

엣지 디바이스에서의 YOLO 추론 성능 최적화

기계공학부 강민석

발표 주제 선정 배경

사용자 맞춤형 자동 조정 모니터 스탠드

발표자: 조승윤, 강동재, 강민석* (1분반)

* 팀장: 010-2274-3820, rkd2274@pusan.ac.kr

지도교수: 민준기

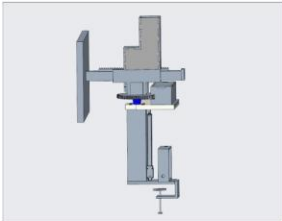
제품 개발의 목표

- 사용자의 자세를 감지하는 스마트 모니터 시스템 개발
- 자동화된 자세 감지 및 실시간 모니터 위치 제어 로직 개발
- 비용 절감을 위한 저 사양 환경에서의 딥러닝 모델 최적화

제품 개발의 필요성

- 인체공학적 사무 기기의 필요성 증대
- 수동 조작이 필요한 기존 제품의 단점 개선
- 딥러닝 기반 컴퓨터 비전 기술로 센서 기반 제품의 한계 극복

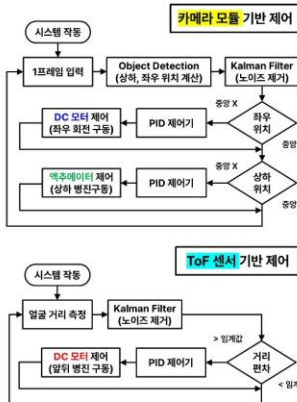
Product Overview



사용자의 얼굴을 추적하는 모니터 스탠드

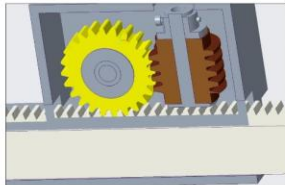
- 오픈소스 딥러닝 모델 기반의 얼굴 위치 추적
- 연산부와 제어부를 분리하여 안정적인 시스템 제어
- 부분 교체가 용이한 완전 모듈형 구조
- 조립 순서를 강제하는 설계로 체결 오류 사전 방지

Flow Chart



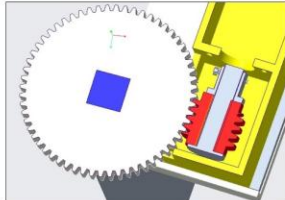
Drive System

Horizontal Linear Drive



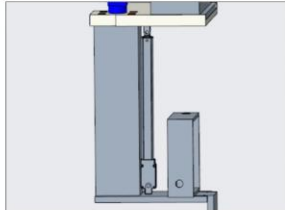
- Worm with Rack and Pinion
- Gear Ratio: 23 : 1
- DC Motor(1) Driven

Horizontal Rotational Drive



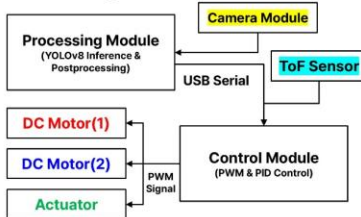
- Worm and Worm Wheel
- Gear Ratio: 60 : 1
- DC Motor(2) Driven

Vertical Linear Actuator



- Standalone Linear Actuator

Control System



Object Detection



- Base Model: YOLOv8
- Fine-Tuned with Face Dataset
- Model Inference Accelerated by NPU

Spec Sheet

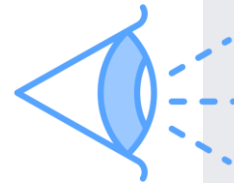
	Drive System		
	Linear Drive	Rotational Drive	Linear Actuator
Ratio	3.14 mm/rot	6 deg/rot	-
ROM	+75mm Forward -59mm Backward	36.2~360° CW 120.5~360° CCW	0~150 mm
Power	12V/2.5A/30W	12V/2.5A/30W	12V/0.25A/3W
*Weight	715g	685g	150g

* Total Weight: 2,365g

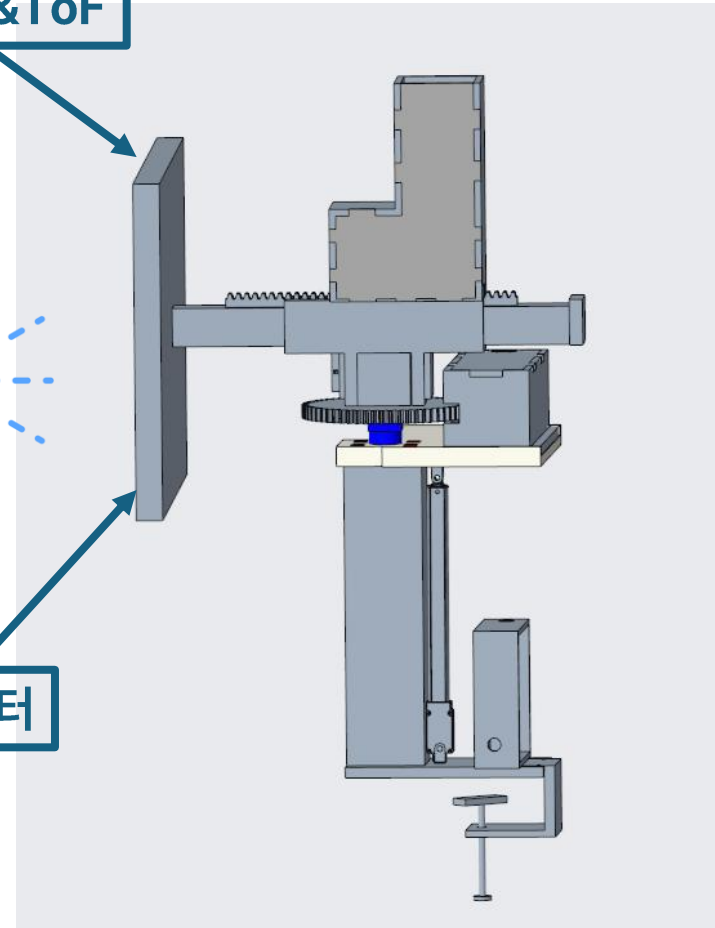
	Control System			
	Processing Module		Control Module	
	Model	yolov8n-face	PWM	Duty
	Res	640x640 pixels	Freq	6~8% 30Hz
Obj. Detection	*Acc	79.6~94.6%	Kp	2.0
	Format	RKNN(INT8)	Ki	0
	Inference	~50ms	Kd	1.0
Latency	Post-Proc	~50ms	Communication	
	Total	~100ms		
	Power	5V/3A/15W	Protocol	USB Serial
			Baud Rate	9600

