作: 5](#_Toc14898)

[8. 支持算法: 5](#_Toc5111)

[二、 视频流节点类 6](#_Toc4552)

[三、 算法推理节点类 7](#_Toc18407)

[四、 结果可视化节点类 8](#_Toc25605)

[五、 日志使用方法 9](#_Toc11434)

[六、 结果上报节点类 11](#_Toc12912)

[七、 CPipe配置文件说明 11](#_Toc7661)

[1. CPipe配置文件 11](#_Toc20797)

[2. Yaml配置文件解析 13](#_Toc20836)

[八、 自定义可视化功能 13](#_Toc20569)

[1. Web文本显示功能: 13](#_Toc11309)

[2. 自定义可视化功能: 13](#_Toc18882)

[九、 模型文件加密 15](#_Toc27713)

[十、 CPipe常用数据类 16](#_Toc10173)

[1. Box类: 16](#_Toc3665)

[2. CData类: 17](#_Toc18858)

[3. CImage类: 18](#_Toc23502)

[十一、 工程案例: 19](#_Toc25733)

[1. 自定义节点类: 19](#_Toc10730)

[2. 部分逻辑代码块-得分点: 20](#_Toc9296)

[3. 部分逻辑代码块-得分逻辑判断: 20](#_Toc22988)

[4. 整体逻辑代码: 22](#_Toc32257)

[5. 初始化整个工程: 23](#_Toc27479)

[6. 可视化页面展示: 24](#_Toc23974)

# CPipe框架介绍

CPipe是基于Python编写的AI视觉算法快速部署框架.本框架基于Node思想,将所有视频流/视频文件/AI算法/上报信息以及逻辑代码块等抽象成一个Node节点.节点之间可以自由连接.

CPipe框架功能:

1. 支持Node节点网页可视化;
2. 支持算法推理结果网页实时可视化;
3. 支持一键快速部署
4. 支持GPU视频解码加速
5. 支持页面添加/删除视频流节点
6. 支持页面配置算法区域功能
7. 支持AI模型格式TensorRT/ONNX/PyTorch等
8. 支持各类AI算法(目标检测/旋转目标检测/分类/人脸识别/人脸质量评估/关键点检测/人流统计/目标跟踪/REID/视频时序分类等)
9. 支持视频类型:RTSP/RTMP/本地视频文件/本地图片
10. 支持日志本地存储/日志云端上报/日志网页实时显示
11. 支持AI模型文件加密功能



## 框架优势

|  |  |  |
| --- | --- | --- |
| **内容** | **使用前** | **使用CPipe+训练平台** |
| **算法工程师经验** | 3年以上 | 1年以上 |
| **开发周期缩减** | 无 | 缩减80%以上 |
| **算法部署环境** | 需要搭建 | 框架提供Docker镜像 |
| **算法硬件加速** | 需要自己编写代码 | 框架自带 |
| **视频编解码加速** | 需要自己编译相关硬件库 | 框架自带 |
| **视频流批量推理** | 需要编写相关并发代码 | 框架自带 |
| **一键无代码部署** | 无 | 框架自带 |
| **算法性能可视化** | 无 | 框架自带 |
| **算法结果实时Web可视化** | 需要前端工程师参与编写代码 | 框架自带 |
| **默认算法支持** | 无 | 框架自带十几种算法 |
| **算法文件加密** | 需要编写加密程序 | 框架自带 |
| **日志** | 需要编写日志程序 | 框架自带 |

价值:

降低研发成本/减少开发周期/提供稳定且高性能的AI算法推理引擎/提供高并发视频流实时算法推理/提供算法模型机代码加密安全

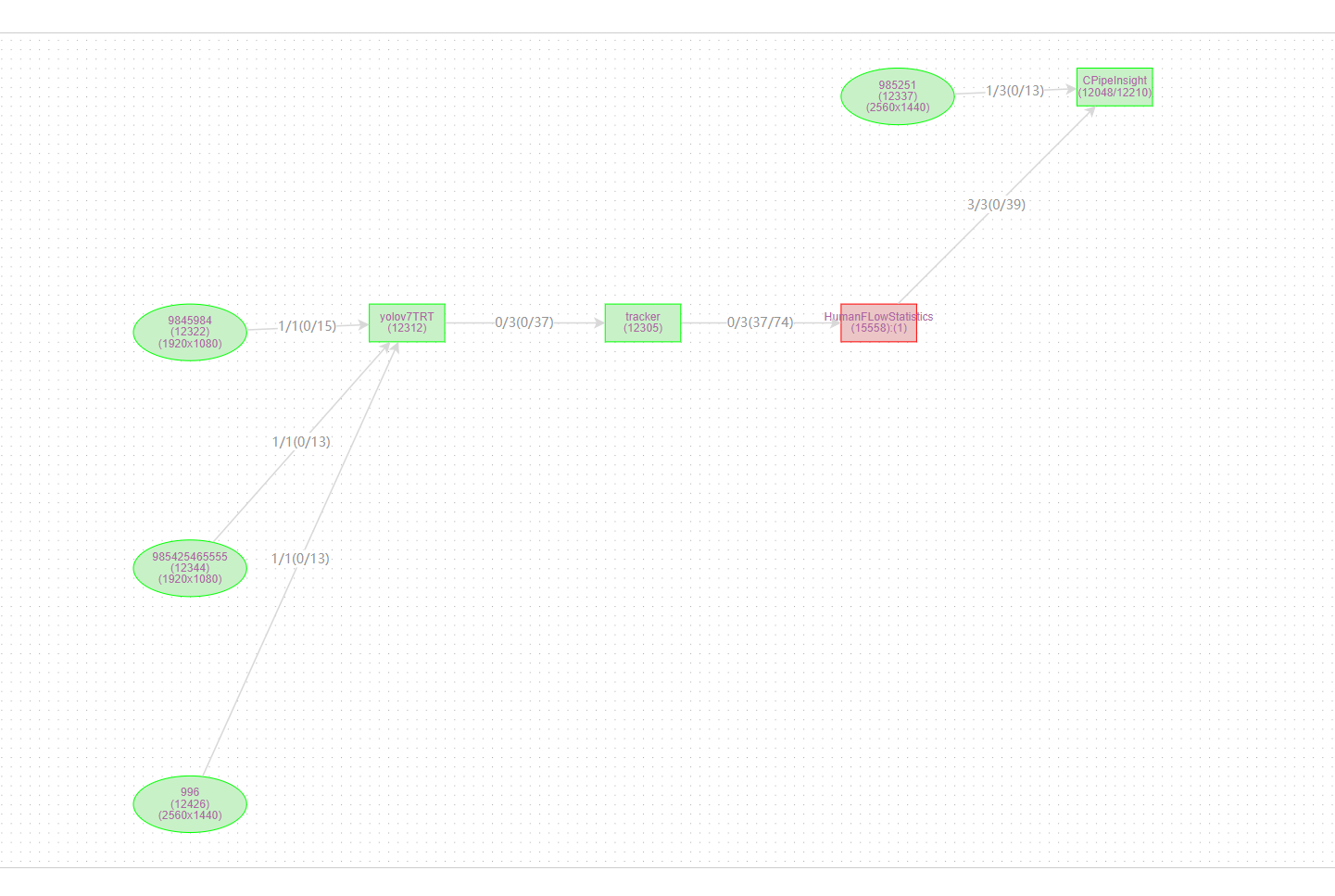
## CPipe安装方法:

采用pip安装指令,支持Python 3.8及以上,并提供相关Docker镜像.

