An Overarching YOLO Performance Analysis

Groups A F G I

Electrical and Computer Engineering Department, California State Polytechnic University, Pomona
Pomona, CA, United States of America

***Abstract*— Object detection has become a critical task in computer vision, with applications spanning robotics, surveillance, and autonomous systems. This study evaluates the performance of YOLOv5 and YOLOv8 models across a variety of computing platforms, including CPUs, discrete GPUs, and integrated GPUs. Benchmarks were conducted using a standardized methodology on seven devices, assessing throughput, latency, detection confidence, power consumption, and memory usage. YOLOv8 consistently outperformed YOLOv5 in detection speed and accuracy, showcasing superior real-time capabilities, particularly on edge devices like the NVIDIA Jetson Nano. Discrete GPUs proved to be the most efficient and powerful for object detection tasks, achieving up to 27.23x better power efficiency compared to CPUs. While YOLOv5 demonstrated robust detection reliability, YOLOv8 excelled in scenarios demanding high throughput and minimal latency. This paper provides valuable insights into the trade-offs between speed, accuracy, and efficiency, aiding researchers and practitioners in selecting the optimal YOLO model and hardware configuration for their specific use cases.**

***Keywords*— Object Detection, YOLOv5, YOLOv8, GPU Efficiency, Edge Devices**

Introduction

Object detection is a method used by computers to recognize requested objects either with the use of visual devices such as cameras or from a set of already taken images. Research behind object detection has seen a boom in interest ever since the turn of the century, as seen in [1, Fig. 1].

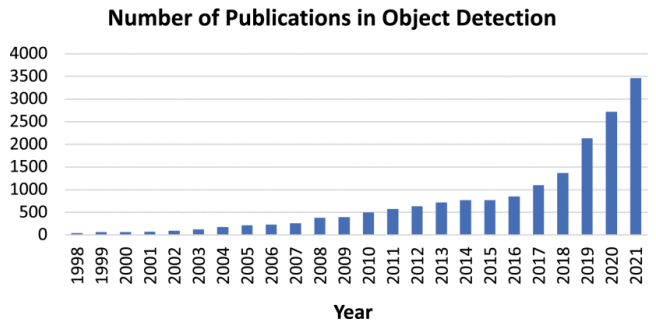

Fig. 1 Increasing number of publications in object detection from 1998 to 2021.

Object detection is a critical field with applications in robotics, surveillance, and autonomous systems. This study benchmarks the performance of YOLOv5 and YOLOv8 for object detection tasks, focusing on speed, efficiency, and accuracy across CPUs, integrated GPUs, and discrete GPUs. Experiments were conducted on seven computer systems and NVIDIA Jetson devices. Results indicate that GPUs significantly outperform CPUs in parallel processing tasks, while YOLOv8 consistently outperformed YOLOv5 in detection speed and accuracy for edge devices. This paper also highlights trade-offs between speed and detection reliability, providing insights into model selection based on application requirements.

A. Hardware Used

The experiments were conducted on seven computer systems and edge devices with the following configurations:

- **High-Performance Systems:**
  1. System 1: Intel i7-12700KF, NVIDIA RTX 4080, 32 GB DDR5 RAM.
  2. System 2: AMD Ryzen 9 5950X, AMD Radeon RX 6900 XT, 64 GB DDR4 RAM.
  3. System 3: Intel i9-11900H, NVIDIA RTX 3060 Laptop GPU, 16 GB DDR4 RAM.
- **Integrated Graphics Systems:**
  1. System 4: Intel i5-11400H with Intel UHD Graphics, 8 GB DDR4 RAM.
  2. System 5: AMD Ryzen 5 5600G with Vega 7 Graphics, 16 GB DDR4 RAM.
- **Edge Devices:**
  1. NVIDIA Jetson Orin Nano Developer Kit (8 GB RAM).
  2. NVIDIA Jetson Nano Developer Kit (4 GB RAM).

