

Results for the different object detection models with OpenVino:

Name	Average FPS	Inference	RAM	CPU	GPU
Yolov11n	19.43	56.29 ms	22.97%	22.44%	40%
Yolov11s	13.04	81.22 ms	24.80%	19.96%	37%
Yolov8n	19.45	56.58 ms	24.45 %	22.41%	28%
Yolov8s	12.13	86.82 ms	24.58%	19.90%	33%
Yolov5nu	19.86	55.53 ms	23.84%	22.43%	25%
Yolov5nu with ONNX	37.18	27.31 ms	24.41%	38.10%	Check video
Yolov8n with ONNX	32.97	30.74	25.78	37.79	Added the video as well
Yolo11n with OONX	33.32	30.52	25.85	40.02	Video as well

Video link for Yolov5nu with ONNX:

https://drive.google.com/file/d/1iL_ywfwc7T-W1vhJ-YPABEit7BEaZz6O/view?usp=sharing

Video link for Yolov8n with ONNX:

<https://drive.google.com/file/d/1JT44sL5t1KTlxmckvDy3aAo6tNIQ4kIK/view?usp=sharing>

Video link for Yolov11n with ONNX:

<https://drive.google.com/file/d/1HToL5vplEi4S8POeSXMGClduZGUqSMBR/view?usp=sharing>

May 15 10:46

username@hostname: ~

```
Intel-gpu-top: Intel Alderlake_n (Gen12) @ /dev/dri/card1 - 312/ 768 MHz
40% RC6; 0.93/ 3.41 W; 211 irq/s/s
```

ENGINES		BUSY		NI		SEMA		NI		WAIT	
Render3D	25.72%							0%		0%	
Blitter	0.00%							0%		0%	
Video	0.00%							0%		0%	
VideoEnhance	0.00%							0%		0%	

PID	Render3D	Blitter	Video	VideoEnhance	NAME
6967					python3
1515					systemd
4108					gnome-shell
2485					nautilus
2422					TeamViewer
1932					xdg-desktop-por
1547					snaps-desktop-l
					mutter-x11-frm

username@hostname: ~

```
@[|||||] 39.4% Tasks: 139, 554 thr, 101 kthr, 4 runn
41.7% Load average: 1.75 1.64 1.64
46.3% Uptime: 04:31:36
@[|||||] 37.4%
Mon[|||||] 12.32/11.40
Swp[|] 192K/2.00G
```

Main 7/0

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU	MEM	TIME+	COMMAND
6967	username	20	0	63408	860M	394M	S	36.1	7.4	0:12.63	python3 openv
7017	username	20	0	63408	860M	394M	S	20.6	7.4	0:04.94	python3 openv
7018	username	20	0	63408	860M	394M	S	20.0	7.4	0:04.97	python3 openv
1515	username	20	0	40408	256M	114M	S	18.7	2.2	58:16.51	/usr/bin/gnom
3160	username	20	0	2053M	110M	74612	S	17.4	0.9	26:59.90	/usr/bin/anyd
1268	username	20	0	871M	166M	95556	S	15.5	1.4	27:18.63	/usr/lib/xorg
1115	root	20	0	660M	2428	24856	S	5.9	0.2	5:40.10	/usr/bin/anyd
3156	username	20	0	2053M	110M	74612	S	4.5	0.9	4:51.42	/usr/bin/anyd
3789	username	20	0	850M	61124	44644	S	4.5	0.5	8:54.35	/usr/libexec/
6944	username	20	0	14168	4892	3484	R	4.5	0.0	0:05.19	htop
3168	username	20	0	2053M	110M	74612	S	3.9	0.9	0:43.34	/usr/bin/anyd
3162	username	20	0	2053M	110M	74612	S	1.9	0.9	2:31.52	/usr/bin/anyd
3167	username	20	0	2053M	110M	74612	S	1.9	0.9	1:02.92	/usr/bin/anyd

F1 help F2 status F3 search F4 filter F5 tree F6 sort by F7 view F8 view F9 view F10 view F11 view F12 view

username@hostname: ~/freqs/freqs_vemv

```
(1, 3, 640, 640)
Processed frame 162/187 | FPS: 18.28 | Inference: 54.70ms
```