* Accuracy: IoU Threshold > 0.5

카메라&ToF



모니터



결과 및 기대효과

- 사용자의 개입 없이 보기 편한 위치로 모니터 위치 자동 제어
- 모니터의 위치와 관계 없이 자유롭게 자세를 바꾸며 사용 가능
- 장시간 불편한 자세 유지로 인한 부작용 최소화

향후 진행 계획

- 구동부 크기를 줄이고, 최대 하중을 늘리기 위해 새로운 Gear Set 연구
- 구조적 안정성을 위해 전반적인 구조 개편 및 부품 정밀도 향상
- 딥러닝 모델 경량화 및 최적화를 통해 반응속도 개선
- 사용자가 설정값을 변경할 수 있는 GUI 설정 프로그램 개발

높은 정확도, 빠른 반응 속도



한정된 예산, 현실적인 가격 설정



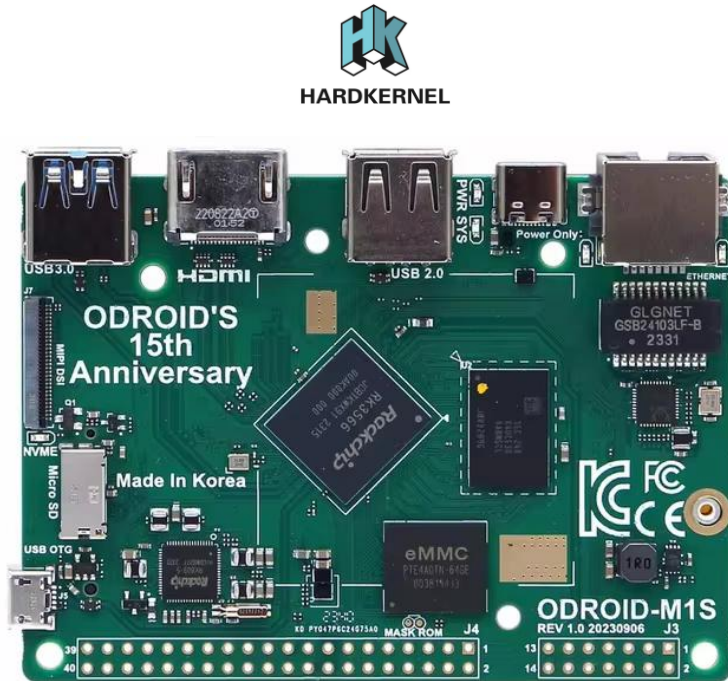
ODROID-M1S(4GB)

- RK3566(ARMv8-A)
- NPU: **0.8TOPS@INT8**
- Power: **5V/3A/15W**
- ₩66,200



Raspberry Pi 5(4GB)

- ARM Cortex-A76 CPU
- NPU: 미탑재
- Power: 5V/5A/25W
- ₩85,200



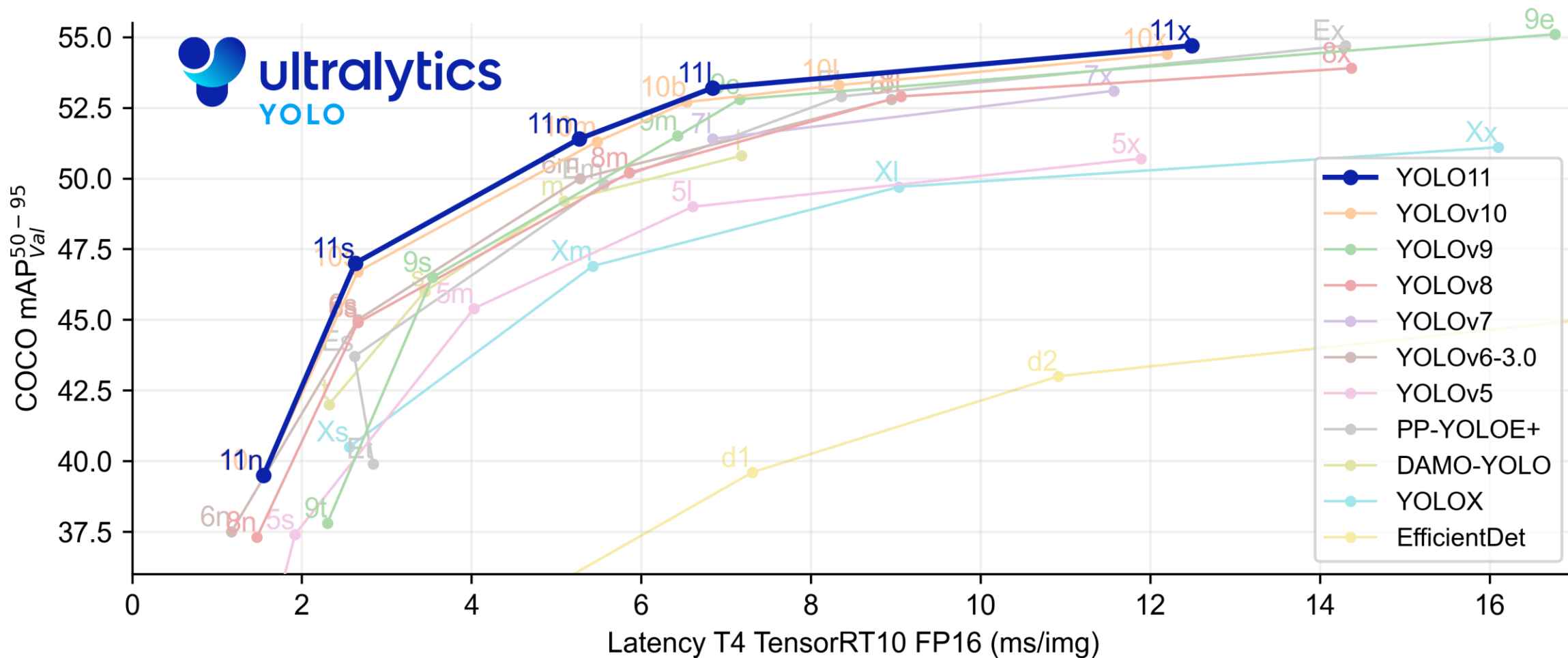
ODROID-M1S(4GB)