例子:

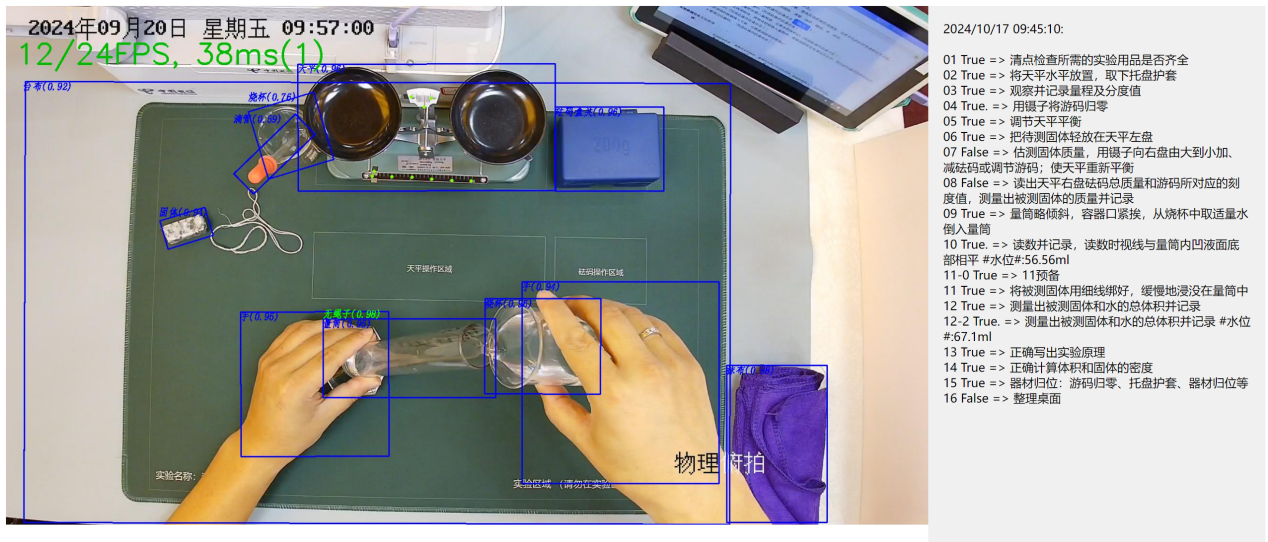
pip install cpipe-3.0.0-cp39-cp39-linux\_x86\_64.whl

## Node节点可视化:



支持显示:节点连接关系、节点运行状态、节点间队列运行状态、视频抛帧情况.

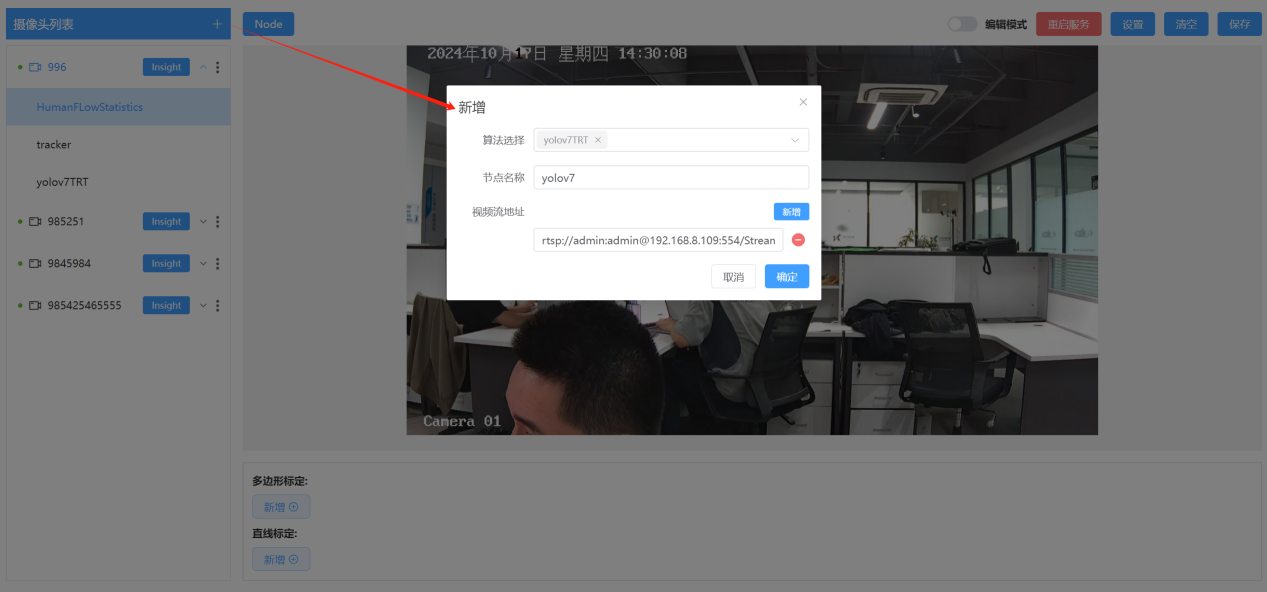
## 算法结果实时可视化:



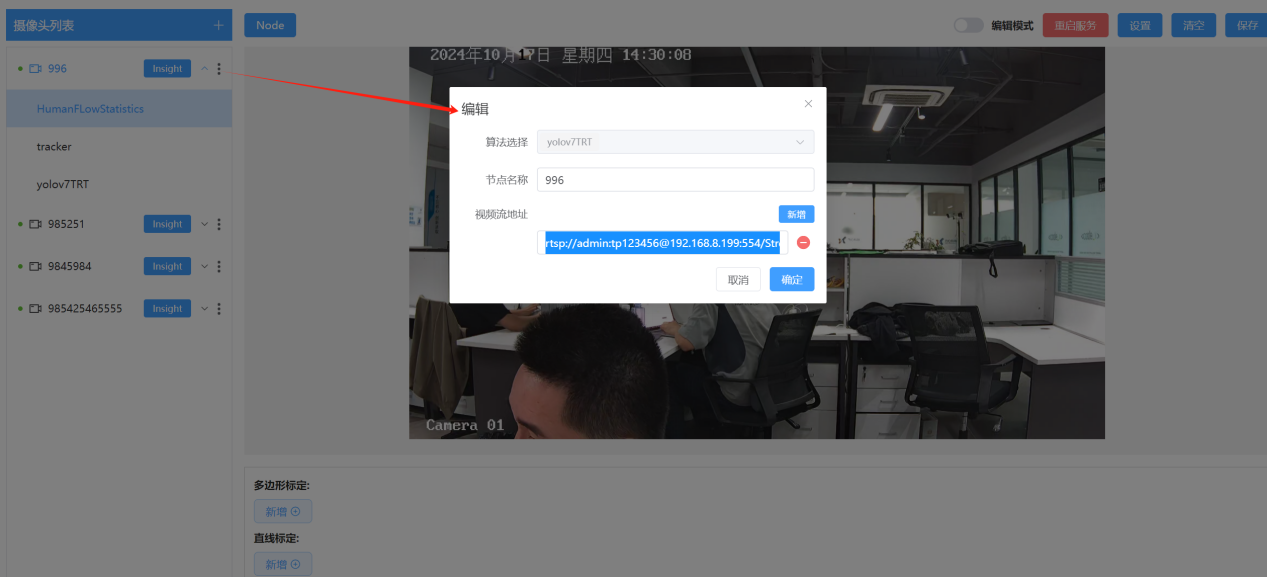
算法实时效果显示以及实时日志显示

## 视频流节点添加/编辑/删除:

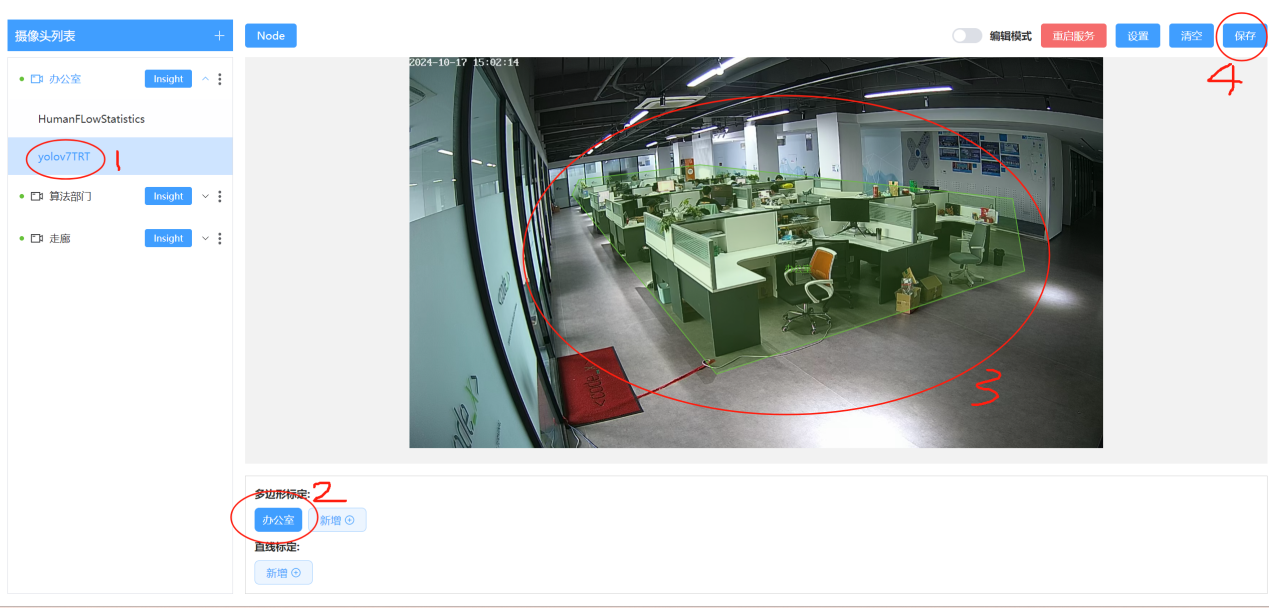
新增视频流节点:



编辑/删除视频流节点:

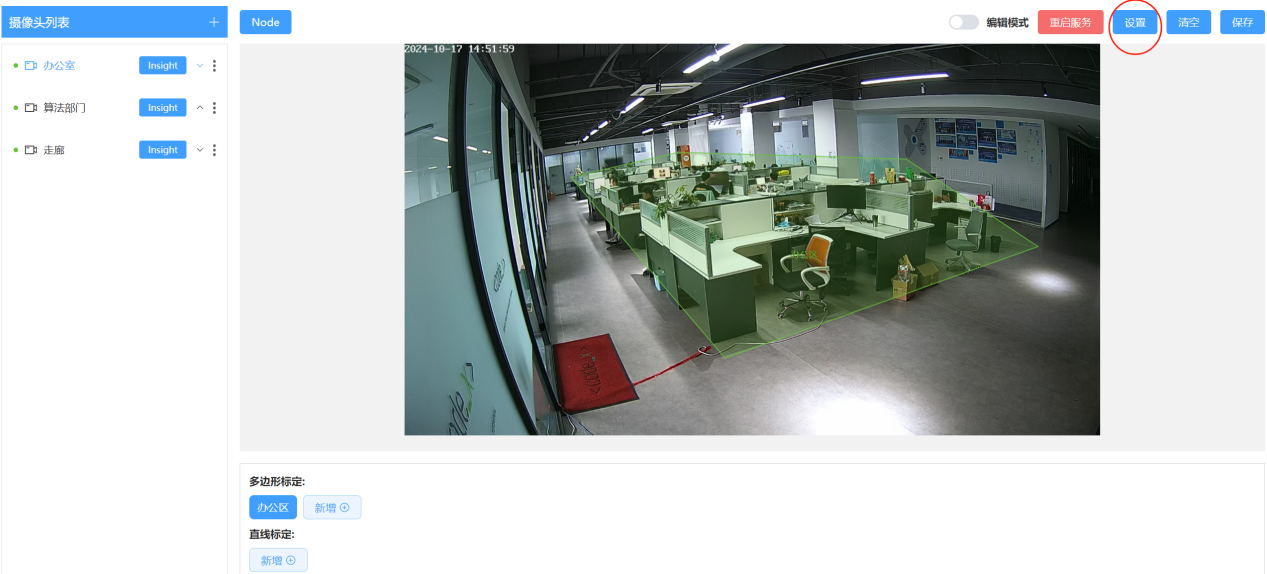


## 算法检测区域添加:



## License授权操作:

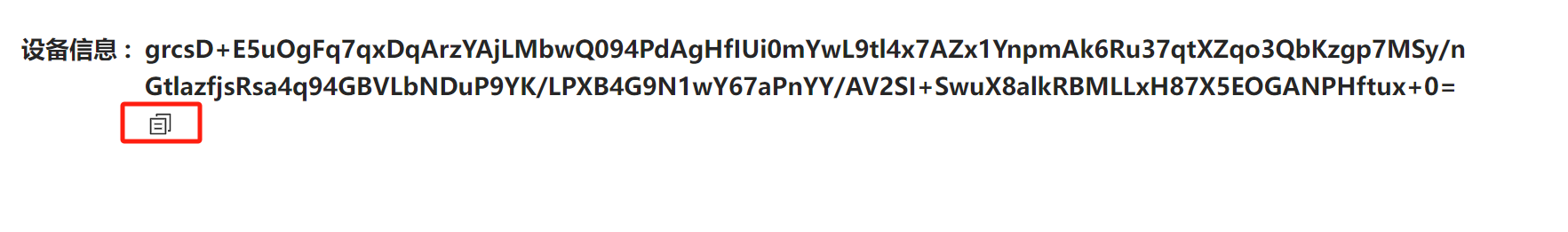
点击主页面***设置***按钮:



再点击***获取***按钮:



最后点击***复制***按钮, 将复制到的设备信息保存并反馈给我司进行授权.



## 支持算法:

|  |  |
| --- | --- |
| **算法类型** | **算法名称** |
| 目标检测 | Yolov7/Yolov8/Yolov10/Yolov11 |
| 旋转目标检测 | Yolov8OBB |
| 语义分割 | mmsegmentation |
| 实例分割 | Yolov11InstanceSeg |
| 手掌关键点检测 | RTMPose |
| 关键点检测 | MoveNet |
| 人脸检测 | Retinaface |
| 人脸识别 | AdaFace/ArcFace |
| 向量匹配 | DinoEmbedding |
| 分类 | Shufflenet/DinoClassifier等 |
| 人体向量匹配 | PPLCNet |
| 跟踪算法 | DeepSort/OCSort |
| 客流统计算法 | HumanFLowStatistics |
| 自定义算法 | 支持 |

# 视频流节点类

支持视频流和视频文件模式("mp4", "avi", "flv")

class VideoStreamer:

VideoStreamer is the base class for the video streamer node.

Args:

nodeName: (str) The name of the node.

stream: (str) The stream address.

queue\_size: (int) The size of the queue.

process\_frame\_interval: (int) The interval of processing frames.

hikvision\_platform: (bool) Whether to use the Hikvision platform.

base\_url: (str) The base url of the Hikvision platform.

appKey: (str) The appKey of the Hikvision platform.

appSecret: (str) The appSecret of the Hikvision platform.

short\_connection\_delay: (float) The short connection delay.

sleep\_time: (float) The sleep time(second, e.g. 0.04s) with video interframe sleep. Just for file mode. e.g. stream="file.mp4"

block\_mode: (bool) Whether to block the frame. Just for file mode. e.g. stream="file.mp4"

once\_mode: (bool) Whether to run once. Just for file mode. e.g. stream="file.mp4"

delay\_start\_time: (float) The delay start time. Just for file mode. e.g. stream="file.mp4"

all\_ready: (bool) Whether to wait for all consumers to be ready.

device: (str) The device of the model, CPU(cpu) or GPU(cuda:x).