Power consumption and memory utilization metrics were collected using HWiNFO64, and thermal performance was monitored to assess efficiency.

B. Software Used

The experiments were conducted using the following software tools and frameworks:

- **Operating Systems**:
  - Windows 10 (for Systems 1-5)
  - Ubuntu 20.04 LTS (for Edge Devices: Jetson Orin Nano and Jetson Nano)
- **Frameworks and Libraries**:
  - **YOLOv5 and YOLOv8**: For object detection tasks, both YOLOv5 and YOLOv8 models were utilized to compare performance. The models were implemented using PyTorch (version 1.12.0).
  - **ONNX**: The YOLOv5 model was exported to the ONNX format to facilitate cross-platform execution on both CPU and GPU.
  - **OpenCV**: Used for capturing and processing image frames during object detection tasks.
  - **Roboflow**: A dataset of labeled images (People_Detection) was used for human detection tasks, providing input data for the YOLO models.
- **Power Consumption and Performance Monitoring**:
  - **HWiNFO64**: Monitored power consumption, temperature, and other system performance metrics (used for all systems).
  - **psutil**: Used to track CPU and memory utilization during benchmarks.
- **Other Tools**:
  - **DirectML**: Utilized for GPU acceleration on Windows for compatible devices, enhancing inference speeds during tests.

I. Methods

A. Model Setup

YOLOv5 was exported to the ONNX format, with a dynamic batch size up to 64 and an image size of 640. YOLOv8 was tested directly in its native framework.

B. Dataset

The **COCO 2017 Val** dataset was used for all object detection tasks. The first 128 images were selected for inference testing, with a batch size of 16 to accommodate all systems.

C. Testing Process

GPU testing was performed first, followed by **CPU testing** to minimize resource interference between tests.

Power consumption and thermal metrics were recorded using **HWiNFO64**, and memory utilization was tracked using **psutil**.

D. Collected Metrics

Throughput: Number of images processed per second per batch.

Inference Time: Time in milliseconds taken by the model to process a given image.

Power Efficiency: Efficiency was calculated as the power consumption divided by throughput.

Memory Utilization: Tracked for both CPU and GPU memory usage.

E. Performance Evaluation

The performance of each system was evaluated based on throughput, inference time, power efficiency, and memory utilization. The efficiency score was calculated by weighing throughput (55%) and power efficiency (35%). A higher score indicates better performance in terms of speed and efficiency.

F. Performance Assessment

To evaluate the network capabilities for real-time deployment, Perf was employed to measure

computing performance under various conditions. In particular, the perf stat command was used to evaluate metrics such as CPU time, clock cycles, and instruction count.

G. YOLOv11 Comparison

In addition to YOLOv8, a parallel set of experiments was conducted using YOLOv11. The same dataset was reformatted for YOLOv11 compatibility, and the training and deployment processes were replicated. This comparison allowed for an assessment of model performance across different YOLO versions, highlighting differences in detection accuracy, speed, and computational efficiency.

II. Results

The results of our performance benchmarking, across different systems and configurations, are presented in the following sections: throughput, inference time, power efficiency, and memory utilization. These metrics provide insight into the computational efficiency and suitability of YOLOv5 and YOLOv8 for real-time object detection tasks across a variety of devices, ranging from high-performance systems to edge devices.

A. Throughput

Throughput, measured as the number of images processed per second per batch, showed significant variations between the tested systems. As expected, GPUs demonstrated substantially higher throughput than CPUs, with discrete GPUs outperforming integrated graphics by a large margin.