- RK3566(ARMv8-A)
- NPU: 0.8TOPS@INT8
- Power: 5V/3A/15W
- ₩66,200



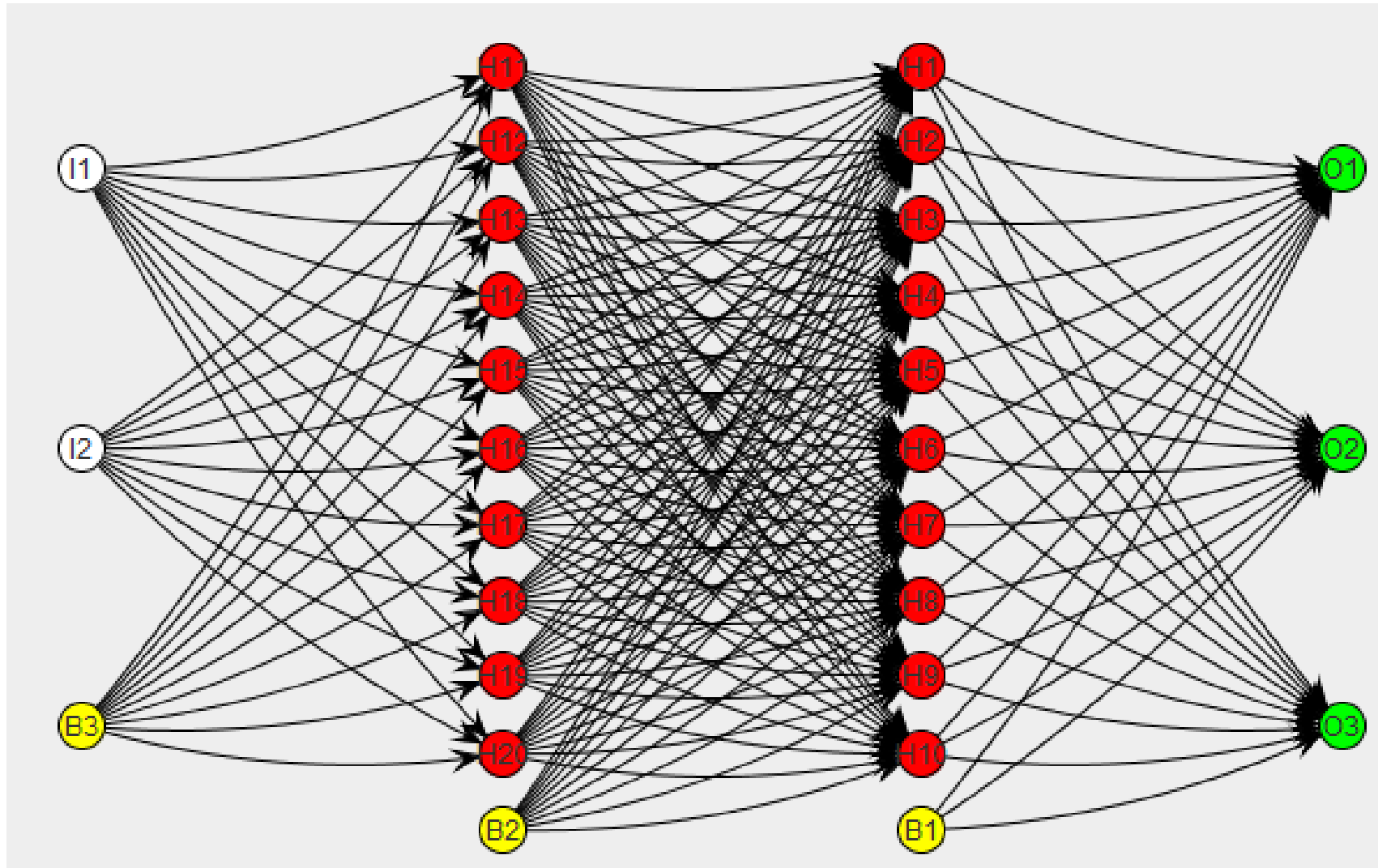
Raspberry Pi 5(4GB)

- ARM Cortex-A76 CPU
- NPU: 미탑재
- Power: 5V/5A/25W
- ₩85,200




CPU를 통한 실시간 추론?

Deep Neural Network



Predict on CPU: PyTorch

 Predict

Return a list with `stream=False`

Return a generator with `stream=True`

```
from ultralytics import YOLO

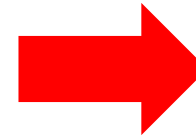
# Load a model
model = YOLO("yolo11n.pt") # pretrained YOLO11n model

# Run batched inference on a list of images
results = model(["image1.jpg", "image2.jpg"], stream=True) # return a generator of Results objects

# Process results generator
for result in results:
    boxes = result.boxes # Boxes object for bounding box outputs
    masks = result.masks # Masks object for segmentation masks outputs
    keypoints = result.keypoints # Keypoints object for pose outputs
    probs = result.probs # Probs object for classification outputs
    obb = result.obb # Oriented boxes object for OBB outputs
    result.show() # display to screen
    result.save(filename="result.jpg") # save to disk
```

