Benchmarking YOLOv8–YOLOv12 for Real-Time Object Detection on Single-Board Computers

Omar Shalash *, Esraa Khatab, Ahmed El-Agamy, Loay Elmokadem, Yasmin Abouelsaad, Jasser Zaki, Mohamed El-Sayed, Hany Said

Posted Date: 14 May 2026

doi: 10.20944/preprints202605.0936.v1

Keywords: YOLO object detection; single-board computers; real-time systems; computer vision

 doi:10.20944/preprints202605.0936.v1

Article

Benchmarking YOLOv8–YOLOv12 for Real-Time Object Detection on Single-Board Computers

Omar Shalash [1,*], Esraa Khatab [2], Ahmed El-Agamy [3], Loay Elmokadem [3], Yasmin Abouelsaad [3], Jasser zaki [3], Mohamed El-Sayed [3] and Hany Said [3]

[1] Artificial Intelligence Research Center (AIRC), College of Engineering and Information Technology, Ajman University, Ajman P.O. Box 346, United Arab Emirates
[2] School of Mathematics and Computer Science Herriot Watt University
[3] Arab Academy for Science, Technology & Maritime Transport
* Correspondence: o.shalash@ajman.ac.ae; Tel.: 971-555288505

Abstract

The YOLO (You Only Look Once) object detection models have undergone rapid evolution, with each version introducing architectural enhancements aiming to improve speed, accuracy, and deployment. Simultaneously, Single-Board Computers (SBCs) have advanced to support increasingly complex AI models in edge environments. This study presents a comprehensive benchmarking of YOLO versions 8 through 12 across a range of SBCs, including Raspberry Pi4/5, NVIDIA Jetson Nano, Jetson Orin, and LattePanda, under different power modes. Key performance metrics, including inference speed (FPS), detection accuracy (mAP), RAM usage, and computational complexity (FLOPs), are evaluated. These findings offer practical insights for developers and researchers to select optimal YOLO variants and SBC configurations for real-time edge deployment.

Keywords: YOLO object detection; single-board computers; real-time systems; computer vision

1. Introduction

Object detection has become a fundamental task in computer vision, enabling machines to classify and localize multiple objects in different scenes within images or video streams. Its importance spans a wide range of real-world applications, including autonomous driving, surveillance, robotics, smart cities, healthcare diagnostics, and retail analysis. As the demand for deploying intelligent systems on resource-constrained edge devices grows, the need for efficient and accurate object detection models becomes increasingly important.

Object detection techniques are categorized into two groups: conventional techniques and detection based on Deep Learning (DL). Classical object detection methods rely on manually crafted features and classical machine learning techniques, typically using approaches like Viola-Jones (VJ), Histogram of Oriented Gradients (HoG), or Deformable Part Models (DPM) [1]. The VJ detector, introduced by Viola and Jones in 2001 [2], detects faces using sliding windows and Haar-like features. In 2005, Dalal and Triggs developed HoG [3], which identifies gradient orientations within image blocks and applies image scaling with fixed detection windows, feeding the data into classifiers like Support Vector Machines. DPM, introduced by Felzenszwalb in 2008 as an extension of HoG [4], combines a coarse detection window with part-based models to improve object localization.

On the other hand, DL-based object detection utilizes neural networks, particularly Convolutional Neural Networks (CNNs), to automatically extract complex features from raw image data [5], thereby significantly improving detection performance across various real-world applications, such as autonomous driving, surveillance, and healthcare. These models fall into two main categories [6]: one-stage detectors, such as You Only Look Once (YOLO) [7] and Single-Shot Detector (SSD) [8], which perform detection in a single step for fast, real-time processing; and two-stage detectors, like Faster

 doi:10.20944/preprints202605.0936.v1

R-CNN and R-CNN variant models [9], which first generate region proposals and then refine them for higher accuracy, though at the cost of increased computational complexity [10–12].

The YOLO family of models has emerged as a leading solution due to its real-time performance and evolving architectural optimizations. Different YOLO versions have been widely used in various real-world applications due to their remarkable balance between speed and accuracy, including autonomous driving, the healthcare sector, and surveillance systems. The model's ability to run efficiently on edge devices also makes it highly suitable for robotics, where drones and mobile robots use YOLO for navigation and object tracking in dynamic environments. However, with the growing number of YOLO variants, it was challenging to select the most suitable version for a specific need. In this study, we focus on benchmarking the latest YOLO versions: YOLOv8 through YOLOv12, across different Single Board Computers (SBCs) and power modes, evaluating their suitability for edge deployment through key performance metrics such as inference speed (FPS), accuracy (mAP), memory usage, computational complexity (FLOPs), and model load time.

2. Literature Review

This section reviews the evolution of YOLO-based object detection models, highlighting key architectural advancements from early versions to the most recent releases. It also examines prior benchmarking and review studies, identifying limitations and gaps in evaluating YOLO performance on resource-constrained single-board computing platforms.