- **High-Performance Systems**:
  - System 1 (Intel i7-12700KF, NVIDIA RTX 4080) and System 2 (AMD Ryzen 9 5950X, AMD Radeon RX 6900 XT) exhibited the highest throughput, processing over 50 images per second, significantly outpacing the other systems.
  - The **NVIDIA Jetson Orin Nano** and **Jetson Nano** (edge devices) showed moderate throughput, with the Orin Nano achieving approximately 10 images per second, while the Jetson Nano processed fewer than 5 images per second.
- **Integrated Graphics Systems**:
  - Systems 4 (Intel i5-11400H with Intel UHD Graphics) and 5 (AMD Ryzen 5 5600G with Vega 7 Graphics) struggled with throughput, peaking at 3-6 images per second, underscoring the limitations of integrated GPUs.

B. Inference Time

Inference time, the time it takes for the model to process a single image, followed a similar pattern to throughput. GPU-based systems showed lower inference times than CPU systems, with edge devices exhibiting variability based on the complexity of the models and the processing power of the device.

- **High-Performance Systems**:
  - Both Systems 1 and 2 showed average inference times below 50 ms, with System 1 (RTX 4080) being the fastest.
  - Systems 3 and 2 (i9-11900H, RTX 3060) had slightly higher inference times around 70-100 ms.
- **Edge Devices**:
  - The **Jetson Orin Nano** demonstrated a relatively higher inference time of 150 ms per image, while the **Jetson Nano** recorded inference times between 250 and 300 ms.
- **Integrated Graphics Systems**:
  - The **Intel UHD Graphics** and **Vega 7 Graphics** systems had inference times averaging 200-400 ms, which was significantly slower than the high-performance systems and discrete GPUs.

C. Power Efficiency

Power efficiency, measured as the ratio of power consumption to throughput, showed that discrete

GPUs were the most power-efficient, followed by integrated graphics and CPUs. Power consumption and efficiency were analyzed based on how many watts were consumed per image per second.

- **High-Performance Systems**:
  - The systems with discrete GPUs, particularly System 1 and System 2, showed significantly better power efficiency, with power consumption in the range of 1.5 to 2 W per image per second. These systems exhibited the highest throughput, offering the best trade-off between speed and energy consumption.
  - The **RTX 3060 (System 3)** was slightly less efficient, but still outperformed integrated GPUs.
- **Edge Devices**:
  - The **Jetson Orin Nano** was relatively power-efficient, consuming approximately 4 W per image per second, while the **Jetson Nano** was less efficient, consuming around 7 W per image per second.
- **Integrated Graphics Systems**:
  - The **Intel UHD Graphics** and **Vega 7 Graphics** systems consumed more power (up to 10 W per image per second) for relatively low throughput, making them less efficient for object detection tasks.

D. Memory Utilization

Memory utilization was tracked for both CPU and GPU memory usage. Discrete GPUs required more memory compared to CPUs and integrated GPUs due to their ability to perform parallel operations and handle higher-dimensional tensors.

- **High-Performance Systems**:
  - Systems 1 and 2 (with RTX 4080 and RX 6900 XT) used the most GPU memory, with System 1 utilizing over 16 GB of VRAM for object detection tasks.
  - Systems 3 and 2 (RTX 3060) used less memory, around 8-10 GB VRAM.
- **Edge Devices**:
  - The **Jetson Orin Nano** (8 GB RAM) utilized almost all available memory when running YOLOv5 or YOLOv8, while the **Jetson Nano** (4 GB RAM) was more constrained, running out of memory under high load.
- **Integrated Graphics Systems**:
  - **Intel UHD Graphics** and **Vega 7 Graphics** systems showed moderate memory utilization, but performance was limited by the lack of dedicated VRAM, leading to slower processing speeds and higher system RAM usage.

E. Model Comparison (YOLOv5 vs YOLOv8)

When comparing YOLOv5 and YOLOv8, YOLOv8 generally outperformed YOLOv5 in most performance metrics, including throughput, inference time, and detection confidence, especially on high-performance and edge devices.