Predict on CPU: PyTorch

```
1 from ultralytics import YOLO
2 from utils.camera import setup_camera
3 import cv2
4 from time import time
5
6 if __name__ == "__main__":
7     model = YOLO("models/yolo11n.pt", task="detect")
8     #model = YOLO("models/yolo11n_openvino_model", task="detect")
9     cap = setup_camera(640, 640)
10
11     frames, loop_time, init_time = 0, time(), time()
12     while cap.isOpened():
13         frames += 1
14         ret, frame = cap.read()
15         if not ret:
16             break
17         results = model.predict(frame, verbose=False)
18         result = results[0]
19         boxes = result.boxes
20         if frames % 10 == 0:
21             print(f"{10 / (time() - loop_time):.4f} FPS")
22             loop_time = time()
23         if frames == 100:
24             break
25         if boxes is None:
26             continue
27         max_size, max_idx = 0, -1
28         for idx, box in enumerate(boxes):
29             x1, y1, x2, y2 = box.astype(int)
30             size = (x2 - x1) * (y2 - y1)
31             max_size, max_idx = (size, idx) if size > max_size else (max_size, max_idx)
32         if max_idx == -1:
33             continue
34         x1, y1, x2, y2 = boxes[max_idx].astype(int)
35         if cv2.waitKey(1) & 0xFF == ord('q'):
36             break
37         if frames == 100:
38             break
39
40     print(f"\ntotal: {frames / (time() - init_time)} FPS")
```

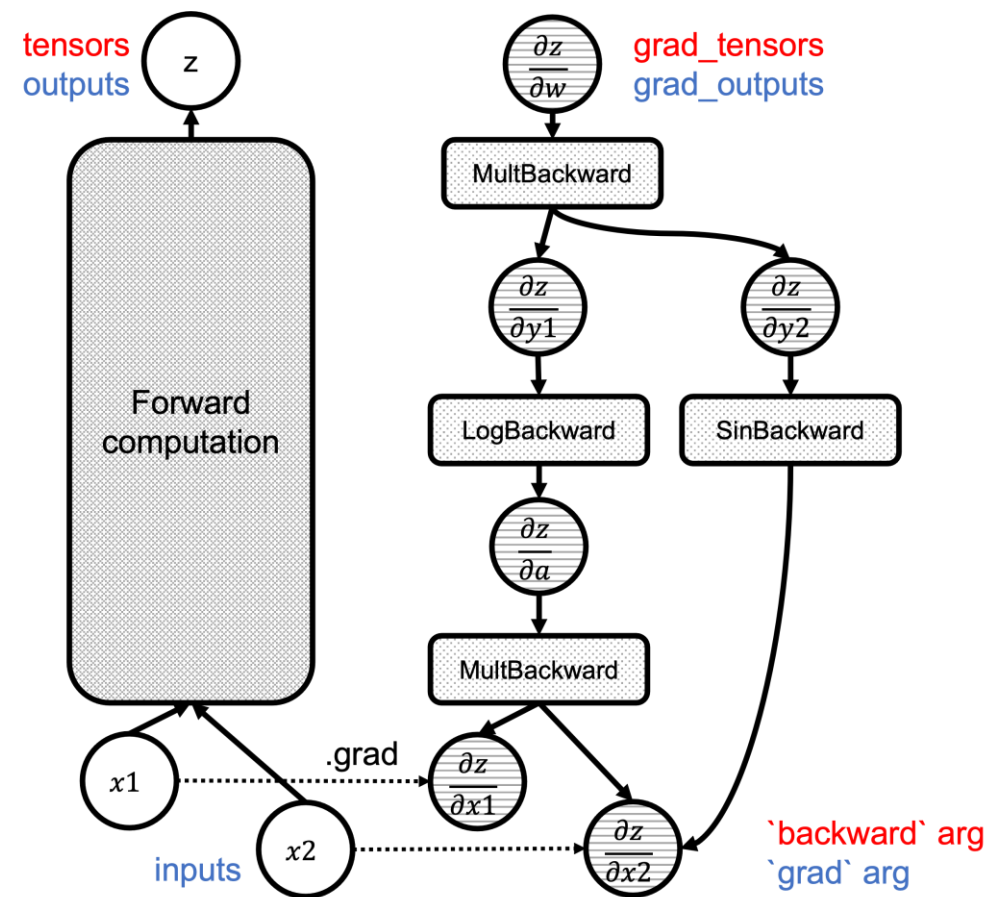


```
0.4008 FPS
0.5223 FPS
0.5262 FPS
0.5200 FPS
0.5213 FPS
0.5262 FPS
0.5253 FPS
0.5318 FPS
0.5243 FPS
0.5233 FPS

total: 0.5087909532277789 FPS
```

평균 FPS(PyTorch)

Predict on CPU: PyTorch



Computational Graph (DAG)

Export YOLO

예

Python

CLI

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolo11n.pt") # load an official model
model = YOLO("path/to/best.pt") # load a custom trained model

# Export the model
model.export(format="onnx")
```

형식	format	인수	모델	메타데이터	인수
PyTorch	-		yolo11n.pt	✓	-
TorchScript	torchscript		yolo11n.torchscript	✓	imgsz, optimize, batch
ONNX	onnx		yolo11n.onnx	✓	imgsz, half, dynamic, simplify, opset, batch
OpenVINO	openvino		yolo11n_openvino_model/	✓	imgsz, half, dynamic, int8, batch
TensorRT	engine		yolo11n.engine	✓	imgsz, half, dynamic, simplify, workspace, int8, batch
CoreML	coreml		yolo11n.mlpackage	✓	imgsz, half, int8, nms, batch
TF SavedModel	saved_model		yolo11n_saved_model/	✓	imgsz, keras, int8, batch
TF GraphDef	pb		yolo11n.pb	✗	imgsz, batch
TF Lite	tflite		yolo11n.tflite	✓	imgsz, half, int8, batch
TF Edge TPU	edgetpu		yolo11n_edgetpu.tflite	✓	imgsz
TF.js	tfjs		yolo11n_web_model/	✓	imgsz, half, int8, batch
PaddlePaddle	paddle		yolo11n_paddle_model/	✓	imgsz, batch
MNN	mnn		yolo11n.mnn	✓	imgsz, batch, int8, half
NCNN	ncnn		yolo11n_ncnn_model/	✓	imgsz, half, batch
IMX500	imx		yolov8n_imx_model/	✓	imgsz, int8

Predict on CPU: ONNX|OpenVINO|NCNN