2.1. YOLO Series Timeline and Evolution

As shown in Figure 1, YOLO's developing pathway is shaping a unique object detection framework that is evolving continuously [13]. YOLOv1-v2 revolutionized object detection by introducing a unified CNN-based architecture that processes the entire image in one pass, dividing it into a grid to predict bounding boxes and class probabilities. YOLOv2 builds on this by adopting Darknet-19, adding anchor boxes for accurate multi-object detection, and supporting multi-resolution inputs; boosting accuracy on benchmarks like COCO and VOC [14,15]. YOLOv3 switched to Darknet-53 with residual connections and multi-scale detection heads, making it more effective at detecting objects of various sizes. It also improved anchor box generation and classification [16]. YOLOv4 formalized the modular design of backbone, neck, and head by using CSPDarknet-53 and PAN+SPP for feature fusion, achieving a strong balance between speed and accuracy [17]. YOLOv5 retains the modular structure but adds deployment-friendly features and components such as SPP and FPN+PAN [18]. YOLOv6 introduced RepVGG-inspired blocks and multi-task loss functions for better precision [19]. YOLOv7 focused on optimizing training and inference with E-ELAN and Trainable Bag-of-Freebies, refining the speed-accuracy trade-off for real-time tasks.

On the other hand, YOLOv8 shifted to an anchor-free paradigm, simplifying the detection head and enhancing multi-scale feature aggregation through Cross Stage Partial Network (CSPNet), Feature Pyramid Network (FPN), and Path Aggregation Network (PAN). It improved the detection of small and large objects using more efficient CNN operations [20]. YOLOv9 introduced Programmable Gradient Information (PGI) and a refined version of Gradient Enhanced Lightweight Aggregation Network (GELAN) as the backbone to reduce information loss during gradient backpropagation. It remained anchor-free, emphasized stable training and feature reuse [21].

YOLOv10 transitioned to one-to-one object prediction, eliminating the need for Non-Maximum Suppression (NMS). Compact Inverted Blocks (CIB) and Range-Guided Block designs are deployed for YOLOv10, to improve localization while reducing redundancy [22]. YOLOv11 further enhanced small object detection via modules like C3k2, SPPF, and C2PSA, which improved feature extraction, spatial pooling, and attention mechanisms, particularly in complex scenes [23].

Finally, YOLOv12 focused on latency reduction by simplifying the attention mechanism, using a positional perceiver, and reducing MLP ratios, while maintaining detection accuracy. It is the most efficient version for deployment on latency-critical edge systems [24].

A summarized comparison between YOLO architectures is presented in Table 1.

 doi:10.20944/preprints202605.0936.v1

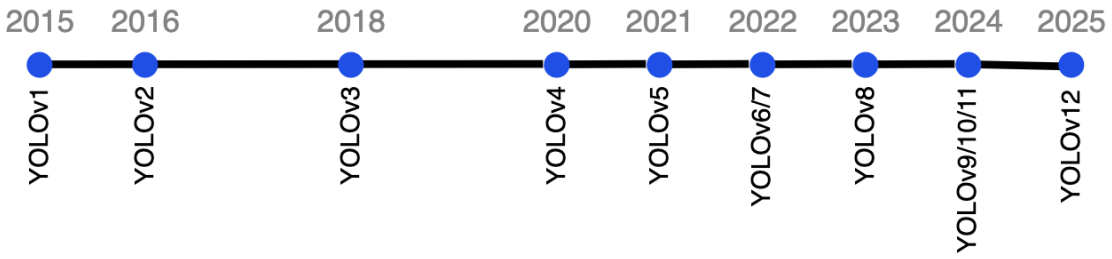

Figure 1. YOLO Series Timeline.

Table 1. Comparison of YOLO Architectures from YOLOv1 to YOLOv12.

Version	Backbone	Neck	Head	Detection Strategy
YOLOv1	Custom CNN	N/A	2 FC layers	Grid cells (7x7)
YOLOv2	Darknet-19	N/A	Anchor boxes, Class probs	Anchor boxes
YOLOv3	Darknet-53	Feature Pyramid	3 Heads (Multi-scale)	Anchor boxes + K-means
YOLOv4	CSPDarknet-53	PAN + SPP	Same as YOLOv3	Improved neck
YOLOv5	CSPDarknet + SPP	FPN + PAN	3 Heads	PyTorch based
YOLOv6	Rep Backbone	FPN + PAN + RepBlock	IoU-based NMS	Multi-task loss
YOLOv7	CSPDarknet + E-ELAN	PAN	Trainable BoF	Optimized for speed
YOLOv8	CSPNet	FPN + PAN	Anchor-free	Improved flow
YOLOv9	GELAN	PGI	Anchor-free	Gradient optimization
YOLOv10	Compact Inv. Block	Range-guided Block	One-to-One Prediction	No NMS
YOLOv11	C3k2 + SPFF	C2PSA	Lightweight head	Enhanced attention
YOLOv12	Simplified C2PSA	Positional Perceiver	Reduced MLP ratio	Latency optimized

2.2. Prior Review Articles on YOLO Models

There are many comprehensive reviews on YOLO models:

- In [25], authors reviewed YOLOv1 through YOLOv5, analyzing their architectures, detection speed, accuracy, and dataset compatibility.
- In [26], authors compared YOLOv7, SSD, and Faster R-CNN on small SBC platforms such as Raspberry Pi and Jetson Nano, evaluating metrics like inference speed (FPS), accuracy (mAP), power usage, and resource efficiency.
- In [27], authors systematically explored the internal architectural design of each YOLO model 1 through model 8, along with their benchmark performance across standard datasets.
- In [28], authors reviewed YOLOv1 through YOLOv8 with detailed performance metrics (e.g., inference time, detection accuracy), architectural designs, datasets, and diverse applications.
- In [29], authors examined the architectural advancements of YOLOv1 through YOLOv11, along with benchmark performance, and applications.