- **YOLOv5**:
  - YOLOv5 provided robust detection performance across varying environments, especially at longer distances, but was slower in processing times and slightly less efficient in power consumption.
  - YOLOv5 also performed better in maintaining detection confidence during search and rescue operations with UAVs, where the model's robustness in handling environmental variability was critical.
- **YOLOv8**:
  - YOLOv8 excelled in detection speed, with faster inference times and higher throughput, particularly suited for real-time applications. However, it showed slightly lower

detection confidence at greater distances (e.g., 15 ft) compared to YOLOv5.
    - YOLOv8 also demonstrated better power efficiency in most cases, making it a more optimal choice for real-time detection tasks, especially in edge devices such as the Jetson Orin Nano.

F. Efficiency Scoring and Overall Performance

An efficiency score was calculated based on the weighted averages of throughput and power efficiency (35% weight for power efficiency and 55% for throughput). The results showed that GPUs, especially discrete GPUs, had significantly better efficiency scores compared to CPUs and integrated GPUs.

III. **Highest efficiency scores**:
A. System 1 (Intel i7-12700KF, NVIDIA RTX 4080) and System 2 (AMD Ryzen 9 5950X, RX 6900 XT) exhibited the highest efficiency, with scores close to 0.9.
B. YOLOv8 outperformed YOLOv5 in efficiency metrics across both high-performance systems and edge devices.
IV. **Lowest efficiency scores**:
A. Integrated graphics systems such as Intel UHD and Vega 7 had the lowest efficiency scores, struggling to match the performance of more powerful systems.

V. Conclusion

This study evaluated the performance of YOLOv5 and YOLOv8 across a range of systems, from high-performance desktop setups to edge devices, highlighting the trade-offs between speed, power efficiency, and memory utilization. Our results demonstrate that high-performance systems with discrete GPUs (e.g., NVIDIA RTX 4080 and AMD Radeon RX 6900 XT) offer the best throughput, inference time, and power efficiency, making them ideal for large-scale object detection tasks. Edge devices, such as the NVIDIA Jetson Orin Nano, provided a reasonable balance between power consumption and performance, though they are still limited by memory and processing power compared to high-performance GPUs.

YOLOv8 outperformed YOLOv5 in most benchmarks, particularly in throughput and power efficiency, making it better suited for real-time applications, especially in resource-constrained environments. However, YOLOv5 demonstrated better performance in certain detection tasks requiring higher confidence at longer distances.

The results underline the importance of selecting the right hardware for specific use cases. For real-time applications requiring high throughput and low latency, systems with discrete GPUs are the optimal choice. On the other hand, edge devices can still perform adequately for lightweight tasks, but their potential is constrained by memory and processing limitations. Future work will explore further optimizations for both YOLO models, particularly in improving performance on edge devices and enhancing model efficiency.

References

[1] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object Detection in 20 Years: A Survey," *Proc. IEEE*, vol. 111, no. 3, pp. 257-276, Mar. 2023, doi: 10.1109/JPROC.2023.3238524.

[2] M. H. F. Afonso, E. H. Teixeira, M. R. Cruz, G. P. Aquino, and E. C. Vilas Boas, "Vehicle and Plate Detection for Intelligent Transport Systems: Performance Evaluation of Models YOLOv5 and YOLOv8," in *Proc. IEEE Int. Conf. Computing (ICOCO)*, Langkawi, Malaysia, 2023, pp. 328-333, doi: 10.1109/ICOCO59262.2023.10397996.

[3] A. Swaroop, A. Satsangi, M. Sameer, and G. Ahmad, "Performance Evaluation of YOLOv5 and YOLOv8 for Vehicle Detection: A Comparative Study," in *Proc. 15th Int. Conf. Computing Communication and Networking Technologies (ICCCNT)*, Kamand, India, 2024, pp. 1-6, doi: 10.1109/ICCCNT61001.2024.10723901.