```
Using ONNX Runtime CPU
0.5681 FPS
0.8732 FPS
0.8500 FPS
0.8787 FPS
0.8606 FPS
0.8964 FPS
0.8730 FPS
0.8846 FPS
0.8607 FPS
0.8298 FPS

total: 0.8237 FPS
```

ONNX

```
Using OpenVINO Extension
0.7839 FPS
2.1359 FPS
2.1437 FPS
2.1174 FPS
2.1519 FPS
2.1115 FPS
2.1542 FPS
2.1340 FPS
2.1394 FPS
2.1228 FPS

total: 1.8207 FPS
```

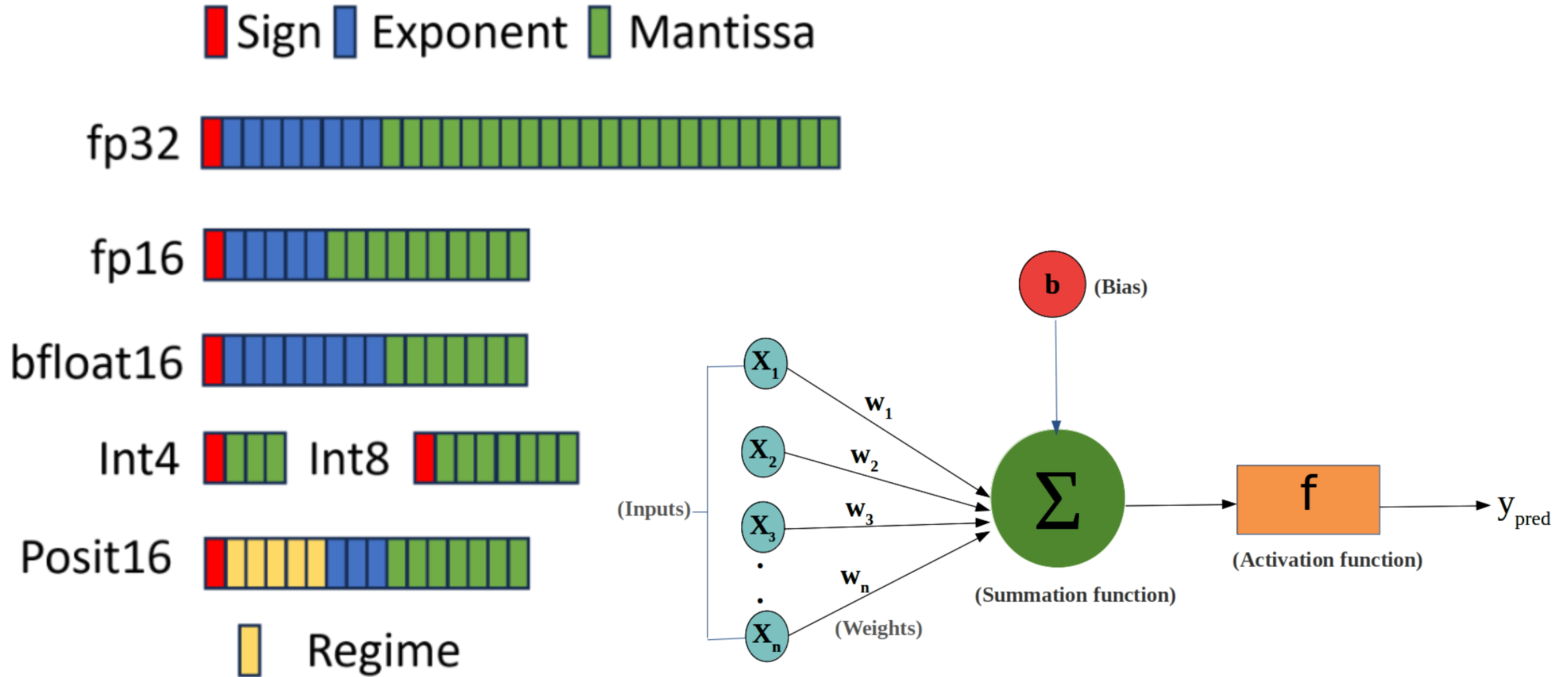
OpenVINO

```
Loading models/yolov10-11
0.9185 FPS
1.6920 FPS
1.8478 FPS
1.6322 FPS
1.7833 FPS
1.5181 FPS
2.7770 FPS
2.0623 FPS
1.6696 FPS
2.0513 FPS