2.3. Research Gap and Contribution

Despite the widespread use of YOLO models, there is still limited research that thoroughly tracks their evolution and compares performance across different versions. This gap becomes critical when deploying YOLO on resource-constrained devices such as Single-Board Computers (SBCs). This includes both Raspberry Pi and NVIDIA Jetson, which are commonly used in edge computing. These devices offer affordable and portable AI solutions but have limited processing power. Understanding how various YOLO versions perform in terms of speed, latency, power consumption, and detection accuracy is crucial for real-time applications. This study addresses that need by systematically evaluating YOLO models from a deployment perspective, helping guide practical and efficient model selection for embedded systems and edge AI use cases. The main contributions of this study are:

- Benchmark multiple YOLO models with their variants
- Compare them across PC vs. SBC under the same dataset
- Evaluate accuracy and system metrics (FPS, load time, latency, FLOPs).

 doi:10.20944/preprints202605.0936.v1

2.4. SBCs

In education and research, SBCs serve as accessible platforms for embedded systems , learning programming, robotics , autonomous vehicls (AV) , electric vehicles (EV) and artificial intelligence (AI). The Raspberry Pi boards, for example, have been extensively used in STEM education, allowing students to experiment with coding, sensor integration, and networking [30]. Meanwhile, Jetson Nano and Jetson Orin are optimized for AI and deep learning applications, enabling object detection, autonomous robotics, and edge computing in a real-time [31]. Their GPU acceleration capabilities make them particularly valuable for machine learning inference and computer vision tasks.

Beyond academia, SBCs have found applications in the Internet of Things (IoT), industrial automation, and smart systems. The LattePanda 3 Delta, with its compatibility for both Windows and Linux, bridges the gap between embedded systems and full-scale computing, making it suitable for tasks that require both microcontroller-like control and desktop-level processing power [32]. Industries utilize SBCs for real-time data collection, automation, and remote monitoring, reducing costs while enhancing task efficiency.

As technology advances, SBCs continue evolve to have more powerful processors, with increased RAM capacities and improved AI capabilities. Their affordability and adaptability make them indispensable tools for innovators, researchers, and developers across various fields [31,33,34]. With ongoing improvements, SBCs are expected to drive advancements in smart cities, healthcare, edge computing, and autonomous systems, shaping the future of embedded computing.

Recent advancements in object detection have led to significant improvements across diverse application domains. Wang et al. proposed PG-YOLO, a lightweight YOLOv8-based model optimized for pomegranate fruitlet detection before thinning, incorporating ShuffleNetv2, depthwise separable convolutions, and multi-head self-attention to achieve high accuracy with minimal computational cost [35]. Similarly, Wu et al. introduced YOLO-Claw, an enhanced YOLOv5 variant for chicken claw detection in precision livestock farming, integrating Tanh-aided channel attention (TCA), C3TCA modules, and AFPN-Neck for robust feature extraction under occlusion and lighting variations [36]. In the maritime domain, Chen et al. developed a poly-YOLO-based framework for ship trajectory extraction, combining multi-scale feature aggregation with Enhanced Deep Sort tracking to improve detection and tracking accuracy in complex sea conditions [37]. For ecological monitoring, Wang et al. proposed WB-YOLO, an improved YOLOv7 model for wild bat detection, incorporating Vision Transformer encoders, deformable convolutions, and hybrid attention mechanisms to handle occlusion and cluttered backgrounds effectively [38]. Finally, Yang et al. introduced Tunnel-YOLO, an advanced YOLOv8-based algorithm for shield tunnel leakage detection, featuring RFCAConv for adaptive receptive fields, C2f_SGE for noise suppression, EFE-Head for fine-grained edge enhancement, and LeShape-IoU for shape-sensitive bounding box regression [39]. Collectively, these studies highlight the trend of tailoring YOLO architectures with attention mechanisms, lightweight modules, and multi-scale fusion strategies to balance accuracy, robustness, and real-time performance across agriculture, livestock, maritime, ecological, and infrastructure domains.

3. Methodology

This section describes the experimental methodology adopted to benchmark YOLOv8 through YOLOv12 on diverse single-board computing platforms. It details the datasets, model configurations, hardware platforms, operating conditions, and evaluation metrics used to ensure a fair and reproducible comparative analysis.

3.1. Dataset

In this study, the COCO (Common Objects in Context) dataset has been used. It is a widely used benchmark in object detection research due to its rich and diverse collection of annotated images [40]. It contains over 330,000 images, with more than 80 object categories, and includes detailed annotations such as object segmentation, keypoints, and contextual information. COCO is particularly valuable for

 doi:10.20944/preprints202605.0936.v1

training and evaluating deep learning models because it presents objects in complex, real-world scenes with varying scales, occlusions, and lighting conditions. In this research, COCO was chosen to assess the performance of different YOLO versions, providing a standardized and challenging environment to measure detection accuracy, speed, and robustness across models.