0: 640x640 3 persons, 27.3ms
Speed: 4.8ms preprocess, 27.3ms inference, 1.7ms postprocess per image at shape (1, 3, 640, 640)
Processed frame 163/187 | FPS: 20.88 | Inference: 48.08ms

0: 640x640 3 persons, 42.9ms
Speed: 5.8ms preprocess, 42.9ms inference, 1.9ms postprocess per image at shape (1, 3, 640, 640)
Processed frame 164/187 | FPS: 15.42 | Inference: 64.87ms

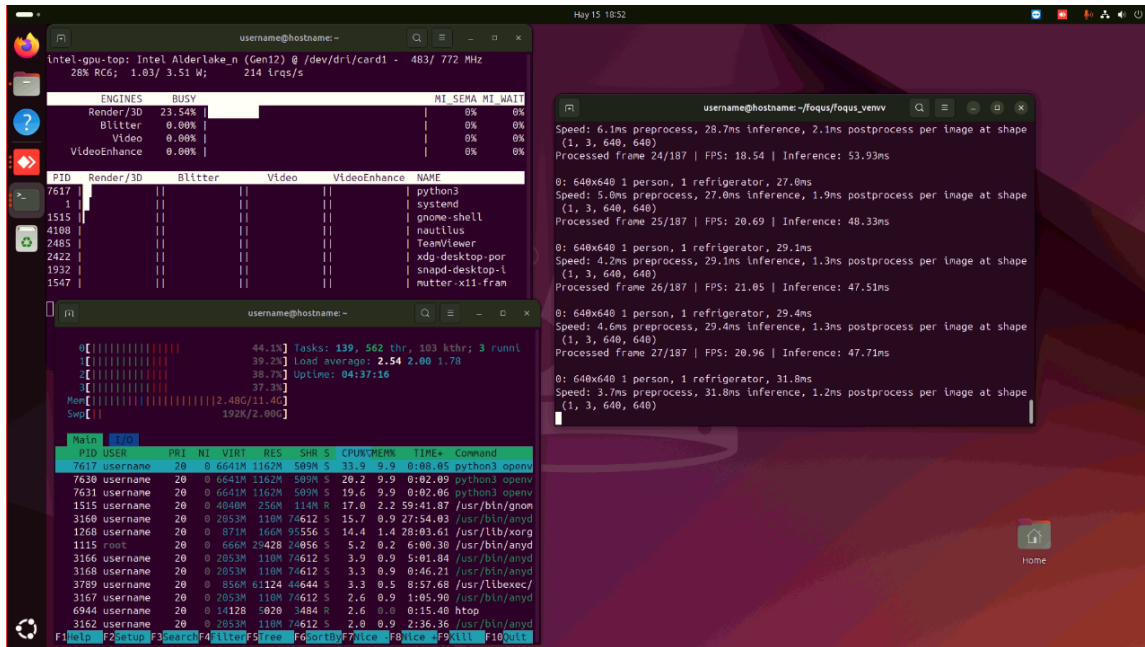
0: 640x640 3 persons, 27.0ms
Speed: 4.1ms preprocess, 27.0ms inference, 1.4ms postprocess per image at shape (1, 3, 640, 640)
Processed frame 165/187 | FPS: 22.57 | Inference: 44.38ms

0: 640x640 3 persons, 43.8ms
Speed: 7.2ms preprocess, 43.8ms inference, 1.2ms postprocess per image at shape (1, 3, 640, 640)
Processed frame 166/187 | FPS: 14.62 | Inference: 68.08ms