代码案例:

**VideoStreamer("视频节点名称", “rtsp://\*\*\*\*\*”, 3, 1)**

# 算法推理节点类

支持各类算法模型,具体见第一章支持算法列表.

class YOLOv10:

YOLOv10 is a class for YOLOv10 model, which is inherited from CDetector.

Args:

nodeName: (str) The name of the node.

modelPath: (str) The path of the model.

queue\_size: (int) The queue size.

inputSize: (list) The input size. e.g. [3, 416, 416]

class\_names: (list) The class names.

valid\_class\_names: (list) The valid class names.

max\_batch\_size: (int) The max batch size.

conf\_thres: (float) The confidence threshold.

iou\_thres: (float) The iou threshold.

warmup: (bool) The warmup flag.

device: (str) The device.

threading\_num: (int) The threading number.

save\_top\_n\_objects: (int) The save top n objects.

area\_flag: (bool) The area flag.

secondary\_class\_names: (list) The secondary class names.

input\_names: (list) The input names.

output\_names: (list) The output names.

gray\_mode: (bool) Whether to use gray mode.

Returns: None

代码案例:

**YOLOv10("zhu\_det", "./src/exma/gutimidu.onnx", queue\_size, (3, 640, 640), max\_batch\_size=1, class\_names=["人头", "台布", "固体", "天平", "手", '托盘护套', "抹布", "滴管", "烧杯", "眼睛", "砝码盒关", "砝码盒开", "量筒", "镊子"])**

class YOLOv8obb:

YOLOv8obb is a class for YOLOv8 model.

Args:

nodeName: (str) The name of the node.

modelPath: (str) The path of the model.

queue\_size: (int) The queue size.

inputSize: (list) The input size. e.g. [3, 416, 416]

class\_names: (list) The class names.

valid\_class\_names: (list) The valid class names.

max\_batch\_size: (int) The max batch size.

conf\_thres: (float) The confidence threshold.

iou\_thres: (float) The iou threshold.

warmup: (bool) The warmup flag.

device: (str) The device.

threading\_num: (int) The threading number.

save\_top\_n\_objects: (int) The save top n objects.

area\_flag: (bool) The area flag.

secondary\_class\_names: (list) The secondary class names.

input\_names: (list) The input names.

output\_names: (list) The output names.

gray\_mode: (bool) Whether to use gray mode.

Returns: None

代码案例:

**YOLOv10("zhu\_det", "./src/exma/gutimidu.onnx", queue\_size, (3, 640, 640), max\_batch\_size=1, class\_names=["人头", "台布", "固体", "天平", "手", '托盘护套', "抹布", "滴管", "烧杯", "眼睛", "砝码盒关", "砝码盒开", "量筒", "镊子"])**

# 结果可视化节点类

class CPipeInsight:

Used to display the video stream in CPipe, and support HTTP display.

Args:

nodeName: The name of the node.

queue\_size: The size of the queue.

show\_scale: The scale of the display. eg: 2 is 1/2 scale.

show\_fps: The show fps.

ip: The ip address.

port: The port.

http\_insight: Whether to enable HTTP display.

ui\_insight: Whether to enable UI display, if True, http\_insight must be False.

save\_video: Whether to save the video stream.

save\_streamer\_node\_name: The streamer node name to save, if None, save all streamer nodes. （Due to performance issues, try to save only one video stream.）

save\_file\_names: The save file name. If None, the file name is the streamer node name + random number. format: {nodeName<Streamer>: file\_name, ...}

save\_path: The save path.

save\_duration\_seconds: The save duration in seconds.

save\_fps: The save fps.

save\_wh: The save width and height.

auto\_exit: Auto exit when all streamer nodes are dead.

show\_polygon\_box: self.kwargs.get("show\_polygon\_box", False)

show\_box: self.kwargs.get("show\_box", True)

show\_box\_name: self.kwargs.get("show\_box\_name", True)

show\_polygon: self.kwargs.get("show\_polygon", True)

show\_mask: self.kwargs.get("show\_mask", True)

show\_key\_points: self.kwargs.get("show\_key\_points", True)

show\_person: self.kwargs.get("show\_person", True)

show\_classification: self.kwargs.get("show\_classification", True)

show\_track: self.kwargs.get("show\_track", True)

chinese\_font\_size = self.kwargs.get("font size", 20)

key\_points\_name\_font\_scale = self.kwargs.get("key\_points\_name\_font\_scale", 1)

key\_points\_color = self.kwargs.get("key\_points\_color", (0, 255, 128))

key\_points\_name\_color = self.kwargs.get("key\_points\_name\_color", (128, 255, 128))

mask\_color = self.kwargs.get("mask\_color", (0, 255, 0))

track\_id\_font\_scale = self.kwargs.get("track\_id\_font\_scale", 1)

track\_id\_color = self.kwargs.get("track\_id\_color", (0, 0, 255))

box\_name\_color = self.kwargs.get("box\_name\_color", (255, 128, 0))

classification\_name\_color = self.kwargs.get("classification\_name\_color", (0, 255, 0))

embedding\_box\_name\_color = self.kwargs.get("embedding\_box\_name\_color", (0, 255, 200))

ssl: Whether to use https.

cert\_path: The cert path with https.

key\_path: The key path with https.

代码案例:

**CPipeInsight(http\_insight=True)**

# 日志使用方法

任何一个继承了Node类的节点都可以直接使用self.logger进行日志记录.

代码案例1:

self.logger.info(f"{self.nodeName}: {self.url}, {host}, {port}")

self.logger.debug

self.logger.warning

self.logger.error

代码案例2:

self.logger.report :上传到日志服务器, 需调在所有节点初始化前对CLogger.init\_report 完成设置

from cpipe.module.model.yolov8 import YOLOv8obb

from cpipe.module.streamer import VideoStreamer, ImageStreamer, VideoStreamers

from cpipe.module.insight import CPipeInsight

from cpipe.module.model.tracker.tracker import Tracker

from cpipe.module.node import Node

from cpipe.config.config import CConfig, LICENSE\_PATH

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

def report\_log\_func(msg):

log\_dict = {

"type": "algorithmLog",

"data": [{

"type": "log\_type",

"algorithmId": "algorithmId",

"algorithmConfigId": "algorithmConfigId",

"content": msg

}]

}

return log\_dict

CLogger().init\_report("websocket", {"websocket\_url": "ws://0.0.0.0:8001"}, report\_log\_func, ("info", "debug", "report"))

init\_report参数说明:

def init\_report(cls, report\_type: str, report\_info: dict, report\_func, report\_level=("error", "report")):

"""

Initialize the report function

Args:

report\_type: "" or "websocket"

report\_info: {"websocket\_url": "..."}

report\_func: function(msg) -> json

report\_level: "error" or "info" or "debug" or "report"

Returns: None

"""

# 结果上报节点类

支持HTTP/MQTT/Websocket上报类型.

class WebsocketReport:

WebsocketReport is a class for sending data through websocket.

Args:

nodeName: (str) The name of the node.

queue\_size: (int) The size of the queue.

ws\_url: (str) The url of the websocket server.

class MQTTReport:

MQTTReport is a class for sending

Args:

nodeName: (str) The name of the node.

queue\_size: (int) The size of the queue.

username: (str) The username of the MQTT server.

password: (str) The password of the MQTT server.

broker: (str) The broker of the MQTT server.

port: (int) The port of the MQTT server.

class HTTPReport:

HTTPReport is a class for sending.