3.2. YOLO Models Evaluated

This study evaluates the performance of the latest YOLO object detection models, specifically YOLOv8 through YOLOv12. Each YOLO version includes multiple scaled variants to accommodate different hardware and performance needs. A summary of their sizes is shown in Table 2. All models support transfer learning and can be fine-tuned on custom datasets. YOLO models benefit from strong community support, extensive documentation, and integration with platforms such as Ultralytics [41], ONNX [42], and TensorRT [43], facilitating rapid prototyping and deployment.

Table 2. YOLO Model Variants and Their Parameter Sizes (in Millions).

Model	n	s	m	l	x
YOLOv8	3.2	11.2	25.9	43.7	68.2
YOLOv9	2.0	7.2	20.1	25.5	58.1
YOLOv10	2.3	7.2	5.4	4.4	29.5
YOLOv11	2.6	9.4	20.1	25.3	56.9
YOLOv12	2.6	9.3	20.2	26.4	59.1

3.3. Single-Board Computer Systems

Single-Board Computers (SBCs) have revolutionized the field of embedded systems. They provide compact, cost-effective, and versatile computing solutions for a wide range of applications. These devices integrate essential computing components such as processors, memory, storage, and connectivity onto a single circuit board, making them ideal for both educational and industrial purposes. SBCs like the Raspberry Pi, Jetson Nano, Jetson Orin, and LattePanda have become widely adopted due to their low power consumption, small form factor, and broad operating system support [31–34].

In this research, YOLO models were evaluated on different SBCs, in addition to a PC. The main differences are summarized in Table 3.

Table 3. Comparison of Featured Single-Board Computers.

Feature	Raspberry Pi 4B [30]	Raspberry Pi 5 [33]	Jetson AGX Orin [31]	Jetson Nano [34]	LattePanda 3 Delta [32]
CPU	BCM2711, Quad-Core Cortex-A72, 1.5 GHz	BCM2712, Quad-Core Cortex-A76, 2.4 GHz	Custom SoC, 2 GPCs, 16 SMs, 1.5 GHz	Quad-Core Cortex-A57, 1.42 GHz	Intel Celeron N5105, Quad-Core, 2.0 GHz (Boost 2.9 GHz)
GPU	VideoCore VI, OpenGL ES 3.1, H.265 Decoding	VideoCore VII	Ampere-based, 2048 CUDA Cores	Maxwell, 128 CUDA Cores	Intel UHD, 24 Execution Units (EUs)
RAM	2 GB / 4 GB / 8 GB LPDDR4	4 GB / 8 GB LPDDR4X	32 GB / 64 GB LPDDR5	4 GB LPDDR4, 1600 MHz	8 GB LPDDR4, 2933 MHz
Storage	microSD Slot	microSD Slot	eMMC with NVMe Support	16 GB eMMC	64 GB eMMC with NVMe Support
Power	5V / 3A USB-C, 3W (Overclocked 7W)	5V / 5A USB-C, 5W-12W	Variable Power Mode, up to 170W	5V, 5W-10W	12V / 3A, 6W-15W
GPIO	40-pin header, 28 GPIOs	40-pin header, 28 GPIOs	40-pin expansion header, Configurable GPIOs	40-pin header, Configurable GPIOs	ATmega32u4-based GPIO, 20 digital I/O pins
OS Support	Raspberry Pi OS, Linux, NOOBS	Raspberry Pi OS, Linux	Linux, NOOBS	Linux, NOOBS	Windows 10/11, Linux

3.3.1. PC

A desktop PC was used as the reference system for the experimental setup, equipped with an NVIDIA GeForce RTX 3060 Ti (8 GB), 48 GB of RAM, and an AMD Ryzen 5 5600X 6-core processor running at 3.70 GHz.

 doi:10.20944/preprints202605.0936.v1

3.3.2. Raspberry Pi 4 - Model B

The Raspberry Pi 4 Model B is a compact yet capable single-board computer powered by a Broadcom BCM2711 quad-core Cortex-A72 CPU running at 1.5 GHz. It features a VideoCore VI GPU with support for OpenGL ES 3.1 and H.265 video decoding, making it suitable for lightweight graphics and media tasks. Available in 2 GB, 4 GB, and 8 GB LPDDR4 RAM variants, this study utilizes the 8 GB model for enhanced performance. Storage is handled via a microSD card, and the board operates on a 5V/3A USB-C power supply, typically consuming around 3W, with the ability to scale up to 7W depending on the power mode. It includes a 40-pin header with 28 GPIO pins for hardware interfacing and supports multiple operating systems, including Raspberry Pi OS, NOOBS, and other Linux distributions. [30].

3.3.3. Raspberry Pi 5 - Model B

The Raspberry Pi 5 Model B brings a significant upgrade over its predecessor, featuring a Broadcom BCM2712 quad-core Cortex-A76 CPU running at 2.4 GHz and a VideoCore VII GPU with enhanced support for OpenGL ES 3.1 and hardware-accelerated H.265 decoding. They are available in 4 GB and 8 GB LPDDR4X RAM variants. This board features a PCIe 2.0 interface, enabling faster storage options beyond the traditional microSD card. It operates on a 5V/5A USB-C power supply, with power consumption ranging from 5W to 12W depending on the selected mode—low, standard, or high power. Like previous models, it is equipped with a 40-pin header, in addition to 28 GPIO pins. This board supports Raspberry Pi OS along with other Linux distributions. [33].