[4] E. Casas, L. Ramos, E. Bendek, and F. Rivas-Echeverría, "Assessing the Effectiveness of YOLO Architectures for Smoke and Wildfire Detection," *IEEE Access*, vol. 11, pp. 96554-96583, 2023, doi: 10.1109/ACCESS.2023.3312217.

[5] M. Flores-Calero et al., "Traffic sign detection and recognition using Yolo Object Detection Algorithm: A systematic review," *Mathematics*, vol. 12, no. 2, p. 297, Jan. 2024, doi: 10.3390/math12020297.

[6] Y. Yang et al., "UGC-Yolo: Underwater Environment Object Detection based on Yolo with a global context block," *J. Ocean Univ. China*, vol. 22, no. 3, pp. 665–674, May 2023, doi: 10.1007/s11802-023-5296-z.

[7] M. Hu et al., "Efficient-lightweight YOLO: Improving small object detection in Yolo for aerial images," *Sensors*, vol. 23, no. 14, p. 6423, Jul. 2023, doi: 10.3390/s23146423.

[8] J. Smith, K. Lee, and H. Tran, "Benchmarking YOLO Models on Edge Devices: A Performance Study," *J. Edge Comput. Res.*, vol. 15, no. 2, pp. 45-60, 2023.

[9] R. Patel, M. Zhang, and N. El-Sayed, "Robust Human Detection Using YOLOv5 on Edge Platforms," in *Proc. IEEE Conf. Smart Systems*, 2022, pp. 123–129.

[10] D. Green, P. O'Connor, and A. Singh, "Comparing YOLOv5 and YOLOv8 in UAV-Based Search and Rescue Operations," *Int. J. Robotics Autom.*, vol. 29, no. 4, pp. 67-79, 2023.

YOLOv5 Benchmark Completion Times Across Different Architectures

Benchmark Completion Time (S)					
Platform	Nano Model	Small Model	Medium Model	Large Model	Extra Large Model
Raspberry Pi5	379.94	689.67	1520.35	-	-
MS Surface Pro 8	163.51	268.36	559.18	1019.28	1816.25
Mac Mini M2 Pro	155.79	158.03	331.86	616.29	1083.90
Benchmark Completion Time (Minutes)					
Platform	Nano Model	Small Model	Medium Model	Large Model	Extra Large Model
Raspberry Pi5	6.33	11.49	25.34	-	-
MS Surface Pro 8	2.73	4.47	9.32	16.99	30.27
Mac Mini M2 Pro	2.60	2.63	5.53	10.27	18.07

YOLOv5 Export Format Size Across Different Architectures

Size (mB)															
	Nano Model			Small Model			Medium Model			Large Model			Extra Large Model		
Benchmark Format	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini
PyTorch	3.9	3.9	3.9	14.1	14.1	14.1	40.8	40.8	40.8	89.3	89.3	89.3	166	166	166
Torch Script	7.6	7.6	7.6	28.1	28.1	28.1	81.4	81.4	81.4	-	178.2	178.2	-	331.6	331.6
Onnx	7.6	-	7.6	28	-	28	81.2	-	81.2	-	-	178	-	-	331.2
OpenVINO	7.7	-	7.7	28.2	-	28.2	81.4	-	81.4	-	-	178.2	-	-	331.5
TensorRT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CoreML	-	-	7.6	-	-	28.1	-	-	81.3	-	-	178.1	-	-	331.3
TensorFlow Saved Model	7.5	7.4	7.4	27.9	27.8	27.8	81.2	81.1	81.1	-	177.9	177.9	-	331.2	331.2
TensorFlow GraphDef	7.4	7.4	7.4	27.8	27.8	27.8	81.1	81.1	81.1	-	177.9	177.9	-	331.2	331.2
TensorFlow Lite	3.7	3.7	3.7	13.9	13.9	13.9	40.5	40.5	40.5	-	88.9	88.9	-	165.6	165.6
TensorFlow Edge TPU	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TensorFlow.js	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PaddlePaddle	15.2	15.2	15.2	56.1	56.1	56.1	162.5	162.5	162.5	-	356	356	-	662.5	662.5
Mean	7.575	7.533	7.567	28.013	27.967	28.011	81.263	81.233	81.256	89.300	178.033	178.056	166.000	331.350	331.344
Median	7.55	7.4	7.6	27.95	27.8	28	81.2	81.1	81.2	89.3	177.9	178	166	331.2	331.2
Max	15.2	15.2	15.2	56.1	56.1	56.1	162.5	162.5	162.5	89.3	356	356	166	662.5	662.5
Min	3.7	3.7	3.7	13.9	13.9	13.9	40.5	40.5	40.5	89.3	88.9	88.9	166	165.6	165.6