total: 1.6710 FPS
```

NCNN

Predict on CPU: Quantization



Predict on CPU: Quantization

인수	유형	기본값	설명
format	str	'torchscript'	내보낸 모델의 대상 형식은 다음과 같습니다. 'onnx', 'torchscript', 'tensorflow' 등을 사용하여 다양한 배포 환경과의 호환성을 정의합니다.
imgsz	int 또는 tuple	640	모델 입력에 사용할 원하는 이미지 크기입니다. 정사각형 이미지의 경우 정수 또는 튜플일 수 있습니다. (height, width) 를 입력합니다.
keras	bool	False	에 대해 Keras 형식으로 내보내기를 활성화합니다. TensorFlow SavedModel 로 내보낼 수 있도록 설정하여 TensorFlow 서비스 및 API와의 호환성을 제공합니다.
optimize	bool	False	TorchScript 으로 내보낼 때 모바일 장치에 대한 최적화를 적용하여 모델 크기를 줄이고 성능을 개선할 수 있습니다.
half	bool	False	FP16(반정밀) 양자화를 활성화하여 모델 크기를 줄이고 지원되는 하드웨어에서 추론 속도를 높일 수 있습니다.
int8	bool	False	INT8 양자화를 활성화하여 모델을 더욱 압축하고 주로 에지 디바이스의 경우 정확도 손실을 최소화하면서 추론 속도를 높입니다.
dynamic	bool	False	ONNX, TensorRT 및 OpenVINO 내보내기에 대한 동적 입력 크기를 허용하여 다양한 이미지 크기를 유연하게 처리할 수 있습니다.
simplify	bool	True	다음을 사용하여 ONNX 내보내기에 대한 모델 그래프를 단순화합니다. onnxslim 를 사용하여 성능과 호환성을 개선할 수 있습니다.
opset	int	None	다른 ONNX 파서 및 런타임과의 호환성을 위해 ONNX 옴셋 버전을 지정합니다. 설정하지 않으면 지원되는 최신 버전을 사용합니다.
workspace	float 또는 None	None	TensorRT 최적화를 위한 최대 작업 공간 크기를 GiB 단위로 설정하여 메모리 사용량과 성능의 균형을 맞춥니다. None 를 통해 최대 장치까지 TensorRT 로 자동 할당할 수 있습니다.
nms	bool	False	정확하고 효율적인 탐지 후처리를 위해 필수적인 비최대 억제(NMS)를 CoreML 내보내기에 추가합니다.
batch	int	1	내보내기 모델 일괄 추론 크기 또는 내보낸 모델이 동시에 처리할 최대 이미지 수를 지정합니다. predict 모드로 전환합니다.
device	str	None	내보낼 장치를 지정합니다: GPU (device=0), CPU (device=cpu), 애플 실리콘의 경우 MPS (device=mps) 또는 DLA for NVIDIA Jetson(device=dla:0 또는 device=dla:1).

Predict on CPU: Quantization

형식	format 인수	모델	메타데이터	인수
PyTorch	-	yolo11n.pt	✓	-
TorchScript	torchscript	yolo11n.torchscript	✓	imgsz, optimize, batch
ONNX	onnx	yolo11n.onnx	✓	imgsz, half, dynamic, simplify, opset, batch
OpenVINO	openvino	yolo11n_openvino_model/	✓	imgsz, half, dynamic, int8, batch
TensorRT	engine	yolo11n.engine	✓	imgsz, half, dynamic, simplify, workspace, int8, batch
CoreML	coreml	yolo11n.mlpackage	✓	imgsz, half, int8, nms, batch
TF SavedModel	saved_model	yolo11n_saved_model/	✓	imgsz, keras, int8, batch
TF GraphDef	pb	yolo11n.pb	✗	imgsz, batch
TF Lite	tflite	yolo11n.tflite	✓	imgsz, half, int8, batch
TF Edge TPU	edgetpu	yolo11n_edgetpu.tflite	✓	imgsz
TF.js	tfjs	yolo11n_web_model/	✓	imgsz, half, int8, batch
PaddlePaddle	paddle	yolo11n_paddle_model/	✓	imgsz, batch
MNN	mnn	yolo11n.mnn	✓	imgsz, batch, int8, half
NCNN	ncnn	yolo11n_ncnn_model/	✓	imgsz, half, batch
IMX500	imx	yolov8n_imx_model/	✓	imgsz, int8

Predict on CPU: Quantization

Supported Model Precision

CPU plugin supports the following data types as inference precision of internal primitives:

- Floating-point data types:
 - **FP32** (Intel® x86-64, Arm®)
 - **BF16** (Intel® x86-64)
 - **FP16** (Intel® x86-64, Arm®)
 - **:ref: `MXFP4 <mxfp4_support>`** (Intel® x86-64)
- Integer data types:
 - **INT32** (Intel® x86-64, Arm®)
- Quantized data types:
 - **uINT8** (Intel® x86-64)
 - **INT8** (Intel® x86-64)
 - **uINT1** (Intel® x86-64)

```
Using OpenVINO LATEC
0.5388 FPS
1.1872 FPS
1.1884 FPS
1.1888 FPS
1.1888 FPS
1.1843 FPS
1.1833 FPS
1.1848 FPS
1.1992 FPS
1.1992 FPS
```

OpenVINO
(INT8 Quantized)

NPU를 사용해보자!



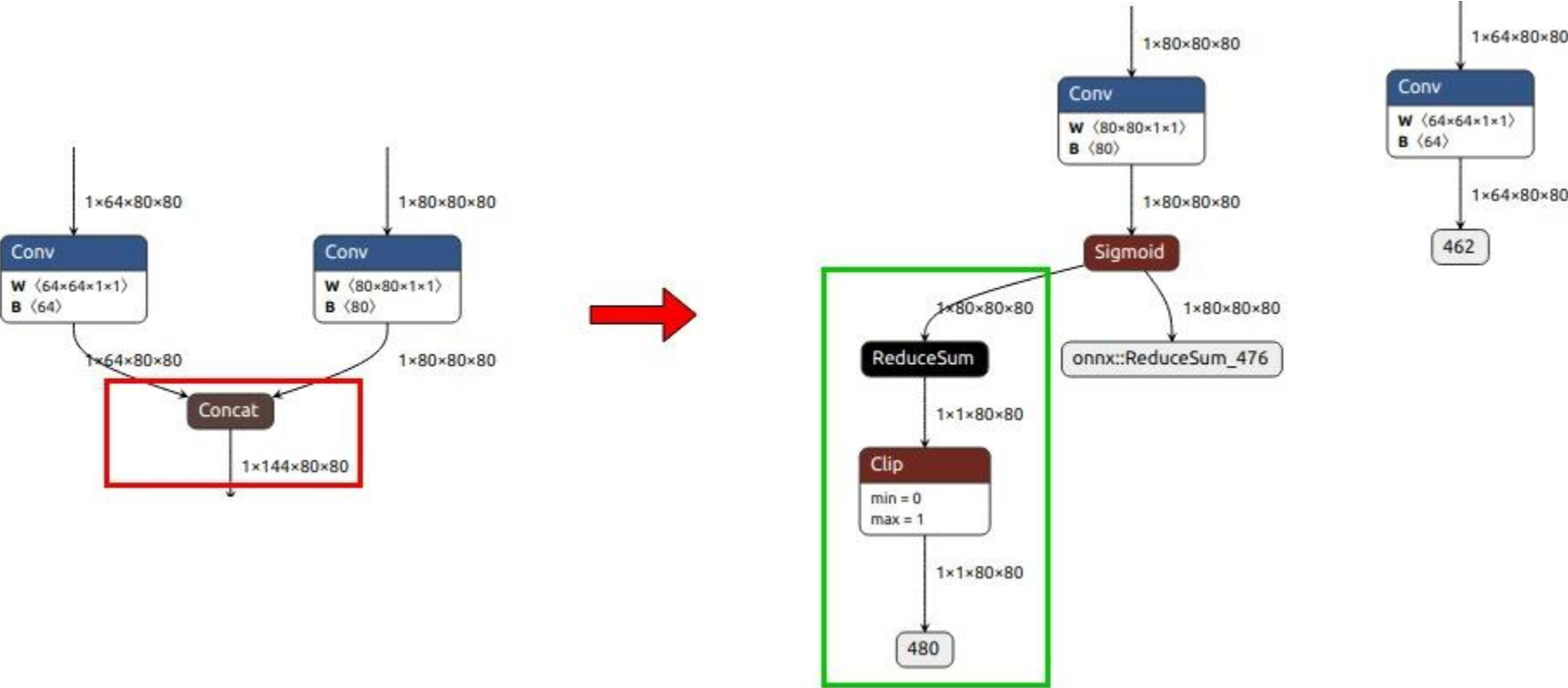
RKNN SDK

RKNN Toolkit2
(Quantization, Convert)

RKNN Toolkit Lite 2
(Inference, Test)

RKNPU2
(Row level API)

Predict on NPU: RKNN Optimization