3.3.4. NVIDIA Jetson Orin

According to the NVIDIA Jetson AGX Orin datasheet [31], the Jetson AGX Orin 32 GB edition features a System-on-Chip (SoC) with two Graphic Processing Clusters (GPCs), each containing 8 Texture Processing Clusters (TPCs) and 16 Streaming Multiprocessors (SMs). Each SM includes 192 KB of L1 cache and 4 MB of L2 cache, with 128 CUDA cores and four 3rd-generation Tensor cores. It doubles the CUDA cores compared to the Volta architecture, where the 64 GB variant offers 2048 CUDA cores and 64 Tensor cores, delivering up to 170 Sparse TOPs of INT8 Tensor compute and 5.3 FP32 TFLOPs of CUDA compute. In comparison, the 32 GB model includes 1792 CUDA cores and 56 Tensor cores, achieving up to 108 Sparse TOPs of INT8 and 3.37 FP32 TFLOPs. The board operates at a frequency of 1.5 GHz and supports RAM configurations of 2 GB or 8 GB. It also features a 40-pin GPIO header and runs on Linux-based operating systems. The one used in this study is NVIDIA Jetson Orin Nano with 8 GB.

3.3.5. NVIDIA Jetson Nano

The Jetson Nano board has a Quad-Core ARM Cortex-A57 64-bit with a 1.42 GHz CPU. The board is equipped with NVIDIA Maxwell 128 CUDA cores and a GPU (921 MHz). The board contains a 4 GB LPDDR4 RAM with a frequency of 1600 MHz. It supports upto 25.6 GB/s of data transfer. The Jetson Nano has 16 GB of eMMC storage. The board consumes power ranging from 5W to 10W based on the power mode. The board supports diverse peripherals, including a 40-pin GPIO and a MIPI port for camera [34]. The NVIDIA Jetson Nano with 4 GB has been evaluated in this study.

3.3.6. LattePanda

The LattePanda 3 Delta has an Intel Celeron N5105 quad-core 2.0 GHz CPU (can be overclocked to 2.9 GHz and a built-in ATmega 32U4 co-processor. The board is equipped with Intel UHD Graphics and 24 Execution Units (EUs) GPU, supporting DirectX 12, OpenGL 4.6, and OpenCL 3.0. The board has 8 GB of LPDDR4 RAM clocked at 2933 MHz. Also, a 64 GB eMMC storage. LattePanda 3 Delta requires a 12V/3A input, with power consumption ranging from 6W to 15W, based on the operating power mode. The board has 20-pin GPIO. The board is compatible with Windows 10, Windows 11, and various Linux distributions [32].

 doi:10.20944/preprints202605.0936.v1

3.4. Evaluation Metrics

To comprehensively assess the performance and deployment suitability of YOLOv8 through YOLOv12 models on edge devices, we employ a diverse set of evaluation metrics that capture both computational efficiency and detection quality. These metrics include Frames Per Second (FPS) for inference speed, Mean Average Precision (mAP) for detection accuracy, RAM utilization for memory efficiency, Floating Point Operations (FLOPs) for computational complexity, and model load time for initialization latency.

Each metric provides a unique perspective to evaluate the trade-offs between critical factors when deploying models on SBCs: speed, accuracy, and resource consumption. By analyzing these metrics across different YOLO variants and power modes, we aim to identify optimal configurations for real-time object detection in edge computing scenarios.

3.4.1. Precision

It measures the proportion of correctly predicted bounding boxes (true positives) out of the total predicted boxes, including false positives, as shown in equation (1). It quantifies the model accuracy toward making correct predictions.

$$Precision = \frac{TruePositives(TP)}{TruePositives(TP) + FalsePositives(FP)} \tag{1}$$

3.4.2. mAP@50-90

Mean Average Precision (mAP@50-90) is a comprehensive metric used to evaluate object detection performance across a range of Intersection over Union (IoU) thresholds. In contrast to traditional mAP@50, which considers a single IoU threshold, mAP@50–90 averages the precision over ten thresholds from 0.50 to 0.95 in steps of 0.05. This provides a more robust assessment of a model's localization accuracy and generalization capability. It is especially beneficial for benchmarking models on challenging datasets and is the standard metric used in the COCO evaluation protocol.

$$mAP_{50:90} = \frac{1}{10} \sum_{t=0.50}^{0.95} AP_t \tag{2}$$

Where AP_t is the Average Precision at IoU threshold t, and the summation is taken over thresholds $t = 0.50, 0.55, \ldots, 0.95$.

3.4.3. Latency

Latency measures the time consumed during model processing of the image and performing the predictions, including all the steps required for the detection process, such as image processing, model inference, and post-processing. It is typically measured in milliseconds (*ms*). Lower latency is crucial for real-time applications, such as autonomous navigation, where fast and accurate detection is mandatory. Moreover, Frames per Second (FPS) can also be used to complement latency, which indicates how many frames can be processed by the model in one second. Together, latency and FPS provide a comprehensive understanding of the model's speed and performance in real-world scenarios.