Args:

nodeName: (str) The name of the node.

queue\_size: (int) The size of the queue.

host: (str) The host of the server.

port: (int) The port of the server.

url: (str) The url of the server.

report\_type: (str) The type of the report. It can be "POST", "PUT" or "GET".

repeat\_time: (int) The repeat time.

timeout\_base: (int) The base timeout.

success\_code: (tuple) The success code.

代码案例:

**HTTPReport(“report”, 3, “http://192.168.8.7”, 8000, ”http://192.168.8.7:8000/report“)**

# CPipe配置文件说明

## CPipe配置文件

首次启动CPipe会在当前工程目录下生成一个config文件夹并在文件夹中生成一个config.yaml文件.

配置文件内容如下:

基础版CPipe License不支持开启

SHARE\_MEMORY\_MODE: False

BETWEEN\_NODES\_SHARE\_MEMORY\_MODE: False

CPIPE\_BLOCKING\_MODE: False

VIDEO\_CUDA\_MODE: False

只有性能版License支持.

#################################CPipe Configs#################################

# Share memory mode, it can improve the speed of the algorithm.

SHARE\_MEMORY\_MODE: False

# If the memory sharing function is enabled among all nodes, it can prevent picture frame dislocation but increases CPU load.

BETWEEN\_NODES\_SHARE\_MEMORY\_MODE: False

# The CPIPE blocking mode must be enabled with BETWEEN\_NODES\_SHARE\_MEMORY\_MODE=True, for between\_nodes\_SHARE\_memory\_mode = true, and is only suitable for local video mode.

CPIPE\_BLOCKING\_MODE: False

# Enable video decode cuda acceleration.

VIDEO\_CUDA\_MODE: False

# Log level eg. DEBUG, INFO, WARNING, ERROR

CLOGER\_LEVEL: "DEBUG"

# Interval for starting processes on all nodes. By default, the system waits 3 seconds before starting the next node.

# If you have a lot of video streaming, it is recommended to increase the NODE\_LAUNCH\_INTERVAL\_TIME value to prevent the system from starting too many processes at once and causing the CPipe to fail to start.

NODE\_LAUNCH\_INTERVAL\_TIME: 0.01

# cmask yaml path.

CMASK\_YAML\_PATH: "./cmask.yaml"

# cmask yaml auto load.

CMASK\_YAML\_AUTO\_LOAD: True

# CPipe Performance license file path.

LICENSE\_PATH: "./cpipe.license"

# Performance license file password.

LICENSE\_PS: "1234567890123456"

# CPipe Basic license file path.

CODEX\_LICENSE\_PATH: "./codex.cpipe.license"

## Yaml配置文件解析

代码示例:

from cpipe.config.config import CConfig

opt = CConfig.read\_config("./project/project.yaml")

# 自定义可视化功能

1. Web文本显示功能:  
   

通过CData的messages成员变量添加.

1. 自定义可视化功能:

通过CData的shows成员变量添加.

支持显示文本/线/多边形.

class CData:

def \_\_init\_\_(self, createNodeName):

"""

CData is the core data storage class of CPipe, which is used to store the output results of all detection, recognition, tracking, and other model and logic codes.

Any image inference result and logic result should be converted into a Box object data and then stored in the CData object.

Args:

createNodeName: The name of the node that generates the CData object.

Attributes:

timestamp: The timestamp of the CData object.

json: The JSON data of the CData object.

images: The image data of the CData object.

states: The special state data of the CData object. The format is {"any": any}.

shows: The result of the algorithm display.

eg:

{

"stream\_name": [

("text", {"coord":np.array([996, 996], dtype=np.int32), "data": 996, "color":(0, 0, 255), "text\_size": 40}),

("line", {"coord":np.array([[996, 996], [996, 996]], dtype=np.int32), "color":(0, 0, 255), "thickness": 1}),

("polygon", {"coord":np.array([[700, 100], [1820, 980], [1000, 1030], [550, 800]], dtype=np.int32), "color":(0, 0, 255), "thickness": 1, "isclosed": True}),

],

}.

messages: The message data of the CData object.

eg:

{

"stream\_name": [

"msg1....",

"msg2....",

"msg3...."

],

}.

bboxes: The Box data of the CData object. The format is {"stream\_name": [Box, Box, Box, ...]}.

"""

self.createNodeName = createNodeName

self.timestamp = 0

self.json = {}

self.images = {}

self.states = {}

self.shows = {}

self.messages = {}

self.bboxes = {}

# 模型文件加密

加密后的模型文件格式必须是.codex结尾.

加密时需提供license文件路径和license秘钥.

直接运行下面的代码进行模型加密或者调用cpipe.tools.model\_encryption

import argparse

from cpipe.module.security import Security, AESHelper

import os

def parse\_args():

parser = argparse.ArgumentParser(description='generate license file')

# parser.add\_argument('--model\_path', default="../src/model\_files/hand-end2end.onnx", type=str, help='model path')

parser.add\_argument('--model\_path', default="../src/dengpao/kaiguan\_11\_29\_128x192\_gray\_batch64.engine", type=str, help='model path')

parser.add\_argument('--password', default="1234567890123456", type=str, help='model path')

parser.add\_argument('--license\_path', default="../cpipe.license", type=str, help='cpipe.license path')

return parser.parse\_args()

def encrypt\_model(modelPath, model\_password, iv):

with open(modelPath, "rb") as f:

data = f.read()

model = AESHelper(model\_password, iv).encrypt(data)

new\_path = modelPath + ".codex"

with open(new\_path, "wb") as nf:

nf.write(model)

nf.close()

return new\_path

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

args = parse\_args()

license\_path = args.license\_path

print("license\_path:", license\_path)

modelPath = args.model\_path

password = args.password

if len(args.password) != 16:

print("密码长度不正确, 请输入16位密码.")

exit(1)

lincense, iv = Security.get\_license\_iv(license\_path)

device\_id = AESHelper(password, iv).decrypt(lincense)

model\_password = device\_id[:16]

if os.path.isfile(modelPath):

new\_path = encrypt\_model(modelPath, model\_password, iv)

print("encrypt model file success. path: " + new\_path)

elif os.path.isdir(modelPath):

for file\_name in os.listdir(modelPath):

model\_file = os.path.join(modelPath, file\_name)

new\_path = encrypt\_model(model\_file, model\_password, iv)

print("encrypt model file success. path: " + new\_path)

print("done.")

# CPipe常用数据类

## Box类:

用于存储所有检测目标的数据,每一个实例化的Box对应一个检测目标.

class Box:

def \_\_init\_\_(self):

"""

Box class is a core data storage class of CPipe, which is used to store the output results of all detection, recognition, tracking, and other model and logic codes.

Any image inference result and logic result should be converted into a Box object data and then stored in the CData object.

Attributes:

box\_coord: The coordinates of the bounding box. The format is [x1, y1, x2, y2].

box\_polygon\_coord: The coordinates of the polygon. The format is [[x1, y1], [x2, y2], [x3, y3], [x4, y4], [x5, y5], ...].

box\_score: The confidence score of the bounding box. The value range is [0, 1].

box\_class: The class ID of the bounding box. The default value is 0.

box\_class\_name: The name of the class. The default value is an empty string.

box\_img: The image of the bounding box. The format is [height, width, channel], is a numpy.ndarray.

box\_key\_points: The key points of the bounding box. The format is [[x1, y1], [x2, y2], [x3, y3], [x4, y4], [x5, y5], ...].

box\_mask: The image mask of the bounding box. The format is a numpy.ndarray of [height, width]. Based on the coordinates of box\_coord, applying box\_mask to the original image can extract the target area corresponding to box\_coord.

box\_embedding: The feature vector of the bounding box. The length of the feature vector is 512/768..., is a numpy.ndarray.

box\_embedding\_name: The name of the feature vector. The default value is an empty string.

box\_embedding\_score: The confidence score of the feature vector. The value range is [0, 1].

classification: The classification information of the target. It is an instance of the Classification class. The default value is None.

person: The person information of the target. It is an instance of the Person class. The default value is None.

track: The tracking information of the target. It is an instance of the Tracker class. The default value is None.