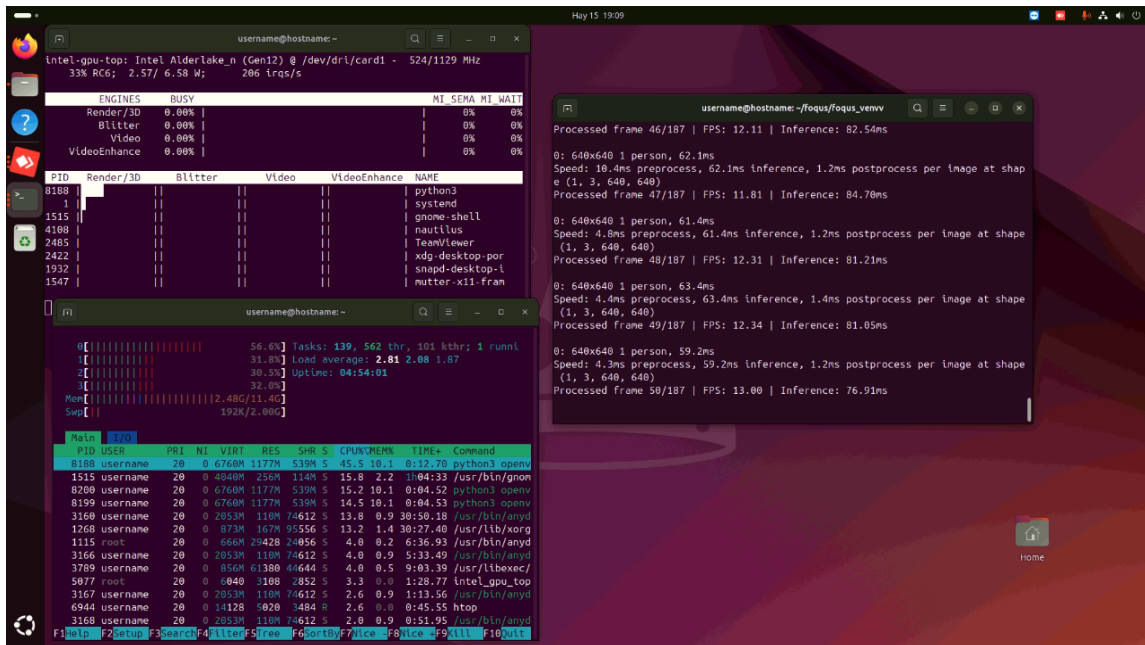
Home

The screenshot displays a Kali Linux desktop environment with three terminal windows open. The top window shows the output of the 'nvidia-smi' command, indicating the NVIDIA GeForce RTX 3090 is running at 37% power with a temperature of 24.1°C and a fan speed of 210 RPM. The middle window shows the output of the 'top' command, listing running processes such as python3, systemd, and sshd. The bottom window shows the output of the 'nvidia-smi' command, indicating the NVIDIA GeForce RTX 3090 is running at 37% power with a temperature of 24.1°C and a fan speed of 210 RPM.

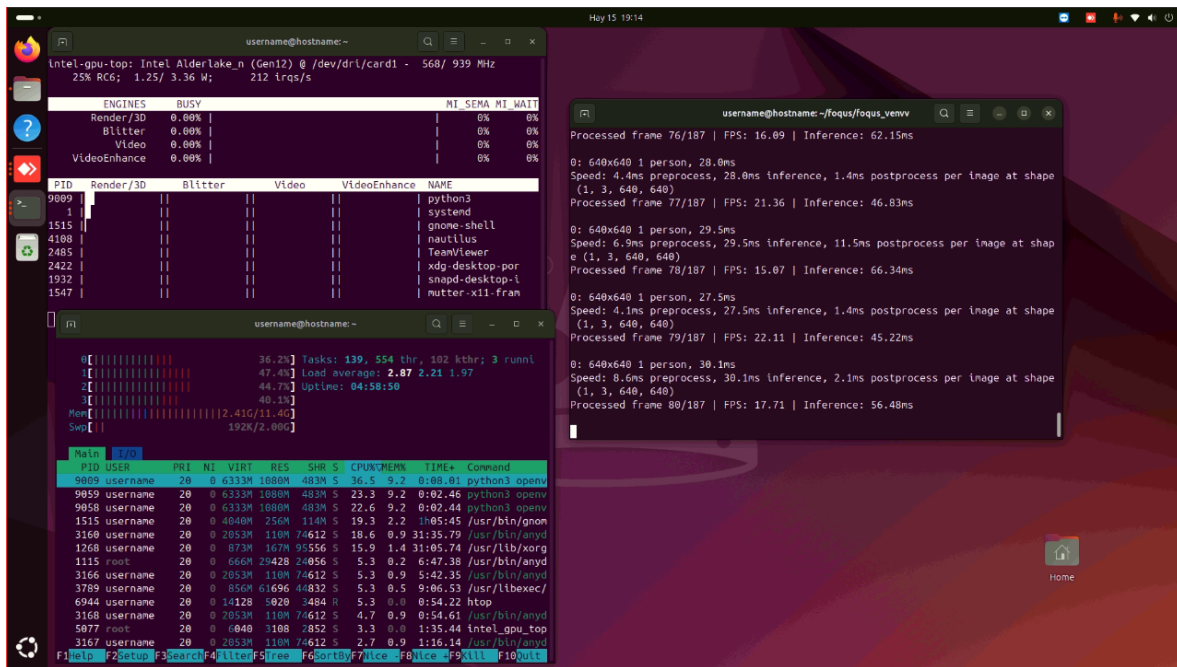
Yolov8n:



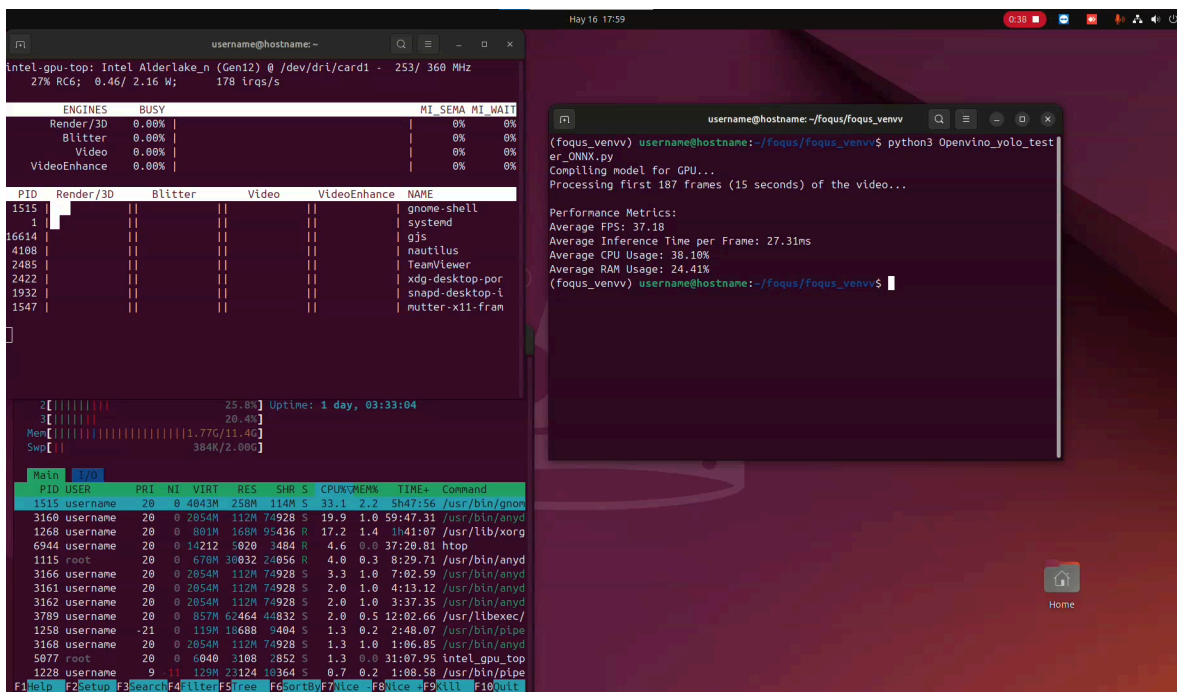
Yolov8s:



Yolov5nu:



YoloV5nu with ONNX format:



The script I used without onnx format:

```
import cv2
import numpy as np
from pathlib import Path
import time
import psutil
import GPUtil
from ultralytics import YOLO
import openvino as ov

# Function to get GPU load
def get_gpu_load():
    gpus = GPUtil.getGPUs()
    if gpus:
        return gpus[0].load * 100 # GPU load as a percentage
    return 0

# Function to get CPU and RAM usage
def get_system_usage():
    cpu_usage = psutil.cpu_percent(interval=0.1)
    ram_usage = psutil.virtual_memory().percent
    return cpu_usage, ram_usage

# Initialize variables for tracking performance
start_time = time.time()
frame_count = 0
total_inference_time = 0
fps_list = []
cpu_usage_list = []
ram_usage_list = []
gpu_load_list = []

# Load the YOLO model
model_name = "yolov5nu"
DET_MODEL_NAME = model_name
det_model = YOLO(f"{DET_MODEL_NAME}.pt")
det_model.to("cpu")

# Convert model to OpenVINO IR if not already converted
```

```

det_model_path =
Path(f"{DET_MODEL_NAME}_openvino_model/{DET_MODEL_NAME}.xml")
if not det_model_path.exists():
    det_model.export(format="openvino", dynamic=True, half=True)

# Load the OpenVINO model
device = "GPU"
core = ov.Core()
det_ov_model = core.read_model(det_model_path)

# Configure OpenVINO model
ov_config = {}
if device != "CPU":
    det_ov_model.reshape({0: [1, 3, 640, 640]})
if "GPU" in device or ("AUTO" in device and "GPU" in
core.available_devices):
    ov_config = {"GPU_DISABLE_WINOGRAD_CONVOLUTION": "YES"}
det_compiled_model = core.compile_model(det_ov_model, device, ov_config)

# Re-instantiate YOLO with OpenVINO
det_model = YOLO(det_model_path.parent, task="detect")
if det_model.predictor is None:
    custom = {"conf": 0.25, "batch": 1, "save": False, "mode": "predict"}
    args = {**det_model.overrides, **custom}
    det_model.predictor =
det_model._smart_load("predictor") (overrides=args,
    _callbacks=det_model.callbacks)
    det_model.predictor.setup_model(model=det_model.model)
det_model.predictor.model.ov_compiled_model = det_compiled_model

# Open the video file
video_name = "test.mp4"
cap = cv2.VideoCapture(video_name)
if not cap.isOpened():
    raise Exception(f"Error: Could not open video file {video_name}")

fps = cap.get(cv2.CAP_PROP_FPS)
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

```

```
# Process only the first 1 minute of the video
max_frames = int(fps * 15) # 1 minute of frames
print(f"Processing first {max_frames} frames (1 minute) of the video...")

try:
    while frame_count < max_frames:
        ret, frame = cap.read()
        if not ret:
            break # Exit if video ends before 1 minute

        # Resize the frame to the model's expected input shape (640x640)
        resized_frame = cv2.resize(frame, (640, 640))

        # Start inference timer
        inference_start_time = time.time()

        # Perform object tracking
        results = det_model.track(resized_frame, show=False)

        # Calculate inference time
        inference_time = time.time() - inference_start_time
        total_inference_time += inference_time

        # Calculate FPS
        current_fps = 1.0 / inference_time
        fps_list.append(current_fps)

        # Get system usage
        cpu_usage, ram_usage = get_system_usage()
        cpu_usage_list.append(cpu_usage)
        ram_usage_list.append(ram_usage)

        # Get GPU load
        gpu_load = get_gpu_load()
        gpu_load_list.append(gpu_load)

        # Increment frame count
        frame_count += 1

        # Print progress
```

```

        print(f"Processed frame {frame_count}/{max_frames} | FPS:
{current_fps:.2f} | Inference: {inference_time * 1000:.2f}ms")

except KeyboardInterrupt:
    print("Process interrupted by user.")

finally:
    # Release the video capture object
    cap.release()

    # Calculate average performance metrics
    if frame_count > 0:
        average_fps = sum(fps_list) / len(fps_list)
        average_inference_time = total_inference_time / frame_count
        average_cpu_usage = sum(cpu_usage_list) / len(cpu_usage_list)
        average_ram_usage = sum(ram_usage_list) / len(ram_usage_list)
        average_gpu_load = sum(gpu_load_list) / len(gpu_load_list)

        # Print the results
        print("\nPerformance Metrics:")
        print(f"Average FPS: {average_fps:.2f}")
        print(f"Average Inference Time per Frame: {average_inference_time
* 1000:.2f}ms")
        print(f"Average CPU Usage: {average_cpu_usage:.2f}%")
        print(f"Average RAM Usage: {average_ram_usage:.2f}%")
        print(f"Average GPU Load: {average_gpu_load:.2f}%")
    else:
        print("No frames processed.")