3.4.4. Load Time

Load time (in milliseconds): It is the time taken to load a model or data into memory from disk. It includes reading the model file from disk, deserializing it, and initializing weights and layers in RAM or GPU memory. It is important in real-time applications. Load time is influenced by disk I/O speed, model size, system memory bandwidth, and CPU/GPU initialization overhead.

3.5. Operating Systems

Table 4 displays all the operating systems used across all devices in this study.

 doi:10.20944/preprints202605.0936.v1

Table 4. Operating Systems for Different Devices.

Device	Operating System
PC	Windows 11 Version 23H2
Raspberry Pi 4 and 5	Debian GNU/Linux 12 (bookworm)
LattePanda	Windows 10 Pro (22H2)
NVIDIA Jetson Nano	Nvidia Jetpack 4.6.6
NVIDIA Orin	Ubuntu 20.04.6 LTS (Focal Fossa)

3.6. Power Modes

Most of the SBCs today have three different power modes. SBCs incorporate configurable power management modes to optimize energy consumption, thermal performance, and computational efficiency based on workload demands. These modes are typically: performance, balanced, and power-saving modes, each governed by dynamic voltage and frequency scaling algorithms. In performance mode, the processor operates at its maximum clock speed, prioritizing computational throughput at a higher power dissipation. Conversely, power-saving mode restricts clock frequencies to minimize energy usage, making it suitable for battery-operated or thermally constrained environments. A balanced mode dynamically adjusts CPU performance in response to real-time workload requirements, offering a compromise between efficiency and speed. Additionally, thermal throttling mechanisms may forcibly reduce performance to prevent overheating, ensuring hardware longevity.

Testing different YOLO versions and models on different power modes is a game-changer for researchers. For example, two of the main goals for drones, rovers, or any battery-powered robotic vehicle are battery lifetime on the robot and the performance of the selected SBC, given the running tasks. Developers tend to choose a high-performance SBC (such as the Jetson Nano and Orin) to ensure real-time performance, along with one of the YOLO nano models to ensure a minimum computational power. Including such power modes in the study enables the researchers to make better choices for both the SBC and YOLO model, with its variants.

3.6.1. Raspberry Pi 4 Power Modes

The Raspberry Pi 4 configuration setting has three configurable power modes. The Low Power Mode (600MHz CPU/200MHz GPU) undervolts the SoC (-8 offset) with a 70°C thermal limit, ideal for always-on applications. The revised Normal Mode (1500MHz CPU/500MHz GPU) enforces stock voltages (0 offset), a higher 85°C thermal limit, and locked GPU clocks (500MHz min/max), offering stable performance for general computing tasks without active cooling. For demanding workloads, High Power Mode (1800MHz CPU/600MHz GPU) employs +2 overvolting and an 80°C limit, requiring active cooling. Raspberry Pi Dual Fans 5V DC with Heatsink Cooling System was installed to boost the board's performance.

3.6.2. Raspberry Pi 5 Power Modes

Similar to Raspberry Pi 4, Raspberry Pi 5 settings can be configured to create three power modes, each tailored for specific performance and efficiency trade-offs. The Low Power Configuration (600MHz CPU/200MHz GPU) minimizes energy consumption by underclocking the CPU and GPU while reducing voltage (-8 offset), making it ideal for lightweight, always-on tasks where thermal management is critical (temp_limit=70°C). The Normal Power Configuration (2400MHz CPU/500MHz GPU) operates at default voltages and clocks, balancing performance and power efficiency for general-purpose workloads (temp_limit=85°C). Finally, the High Performance Configuration (2900MHz CPU, 1100MHz GPU, force_turbo=1) pushes the hardware to its limits with overvolting (+6 CPU, +4 SDRAM) and elevated thermal thresholds (temp_limit=90°C), unlocking maximum throughput for compute-intensive applications—though active cooling is mandatory to sustain stability.

 doi:10.20944/preprints202605.0936.v1

3.6.3. LattePanda Power Modes

For LattePanda, the power modes tested were Power Saver, Balanced, and High Performance, which were set through Windows settings.

3.6.4. Jetson Nano Power Modes

The Jetson Nano can operate in two primary power modes. The default mode is the 5W configuration, which prioritizes energy efficiency by reducing CPU/GPU clock speeds, making it suitable for lightweight or thermally constrained applications. The alternative 10W mode unlocks higher performance by increasing power limits. This requires active cooling to prevent thermal throttling; this is one of the main reasons fans were installed on experimental SBCs. The Jetson Nano's hardware enforces a strict 10W ceiling, limiting sustained performance. This simpler power management scheme reflects the Nano's entry-level positioning, where thermal design and power delivery are optimized for cost and simplicity rather than scalability. An attachment to the board was fixed. Raspberry Pi Active Cooler, is an official, fan-based cooling solution designed specifically for the RaspberryPi 5, that addresses its higher thermal demands compared to earlier models.

3.6.5. Jetson Orin Nano Power Modes

For Jetson Orin Nano, there are three available power modes. The first power mode is MODE_10W, which provides the lowest power with reduced performance. Followed by MODE_15W, which is the default mode (balanced performance over power). Lastly, MAXN power mode, which is the max performance, draws power up to 20W (no strict Thermal Design Power (TDP) cap, but thermals limit sustained power).