"""

self.box\_coord = []

self.box\_polygon\_coord = []

self.box\_score = 0

self.box\_class = 0

self.box\_class\_name = ""

self.box\_img = None

self.box\_key\_points = None

self.box\_key\_point\_scores = None

self.box\_key\_point\_names = None

self.box\_mask = None

self.box\_embedding = None

self.box\_embedding\_name = None

self.box\_embedding\_score = None

self.classification: Optional[Classification] = None

self.person: Optional[Person] = None

self.track: Optional[Tracker] = None

## CData类:

用于存储每个Node节点间队列数据的存储. 一个CData包含一帧数据的所有信息.

class CData:

def \_\_init\_\_(self, createNodeName):

"""

CData is the core data storage class of CPipe, which is used to store the output results of all detection, recognition, tracking, and other model and logic codes.

Any image inference result and logic result should be converted into a Box object data and then stored in the CData object.

Args:

createNodeName: The name of the node that generates the CData object.

Attributes:

timestamp: The timestamp of the CData object.

json: The JSON data of the CData object.

images: The image data of the CData object.

states: The special state data of the CData object. The format is {"any": any}.

shows: The result of the algorithm display.

eg:

{

"stream\_name": [

("text", {"coord":np.array([996, 996], dtype=np.int32), "data": 996, "color":(0, 0, 255), "text\_size": 40}),

("line", {"coord":np.array([[996, 996], [996, 996]], dtype=np.int32), "color":(0, 0, 255), "thickness": 1}),

("polygon", {"coord":np.array([[700, 100], [1820, 980], [1000, 1030], [550, 800]], dtype=np.int32), "color":(0, 0, 255), "thickness": 1, "isclosed": True}),

],

}.

messages: The message data of the CData object.

eg:

{

"stream\_name": [

"msg1....",

"msg2....",

"msg3...."

],

}.

bboxes: The Box data of the CData object. The format is {"stream\_name": [Box, Box, Box, ...]}.

"""

self.createNodeName = createNodeName

self.timestamp = 0

self.json = {}

self.images = {}

self.states = {}

self.shows = {}

self.messages = {}

self.bboxes = {}

## CImage类:

用于存储每一帧画面的数据.

class CImage:

def \_\_init\_\_(self):

"""

Used to store the image data of a Streamer in CPipe.

Attributes:

frame\_total: Streamer total frame number. The default value is 0. Only the video stream read by Streamer has this attribute.

frame\_current\_idx: The current frame index of the Streamer. The default value is 0. Only the video stream read by Streamer has this attribute.

"""

self.frame\_total = 0

self.frame\_current\_idx = 0

# 工程案例:

案例基于物理考试评分系统,包括如果创建一个新的节点,以及逻辑代码编写、整体初始化和效果展示.

## 自定义节点类:

*from* cpipe.module.node *import* Node

class Exam(Node):

def \_\_init\_\_(self, nodeName, queue\_size, is\_running: multiprocessing.Value):

super().\_\_init\_\_(nodeName, queue\_size)

self.\_\_is\_running = is\_running

self.streamer\_materials = {}

def init\_scoring\_points(self):

raise NotImplementedError("init\_scoring\_points not implemented")

@property

def is\_running(self):

if self.\_\_is\_running.value == 1:

return True

else:

return False

@is\_running.setter

def is\_running(self, value: bool):

if value:

self.\_\_is\_running.value = 1

else:

self.\_\_is\_running.value = 0

## 部分逻辑代码块-得分点:

class ScoringPoint:

作用:

ScoringPoint 类用于表示评分点。

参数:

streamer\_name (str): 流媒体名称。

check\_num (str): 检查编号。

score (float): 分数。

logic (callable): 评分逻辑函数。

labels (list): 标签列表。

check\_remark (str): 检查备注。

pre\_scoring\_points (list): 前置评分点列表。

post\_scoring\_points (list): 后置评分点列表。

max\_history\_len (int): 最大历史记录长度。

history\_threshold (int): 历史记录阈值。

is\_done (bool): 是否完成。

\*\*kwargs: 其他关键字参数。

代码案例:

**ScoringPoint(ce\_streamer\_name, "12", 1.0, self.water\_line\_score, ["水位线", "眼睛", "固体"], "测量出被测固体和水的总体积并记录", [[zhu\_streamer\_name, "11"]], max\_history\_len=15, history\_threshold=4)**

## 部分逻辑代码块-得分逻辑判断:

判断水位线位置.

def **water\_line**(*self*, *bboxes*, \**args*, \*\**kwargs*):

        bboxes\_dict = {one\_box.box\_class\_name: one\_box *for* one\_box *in* *bboxes*}

*if* *kwargs*.get("history\_key") == "initial\_water\_line\_history":

*if* "固体" in bboxes\_dict.keys() and "量筒" in bboxes\_dict.keys():

*# if CLogic.is\_box\_inside(bboxes\_dict["固体"].box\_coord, bboxes\_dict["量筒"].box\_coord):*

*if* CLogic.box\_iou(bboxes\_dict["固体"].box\_coord, bboxes\_dict["量筒"].box\_coord) > 0:

                    scoring\_point = *kwargs*.get("scoring\_point")

                    scoring\_point.end = True

                    print("init\_water\_line end!!!")

*return* 0

*if* scoring\_point := *kwargs*.get("scoring\_point"):

            box\_names = set(bboxes\_dict.keys())

            tmp = [one *for* one *in* scoring\_point.labels *if* one != "眼睛"]

            labels = set(tmp)

                all\_water\_line\_center\_y = []

                all\_water\_line\_center\_y\_label = []

                all\_water\_line\_center\_idx = []

*for* idx, one\_ml *in* enumerate(['10ml', '20ml', '30ml', '40ml', '50ml', '60ml', '70ml', '80ml', '90ml', '100ml']):

*if* one\_ml in bboxes\_dict.keys():

                        all\_water\_line\_center\_y.append(bboxes\_dict[one\_ml].box\_coord[3])

                        all\_water\_line\_center\_y\_label.append(int(one\_ml[:-2]))

                        all\_water\_line\_center\_idx.append(idx)

*if* len(all\_water\_line\_center\_y) < 6:

*return* 0

*else*:

                    diff\_ret = np.abs(np.diff(np.array(all\_water\_line\_center\_y)))

                    diff\_ret\_idx = np.abs(np.diff(np.array(all\_water\_line\_center\_idx)))

                    diff\_ret = diff\_ret / diff\_ret\_idx

                    md = np.median(np.abs(diff\_ret))

                    mx = np.max(diff\_ret)

                    mi = np.min(diff\_ret)

*if* np.abs(md - mx) > md \* 0.2:

*return* 0

*if* np.abs(md - mi) > md \* 0.2:

*return* 0

*if* *kwargs*.get("history\_key") == "final\_water\_line\_history":

*if* "10ml" not in bboxes\_dict:

*return* 0

*if* CLogic.box\_iou(bboxes\_dict["10ml"].box\_coord, bboxes\_dict["固体"].box\_coord) <= 0:

*if* scoring\_point.start and time.time() - scoring\_point.start\_time > 3:

                            scoring\_point.end = True

*self*.logger.info("final\_water\_line end!!!")