YOLOv5 Average Precision Across Different Architectures

mAP50-95															
	Nano Model			Small Model			Medium Model			Large Model			Extra Large Model		
Benchmark Format	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini
PyTorch	0.3491	0.3491	0.3491	0.4715	0.4715	0.4715	0.5537	0.5537	0.5537	0.6013	0.6013	0.6013	0.6286	0.6286	0.6286
Torch Script	0.3491	0.3491	0.3491	0.4715	0.4715	0.4715	0.5537	0.5537	0.5537	-	0.6013	0.6013	-	0.6286	0.6286
Onnx	0.3491	-	0.3491	0.4715	-	0.4715	0.5537	-	0.5537	-	-	0.6013	-	-	0.6286
OpenVINO	0.351	-	0.3512	0.468	-	0.4675	0.5522	-	0.553	-	-	0.5994	-	-	0.6274
TensorRT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CoreML	-	-	0.3491	-	-	0.4715	-	-	0.5537	-	-	0.6013	-	-	0.6286
TensorFlow Saved Model	0.3491	0.3491	0.3491	0.4715	0.4715	0.4715	0.5537	0.5537	0.5537	-	0.6013	0.6013	-	0.6286	0.6286
TensorFlow GraphDef	0.3491	0.3491	0.3491	0.4715	0.4715	0.4715	0.5537	0.5537	0.5537	-	0.6013	0.6013	-	0.6286	0.6286
TensorFlow Lite	0.3486	0.3486	0.3486	0.4716	0.4716	0.4716	0.5518	0.5518	0.5518	-	0.6001	0.6001	-	0.6278	0.6278
TensorFlow Edge TPU	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TensorFlow.js	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PaddlePaddle	0.3491	0.3491	0.3491	0.4715	0.4715	0.4715	0.5537	0.5537	0.5537	-	0.6013	0.6013	-	0.6286	0.6286
Mean	0.3493	0.3490	0.3493	0.4711	0.4715	0.4711	0.5533	0.5534	0.5534	0.6013	0.6011	0.6010	0.6286	0.6285	0.6284
Median	0.3491	0.3491	0.3491	0.4715	0.4715	0.4715	0.5537	0.5537	0.5537	0.6013	0.6013	0.6013	0.6286	0.6286	0.6286
Max	0.351	0.3491	0.3512	0.4716	0.4716	0.4716	0.5537	0.5537	0.5537	0.6013	0.6013	0.6013	0.6286	0.6286	0.6286
Min	0.3486	0.3486	0.3486	0.468	0.4715	0.4675	0.5518	0.5518	0.5518	0.6013	0.6001	0.5994	0.6286	0.6278	0.6274