In conclusion, the Raspberry Pi 4 and 5 are the only two boards equipped with coolers, and their performance mode has to be manually set. The three other boards had built-in configurations for optimizing power modes; these configurations are summarized in Table 5.

Table 5. Power Mode Configurations.

Board and Mode	Settings
RPi 4	
- Low Power	600MHz CPU, 200MHz GPU
- Normal	1500MHz CPU, 500MHz GPU
- High Power	1800MHz CPU, 600MHz GPU
RPi 5	
- Low Power	600MHz CPU, 200MHz GPU
- Normal	2400MHz CPU, 500MHz GPU
- High Perf.	2900MHz CPU, 1100MHz GPU
LattePanda	
- Power Saver	Windows efficiency mode
- Balanced	Windows default
- High Perf.	Windows max performance
Jetson Nano	
- 5W Mode	Reduced clocks
- 10W Mode	High performance
Orin Nano	
- 10W Mode	Low power
- 15W Mode	Default
- MAXN	Up to 20W

4. Results and Discussions

In this section a detailed analysis of the experimental results obtained from benchmarking YOLOv8 through YOLOv12 across multiple single-board computing platforms. The discussion focuses

 doi:10.20944/preprints202605.0936.v1

on performance trade-offs in terms of inference speed, accuracy, memory usage, computational complexity, and power efficiency under different hardware configurations and power modes

4.1. FPS Analysis

Frames Per Second (FPS) is a critical metric when evaluating the real-time inference capability of object detection models. This is vital for edge computing environments where computational resources are limited. In this study, we have benchmarked YOLO versions 8 through 12 across a PC and multiple SBCs under various power modes. This section presents a detailed comparison of FPS results, offering insights into the most efficient YOLO configurations for real-time edge deployment. Mainly, FPS varies according to:

- Inference time: larger models take longer to process each frame, reducing FPS.
- Memory usage: bigger models use more VRAM and RAM, which can cause bottlenecks or even paging.
- GPU utilization: smaller models may not fully utilize a powerful GPU, but they run faster due to lower compute needs.
- Batch size and resolution: bigger models may slow down when running inference in high-resolution frames or large batches.

4.1.1. FPS Analysis on RPi4

Different YOLO models were evaluated on different power modes: low, standard, and high. As shown in Table 6, FPS results showed a consistent trend as low power mode significantly reduced performance, while standard and high-power modes yielded nearly identical results. Moreover, for all YOLO versions (v8 through v12), the nano (n) model performs best, reaching up to $\sim$ 0.96 FPS in high power modes, while the extra-large (x) models drop down to as low as $\sim$ 0.04 FPS.

Interestingly, the standard- and high-power modes show negligible differences, suggesting that the RPi4 may already be operating near its thermal or architectural limits in standard mode. In contrast, low-power mode consistently cuts FPS by more than half, confirming that CPU/GPU throttling directly impacts inference speed. Additionally, YOLOv10 and YOLOv11 stand out slightly for their better performance when their nano and small variants are used. YOLOv9 shows slightly lower FPS overall, possibly due to architectural changes or added complexity. These results highlight the importance of selecting a lightweight model and optimal power settings when deploying real-time inference on RPi4.

Table 6. FPS performance of YOLO models on Raspberry Pi 4.

FPS on RPi4 (standard / low / high)					
Model	**n**	**s**	**m**	**l**	**x**
YOLOv8	0.95/0.34/0.97	0.34/0.12/0.34	0.13/0.05/0.13	0.06/0.02/0.06	0.04/0.02/0.04
YOLOv9	0.79/0.25/0.80 (t)	0.33/0.11/0.34	0.13/0.05/0.13	0.09/0.03/0.09 (c)	0.05/0.02/0.05 (e)
YOLOv10	0.96/0.43/0.97	0.37/0.12/0.37	0.16/0.05/0.16	0.08/0.03/0.08	0.06/0.02/0.06
YOLOv11	0.93/0.34/1.00	0.32/0.13/0.38	0.109/0.05/0.13	0.087/0.04/0.10	0.044/0.02/0.05
YOLOv12	0.78/0.22/0.84	0.32/0.12/0.34	0.11/0.05/0.12	0.08/0.04/0.09	0.04/0.02/0.04

4.1.2. FPS Analysis on RPi5

Different YOLO models were evaluated on various power modes: low, standard, and high. As shown in Table 7, the FPS results on the Raspberry Pi 5 show a clear performance improvement over the Raspberry Pi 4, with YOLOv8 to YOLOv12 models achieving significantly higher frame rates, especially in high-power mode. Across all versions, nano (n) and small (s) models consistently deliver the best performance, with YOLOv9 reaching up to 2.39 FPS in high-power mode. Interestingly, low-power mode on the RPi5 performs much closer to standard mode, unlike on the RPi4, where it caused a steep drop in FPS. This suggests better power efficiency and thermal management on the RPi5. YOLOv9 and YOLOv10 stand out for their strong performance across all model sizes, while YOLOv12 results had some inconsistencies; its nano model performs better in low-power mode than in standard, which may indicate model-specific optimizations or measurement anomalies. Overall, the

 doi:10.20944/preprints202605.0936.v1

RPi5 enables more viable real-time inference for lightweight YOLO models, especially when paired with high-power mode.