*return* 0

*else*:

                        scoring\_point.start = True

                all\_water\_line\_center\_y = np.array(all\_water\_line\_center\_y)

                water\_line\_center\_y = (bboxes\_dict["水位线"].box\_coord[1] + bboxes\_dict["水位线"].box\_coord[3]) / 2

                all\_water\_line\_center\_y\_between = water\_line\_center\_y - all\_water\_line\_center\_y

                water\_line\_center\_y\_min\_idx = np.argmin(np.abs(all\_water\_line\_center\_y\_between))

                base\_label = all\_water\_line\_center\_y\_label[water\_line\_center\_y\_min\_idx]

*if* all\_water\_line\_center\_y\_between[water\_line\_center\_y\_min\_idx] > 0:

                    up\_label = all\_water\_line\_center\_y\_label[water\_line\_center\_y\_min\_idx]

*if* water\_line\_center\_y\_min\_idx - 1 >= 0:

                        down\_label = all\_water\_line\_center\_y\_label[water\_line\_center\_y\_min\_idx - 1]

                        dis\_pixel = np.abs(all\_water\_line\_center\_y\_between[water\_line\_center\_y\_min\_idx] - all\_water\_line\_center\_y\_between[water\_line\_center\_y\_min\_idx - 1])

*else*:

*return* 0

*else*:

                    down\_label = all\_water\_line\_center\_y\_label[water\_line\_center\_y\_min\_idx]

*if* water\_line\_center\_y\_min\_idx + 1 < len(all\_water\_line\_center\_y):

                        up\_label = all\_water\_line\_center\_y\_label[water\_line\_center\_y\_min\_idx + 1]

                        dis\_pixel = np.abs(all\_water\_line\_center\_y\_between[water\_line\_center\_y\_min\_idx + 1] - all\_water\_line\_center\_y\_between[water\_line\_center\_y\_min\_idx])

*else*:

*return* 0

                dis\_scale = up\_label - down\_label

                total\_water\_line = base\_label + (dis\_scale / dis\_pixel) \* (all\_water\_line\_center\_y[water\_line\_center\_y\_min\_idx] - water\_line\_center\_y)

*self*.exma\_result[*kwargs*.get("history\_key")].append(total\_water\_line)

*if* len(*self*.exma\_result[*kwargs*.get("history\_key")]) > scoring\_point.max\_history\_len:

*self*.exma\_result[*kwargs*.get("history\_key")].pop(0)

*if* len(*self*.exma\_result[*kwargs*.get("history\_key")]) > scoring\_point.history\_threshold:

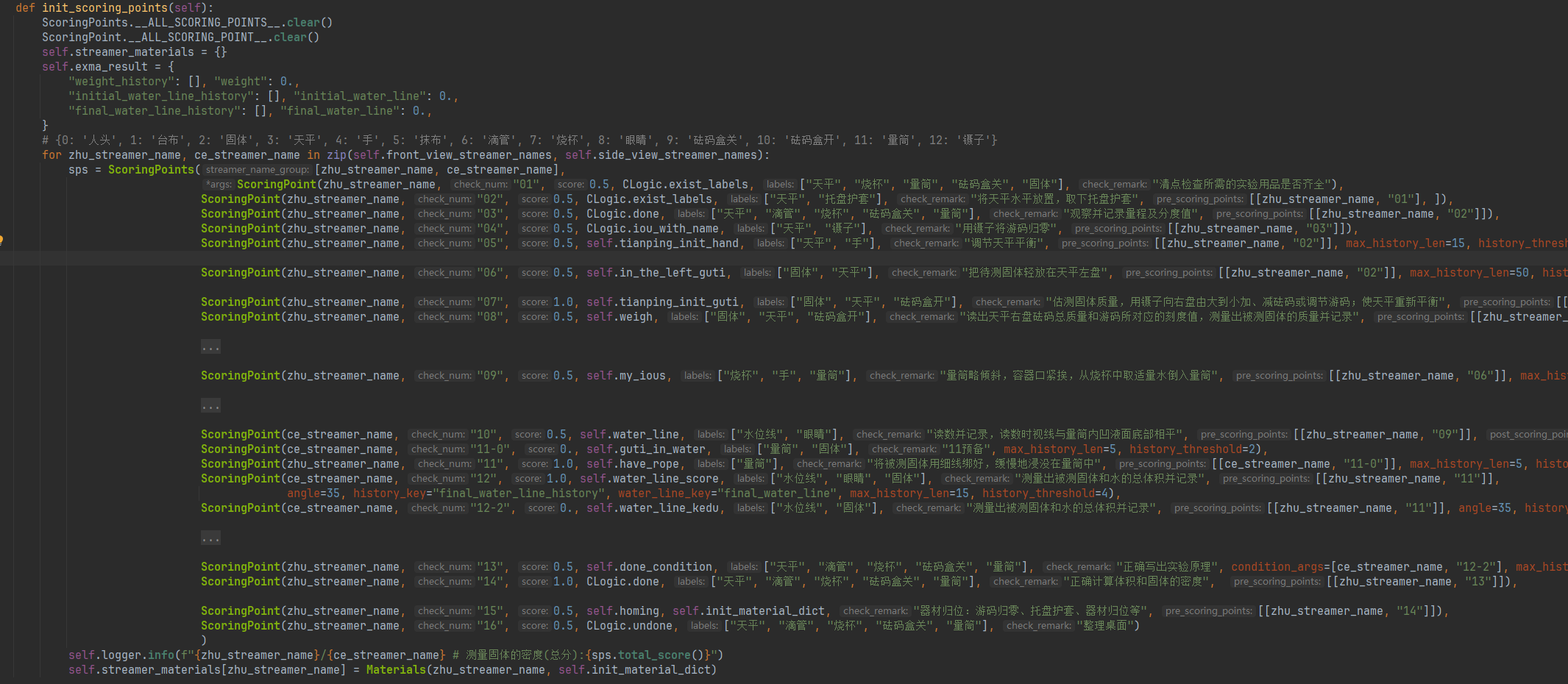
*self*.exma\_result[*kwargs*.get("water\_line\_key")] = np.median(np.array(*self*.exma\_result[*kwargs*.get("history\_key")]))

                    scoring\_point.show\_info = f"#水位#:{str(round(*self*.exma\_result[*kwargs*.get('water\_line\_key')], 2))}ml"

*return* ret

*return* 0

## 整体逻辑代码:



功能说明:

A.初始化所有得分点以及对应所有视频流.

B.一个考试一个ScoringPoints, 一个ScoringPoints包括多个ScoringPoint.

C.ScoringPoint参数包括: 视频流名称、得分点编号、分数、操作逻辑函数、检测目标名称、前置条件、后置条件、最大历史记录、最大历史阈值等等.