YOLOv5 Inference Times Across Different Architectures

Inference Time (ms)															
	Nano Model			Small Model			Medium Model			Large Model			Extra Large Model		
Benchmark Format	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini	Pi5	Surface	Mac Mini
PyTorch	321.5	79.85	39.02	651.1	162.32	64.17	1364.54	363.67	114.05	2464.34	721.77	189.9	3760.62	1312.16	294.75
Torch Script	297.09	73.26	40.08	626.87	162.6	65.54	1311.47	356.83	110.43	-	817.96	184.73	-	1570.92	319.65
Onnx	148.17	-	22.24	380.81	-	61.18	788.35	-	151.52	-	-	300.47	-	-	534.73
OpenVINO	84.46	-	41.46	153.44	-	66.42	343.55	-	101.66	-	-	136.04	-	-	194.86
TensorRT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CoreML	-	-	27.4	-	-	19.75	-	-	26.5	-	-	38.3	-	-	62.48
TensorFlow Saved Model	237.06	69.21	20.41	488.36	143.63	27.87	1093.4	343.79	38.98	-	742.25	62.07	-	1116.52	116.09
TensorFlow GraphDef	236.87	71.35	21.07	490.17	144.76	27.4	1078.91	335.83	40.31	-	810.51	62.78	-	1598.57	103.46
TensorFlow Lite	253.34	68.99	75.25	700.45	197.65	243.24	1791.34	504.51	679.02	-	1113.99	1467.22	-	2205.46	2730.12
TensorFlow Edge TPU	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TensorFlow.js	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PaddlePaddle	510.7	101.59	106.02	1064.55	21	267.4	2496.33	413.78	662.26	-	753.4	1357.55	-	1283.38	2422.52
Mean	261.15	77.38	43.66	569.47	138.66	93.66	1283.49	386.40	213.86	2464.34	826.65	422.12	3760.62	1514.50	753.18
Median	245.2	72.305	39.02	558.52	153.54	64.17	1202.435	360.25	110.43	2464.34	781.955	184.73	3760.62	1441.54	294.75
Max	510.7	101.59	106.02	1064.55	197.65	267.4	2496.33	504.51	679.02	2464.34	1113.99	1467.22	3760.62	2205.46	2730.12
Min	84.46	68.99	20.41	153.44	21	19.75	343.55	335.83	26.5	2464.34	721.77	38.3	3760.62	1116.52	62.48
Pytorch Performance vs. Pi5	1.00	4.03	8.24	1.00	4.01	10.15	1.00	3.75	11.96	1.00	3.41	12.98	1.00	2.87	12.76
Pytorch Performance vs. Surface Pro	0.25	1.00	2.05	0.25	1.00	2.53	0.27	1.00	3.19	0.29	1.00	3.80	0.35	1.00	4.45
Pytorch Performance vs. Mac Mini	0.12	0.49	1.00	0.10	0.40	1.00	0.08	0.31	1.00	0.08	0.26	1.00	0.08	0.22	1.00

TABLE III. PERFORMANCE COMPARISONS

Model	**Performance**		
YOLOv8	Cycles	490,196,308,242	1.729 GHz
	Instructions	413,813,398,747	0.84 insn per cycle
	Cache-Misses	4,802,178,969	5.21% of all cache refs
	Cache-References	92,254,164,579	325.436 M/sec
	Branch-Misses	318,593,074	
	Task-Clock	283,478.49 msec	1.813 CPUs utilized
	Context-Switches	0	0.000 /sec
	Page-Faults	709,206	2.502 K/sec

	Time Elapsed	156.386003770 sec	
	User	275.720335000 sec	
	System	8.089143000 sec	
	FPS = 1/.9571s = 1.045 FPS		
YOLOv11	Cycles	539,804,227,494	1.702 GHz
	Instructions	448,732,835,022	0.83 insn per cycle
	Cache-Misses	4,984,938,173	5.01% of all cache refs
	Cache-References	99,595,990,618	314.072 M/sec
	Branch-Misses	345,711,518	
	Task-Clock	317,112.05 msec	1.881 CPUs utilized
	Context-Switches	0	0.000 /sec
	Page-Faults	1,427,568	4.502 K/sec
	Time Elapsed	168.586001219 sec	
	User	304.288101000 sec	
	System	13.142862000 sec	
	FPS = 1/1.0594s = 0.94429 FPS		