```


Script for ONNX format:

```
import cv2
import numpy as np
from pathlib import Path
import time
import psutil
import subprocess
from ultralytics import YOLO
import opencvino as ov

def get_system_usage():
    cpu_usage = psutil.cpu_percent(interval=0.1)
    ram_usage = psutil.virtual_memory().percent
    return cpu_usage, ram_usage

start_time = time.time()
frame_count = 0
total_inference_time = 0
fps_list = []
cpu_usage_list = []
ram_usage_list = []

model_name = "yolov5nu" # Change to your desired YOLO model
DET_MODEL_NAME = model_name
det_model = YOLO(f"{DET_MODEL_NAME}.pt")
det_model.to("cpu")

onnx_model_path = Path(f"{DET_MODEL_NAME}.onnx")
if not onnx_model_path.exists():
    print("Exporting YOLO model to ONNX format...")
    det_model.export(format="onnx", dynamic=True, half=True)
device = "GPU" # Use "CPU" for edge devices without GPU
core = ov.Core()
if device not in core.available_devices:
    print(f"Device {device} is not available. Available devices:
{core.available_devices}")
```

```

    device = "CPU" # Fallback to CPU
    print(f"Falling back to {device}.")
det_ov_model = core.read_model(onnx_model_path)

ov_config = {}
if device != "CPU":
    det_ov_model.reshape({0: [1, 3, 640, 640]}) # Reshape for GPU
if "GPU" in device or ("AUTO" in device and "GPU" in
core.available_devices):
    ov_config = {"GPU_DISABLE_WINOGRAD_CONVOLUTION": "YES"} # Optimize
for GPU
print(f"Compiling model for {device}...")
det_compiled_model = core.compile_model(det_ov_model, device, ov_config)
input_layer = det_compiled_model.input(0)
output_layer = det_compiled_model.output(0)

# Open the video file
video_name = "test.mp4" # Change it to your video file
cap = cv2.VideoCapture(video_name)
if not cap.isOpened():
    raise Exception(f"Error: Could not open video file {video_name}")

fps = cap.get(cv2.CAP_PROP_FPS)
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Process only the first 15 seconds of the video
max_frames = int(fps * 15) # Change as needed
print(f"Processing first {max_frames} frames (15 seconds) of the
video...")

try:
    while frame_count < max_frames:
        ret, frame = cap.read()
        if not ret:
            break # Exit if video ends
        resized_frame = cv2.resize(frame, (640, 640))
        input_data = resized_frame.transpose(2, 0, 1) # HWC to CHW

```

```

        input_data = np.expand_dims(input_data, axis=0) # Add batch
dimension
        input_data = input_data.astype(np.float32) / 255.0 # Normalize to
[0, 1]

        inference_start_time = time.time()
        results = det_compiled_model([input_data])[output_layer]
        inference_time = time.time() - inference_start_time
        total_inference_time += inference_time
        current_fps = 1.0 / inference_time
        fps_list.append(current_fps)
        cpu_usage, ram_usage = get_system_usage()
        cpu_usage_list.append(cpu_usage)
        ram_usage_list.append(ram_usage)
        frame_count += 1

        # Print progress or you can comment this out
        print(f"Processed frame {frame_count}/{max_frames} | FPS:
{current_fps:.2f} | "
              f"Inference: {inference_time * 1000:.2f}ms")

except KeyboardInterrupt:
    print("Process interrupted by user.")

finally:
    cap.release()

    # Calculate average performance metrics
    if frame_count > 0:
        average_fps = sum(fps_list) / len(fps_list)
        average_inference_time = total_inference_time / frame_count
        average_cpu_usage = sum(cpu_usage_list) / len(cpu_usage_list)
        average_ram_usage = sum(ram_usage_list) / len(ram_usage_list)

        # Print the results
        print("\nPerformance Metrics:")
        print(f"Average FPS: {average_fps:.2f}")
        print(f"Average Inference Time per Frame: {average_inference_time
* 1000:.2f}ms")
        print(f"Average CPU Usage: {average_cpu_usage:.2f}%")

```

```
    print(f"Average RAM Usage: {average_ram_usage:.2f}%")
else:
    print("No frames processed.")
```