## **初始化整个工程:**

*from* cpipe.module.streamer *import* VideoStreamer

*from* cpipe.module.insight *import* CPipeInsight

*from* cpipe.module.model.yolov8 *import* YOLOv8obb

*from* cpipe.module.report *import* HTTPReport

*from* project.exam *import* Exam

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

# 初始化主视图视频流节点

stream\_zhu = VideoStreamer("zhushitu", "rtsp://admin:admin@192.168.8.109:554/Streaming/Channels/101", 3, 1, *once\_mode*=True, *mode*="cuda")

# 初始化侧视图视频流节点

stream\_ce = VideoStreamer("ceshitu", "rtsp://admin:admin@192.168.8.111:554/Streaming/Channels/101", 3, 1, *once\_mode*=True, *mode*="cuda")

# 初始化主视图检测节点

    zhu\_det = YOLOv8obb("zhu\_det",

                        "./src/exma/gutimidu1.onnx",

                        queue\_size,

                        ["images"],

                        ["output0"],

                        (3, 640, 640),

*warmup*=True,

*max\_batch\_size*=1,

*class\_names*=["人头", "台布", "固体", "天平", "手", '托盘护套', "抹布", "滴管", "烧杯", "眼睛", "砝码盒关", "砝码盒开", "量筒", "镊子"],

*conf\_thres*=0.5, *iou\_thres*=0.5)

# 初始化侧视图检测节点

ce\_det = YOLOv8obb("ce\_det",

                        "./src/exma/gutimidu2.onnx",

                        queue\_size,

                        ["images"],

                        ["output0"],

                        (3, 640, 640),

*warmup*=True,

*max\_batch\_size*=1,

*class\_names*=["人头", "台布", "固体", "天平", "手", '托盘护套', "抹布", "滴管", "烧杯", "眼睛", "砝码盒关", "砝码盒开", "量筒", "镊子"],

*conf\_thres*=0.5, *iou\_thres*=0.5)

# 初始化上报节点

report = HTTPReport("report", 3, "http://192.168.8.7", 8000, "http://192.168.8.7:8000/report")

# 初始化考试节点

exam = Exam("gutimidu", 3, ["zhushitu"], ["ceshitu"], is\_running)

# 初始化可视化节点

cpipeinsight = CPipeInsight(*http\_insight*=True)

# 链接所有节点关系

    stream\_zhu += [zhu\_det, exam, report]

    stream\_ce += [ce\_det, exam, report]

exam += cpipeinsight

# 启动整个程序

# check\_node=True, 默认自动检查所以节点

# check\_interval=60, 检查间隔(秒)

# auto\_restart=True, 如果节点退出自动恢复,配合check\_node=True使用

# launch\_config\_path=None, 无代码一键启动cpipe工厂使用

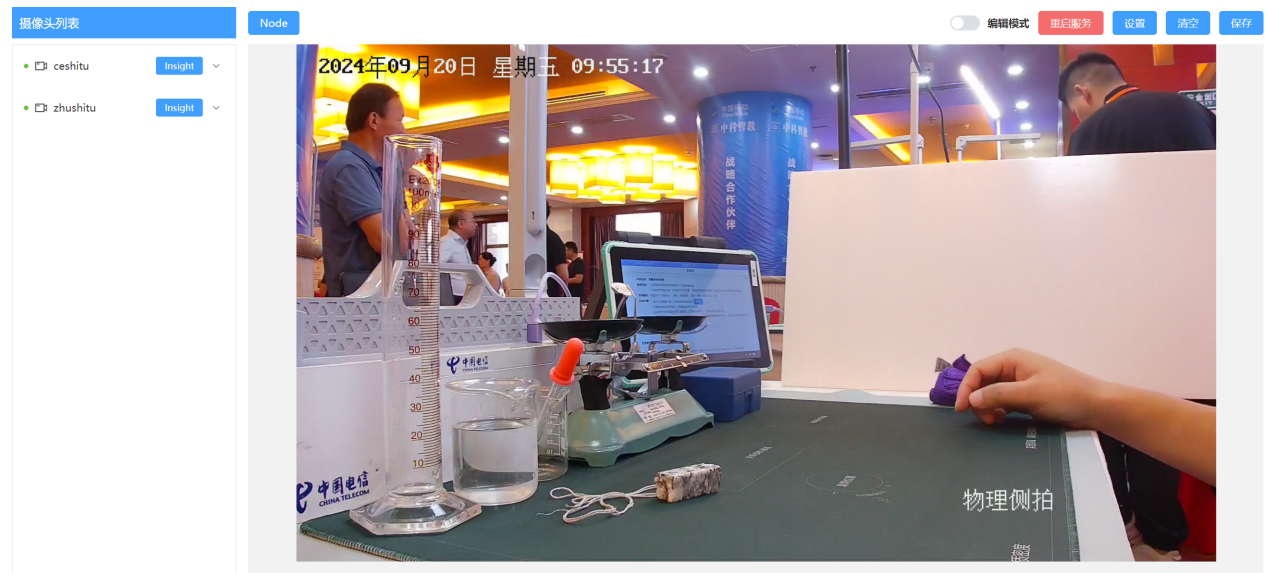
# auto\_handle\_signal=True 自动捕获退出信号

    Node.launch()

## **可视化页面展示:**

**直接在浏览器打开地址: http://127.0.0.1:9966**

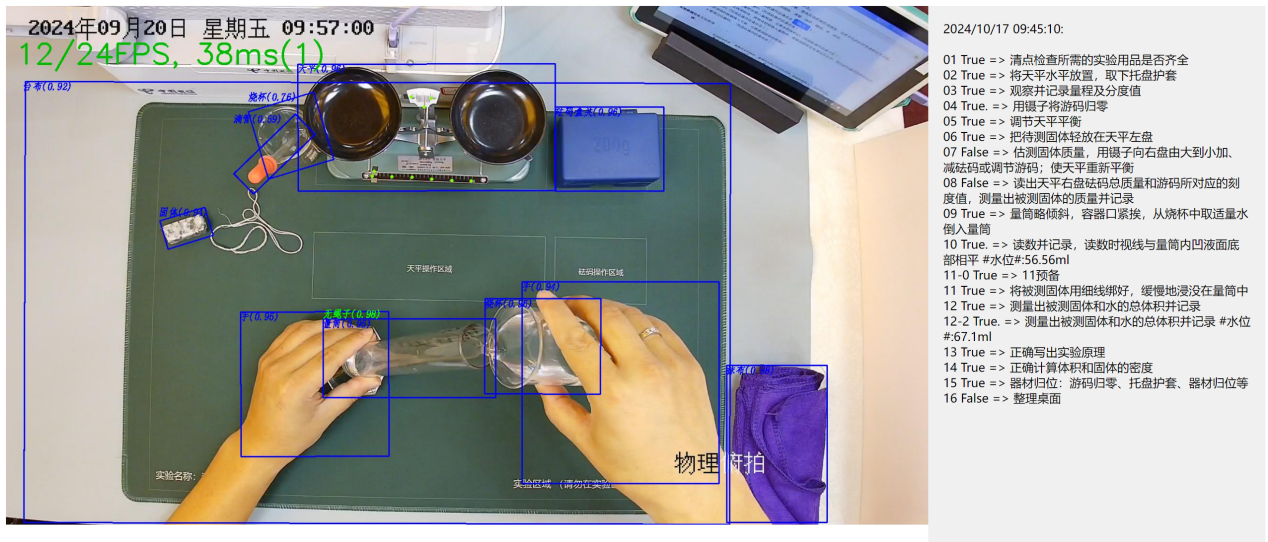
**首页画面:**



**点击首页画面中ceshitu摄像头的Insight按钮进入下面结果可视化画面:**



**点击首页画面中zhushitu摄像头的Insight按钮进入下面结果可视化画面:**



# Windows环境使用方法:

1. 安装WSL2:(ubuntu系统)

参考链接:

<https://blog.csdn.net/yoyokaka/article/details/140186256>

注: WSL2用的是windows系统的显卡驱动程序,但是cuda和cudnn需要自己在WSL2的ubuntu里面安装的.

1. 安装cpipe:

和在其他linux使用方法一样.

1. Python程序debug怎么办:

可以用pycharm自带的WSL适配功能,可以直接读取到WSL的python或conda环境.

1. License:

请使用CPipe3.5.0以上版本的license.

1. CPipe页面如何打开:

直接开启127.0.0.1:9966即可

1. 如何将WSL2中的网络对外(局域网其它主机)访问:

打开windows的powershell输入下面的指令:  
netsh interface portproxy add v4tov4 listenport=22022 listenaddress=0.0.0.0 connectport=22022 connectaddress=172.22.209.98

这里的172.22.209.98是WSL的ifconfig查看到的本机ip地址. 22022是映射的端口.