```

from time import time
41 from utils.yolo11 import YOLO11
40 from utils.wrappers import ModelWrapper
39 from utils.camera import setup_camera
38 import cv2
37 import os
36
35 MODEL_PATH = './models/yolo11n.rknn'
34 CAM_WIDTH = 640
33 CAM_HEIGHT = 640
32
31 if __name__ == '__main__':
30     model, _ = ModelWrapper.setup(MODEL_PATH)
29     cap = setup_camera(CAM_WIDTH, CAM_HEIGHT)
28     yolo11 = YOLO11(model)
27     frames, init_time, loop_time = 0, time(), time()
26
25     os.system('clear')
24     while cv2.waitKey(1) < 0:
23         frames += 1
22         status, frame = cap.read()
21         if not status:
20             break
19         result_img, pos = yolo11.detect_largest_object(frame)
18         cv2.imshow('Webcam', result_img)
17         if frames % 30 == 0:
16             cur = time()
15             if frames == 30:
14                 print(f"{30 / (cur - loop_time):.4f} FPS (warm-up)")
13                 init_time = cur
12                 loop_time = cur
11                 continue
10
9                 print(f"{30 / (cur - loop_time):.4f} FPS")
8                 loop_time = cur
7                 if frames == 330:
6                     break
5
4         print(f"\ntotal: {(frames - 30) / (time() - init_time):.4f} FPS (after warm-up)")
3
2         cap.release()
1         cv2.destroyAllWindows()
0         model.release()

```

```

def detect_largest_object(self, input):
    img, _, _ = self.resize_image(input)
    outputs = self.model.run(img)
    boxes, _, scores = self.post_process(outputs)

    if boxes is None or scores is None:
        return img, None

    idx, max_size = -1, 0
    for i, box in enumerate(boxes):
        _, _, w, h = box.astype(int)
        size = w * h
        if size > max_size:
            max_size = size
            idx = i

    if idx == -1:
        return img, None

    dstimg, center_x, center_y = self.draw(img, boxes[idx], scores[idx])
    return dstimg, (center_x, center_y)

```

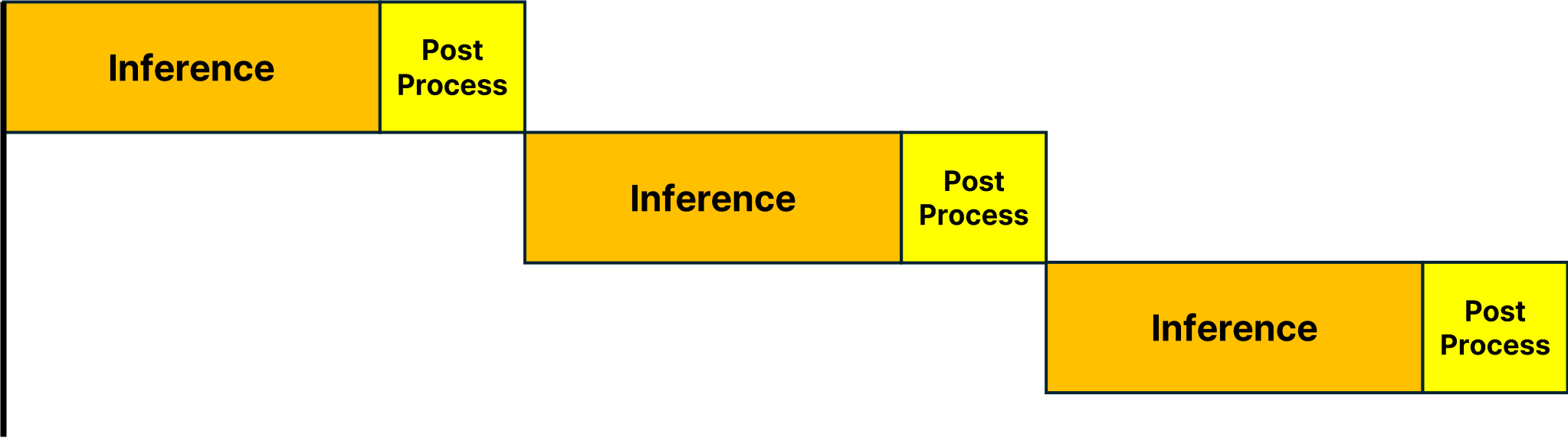

Predict on NPU: RKNN

```
4.6187 FPS (warm-up)
10.7755 FPS
10.7906 FPS
10.6942 FPS
10.7239 FPS
10.7039 FPS
10.6407 FPS
10.1629 FPS
10.7041 FPS
10.6971 FPS
10.7354 FPS

total: 10.6599 FPS (after warm-up)
```

RKNN
(INT8 Quantized)

Predict on NPU: RKNN



```

44 WORKER_N = 4
43
42
41 if __name__ == '__main__':
40     cap = setup_camera(CAM_WIDTH, CAM_HEIGHT)
39     pool = RKNNPoolExecutor(MODEL_PATH, WORKER_N, func)
38     if cap.isOpened():
37         for i in range(WORKER_N + 1):
36             ret, frame = cap.read()
35             if not ret:
34                 cap.release()
33                 del pool
32                 exit(-1)
31             pool.put(frame)
30     frames, loop_time, init_time = 0, time(), time()
29
28     os.system('clear')
27     while cap.isOpened():
26         frames += 1
25         ret, frame = cap.read()
24         if not ret:
23             break
22         pool.put(frame)
21         frame, _ = pool.get()
20         if frame is None:
19             break
18         cv2.imshow('rknn', frame)
17         if cv2.waitKey(1) & 0xFF == ord('q'):
16             break
15         if frames % 30 == 0:
14             cur = time()
13             if frames == 30:
12                 print(f"{30 / (cur - loop_time):.4f} FPS (warm-up)")
11                 init_time = cur
10                 loop_time = cur
9                 continue
8                 print(f"{30 / (time() - loop_time)} FPS")
7                 loop_time = time()
6             if frames == 330:
5                 break
4
3     print(f"\ntotal: {(frames - 30) / (time() - init_time):.4f} FPS (after warm-up)")

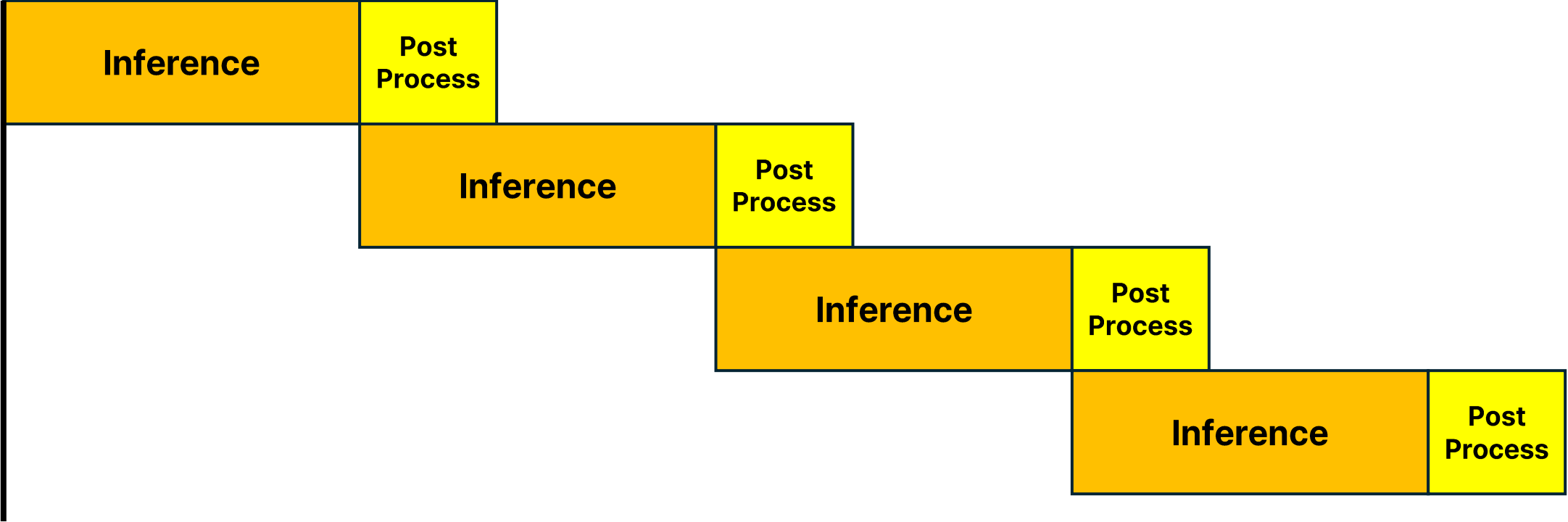
```

```

32
31 def init_rknn(model_path: str, N: int):
30     rknn = []
29     for _ in range(N):
28         rknn.append(RKNNWrapper(model_path))
27     return rknn
26
25
24 class RKNNPoolExecutor():
23     def __init__(self, model_path: str, N: int, func):
22         self.queue = Queue()
21         self.rknn = init_rknn(model_path, N)
20         self.pool = ThreadPoolExecutor(max_workers=N)
19         self.func = func
18         self.num = 0
17         self.N = N
16
15     def put(self, frame: MatLike):
14         future = self.pool.submit(self.func, self.rknn[self.num % self.N], frame)
13         self.queue.put(future)
12         self.num += 1
11
10     def get(self):
9         if self.queue.empty():
8             return None, None
7         future = self.queue.get()
6         result = future.result()
5         return result
4
3     def release(self):
2         self.pool.shutdown()
1         for rknn in self.rknn:
40             rknn.release()

```

Predict on NPU: RKNN



Predict on NPU: Thread Pool

```
4.6187 FPS (warm-up)
10.7755 FPS
10.7906 FPS
10.6942 FPS
10.7239 FPS
10.7039 FPS
10.6407 FPS
10.1629 FPS
10.7041 FPS
10.6971 FPS
10.7354 FPS

total: 10.6599 FPS (after warm-up)
```

RKNN
(Single-Thread)

```
6.3787 FPS (warm-up)
15.35476116242078 FPS
16.182933374282886 FPS
14.724228587436587 FPS
17.007263042822434 FPS
15.10413680748915 FPS
15.629952480041021 FPS
17.011183278496905 FPS
15.370820742580868 FPS
15.241264373649665 FPS
15.100203505556621 FPS

total: 15.6357 FPS (after warm-up)
```

RKNN
(Multi-Thread)