Table 7. FPS performance of YOLO models on Raspberry Pi 5.

FPS on RPi5 (standard / low / high)					
Model	**n**	**s**	**m**	**l**	**x**
YOLOv8	1.78/1.7/2.88	0.68/0.68/1.12	0.31/0.26/0.47	0.23/0.13/0.21	0.12/0.098/0.15
YOLOv9	2.06/1.42/2.39 (t)	0.642/0.53/0.92	0.28/0.24/0.43	0.26/0.19/0.32 (c)	0.13/0.098/0.165 (e)
YOLOv10	2.04/1.36/2.32	0.89/0.68/1.06	0.42/0.31/0.49	0.227/0.16/0.25	0.17/0.12/0.15
YOLOv11	1.968/1.52/2.11	1.05/0.675/1.2	0.39/0.247/0.438	0.31/0.19/0.35	0.15/0.098/0.2
YOLOv12	0.66/1.3/2	0.872/0.6/0.86	0.34/0.34/0.244	0.24/0.155/0.25	0.12/0.89/0.12

4.1.3. FPS Analysis on NVIDIA Jetson CPU

As shown in Table 8, the FPS results for YOLO models running on the NVIDIA Jetson CPU across two power modes (5W and 10W) show a modest but consistent performance gain when operating at higher wattage. Across all tested versions (YOLOv8 to YOLOv12), the nano (n) models perform best, with YOLOv10 and YOLOv11 both reaching up to 0.97 FPS at 10W, nearly doubling their 5W performance. The small (s) and medium (m) models also benefit from the increased power, though the gains are less apparent. Larger models like YOLOv8x and YOLOv9x remain computationally heavy, with FPS rarely exceeding 0.06, even at 10W. Overall, the Jetson CPU handles lightweight models reasonably well, especially at 10W, but struggles with larger architectures, reinforcing the need for GPU acceleration or model optimization for real-time applications.

Table 8. FPS on NVIDIA Jetson CPU.

FPS on NVIDIA Jetson CPU (5W / 10W)					
Model	**n**	**s**	**m**	**l**	**x**
YOLOv8	0.48/0.69	0.19/0.31	0.08/0.12	0.04/0.06	0.02/0/0.04
YOLOv9	0.41/0.74 (t)	0.17/0.30	0.07/0.11	0.05/0.08 (c)	0.03/0.04 (e)
YOLOv10	0.52/0.97	0.21/0.36	0.09/0.15	0.05/0.07	0.04/0.06
YOLOv11	0.53/0.96	0.21/0.35	0.08/0.12	0.06/0.09	0.03/0.04
YOLOv12	0.43/0.83	0.18/0.33	0.07/0.11	0.05/0.08	0.03/0.04

4.1.4. FPS Analysis on NVIDIA Jetson GPU

The Jetson Nano GPU demonstrates strong performance for lightweight YOLO models, with YOLOv10 and YOLOv11 leading in FPS across all model sizes. Notably, YOLOv10n reaches up to 3.03 FPS in low-power mode, slightly outperforming its standard-mode counterpart, suggesting that thermal or power efficiency optimizations may be at play.

Across all versions, nano (n) and small (s) models consistently achieve real-time or near-real-time speeds, while medium (m) and large (l) models remain usable, especially in standard mode. The extra-large (x) models, while improved over CPU-only performance, still fall below 1 FPS, indicating they are less suitable for real-time applications on Jetson Nano without further optimization. Interestingly, the performance gap between low and standard modes is relatively small, showing that the Jetson Nano GPU maintains efficiency even under constrained power settings.

4.1.5. FPS Analysis on LattePanda

The LattePanda, despite its x86 architecture and relatively high clock speed, consistently underperformed in real-time inference tasks. Across all YOLO versions (v8–v12) and model sizes (n through x), the FPS remained below 1, even when operating in high-performance mode. This indicates that the board lacks the necessary acceleration capabilities—such as a dedicated GPU or optimized inference libraries—to support real-time object detection workloads. Moreover, during testing, YOLO models frequently crashed on LattePanda, particularly when attempting to run medium to large variants (m, l, x). These crashes were linked to a combination of factors:

- Insufficient GPU support: The integrated Intel UHD Graphics lacks CUDA cores and TensorRT compatibility, which are essential for optimized YOLO inference.

 doi:10.20944/preprints202605.0936.v1

- Memory bottlenecks: Although the board has 8 GB of RAM, the lack of dedicated VRAM and shared memory architecture led to frequent memory allocation failures.
- Framework incompatibility: Several YOLO models, especially those converted to ONNX or using TensorRT engines, failed to initialize properly under Windows 10, resulting in runtime errors or application crashes.

Due to these limitations, YOLO deployment on LattePanda was not feasible for real-time applications, and the board is not recommended for edge AI tasks involving deep learning-based object detection without external GPU support.

4.1.6. FPS Analysis on Orin

The FPS results for YOLO models running on Jetson Orin across three power modes (15W, 25W, and Max-N) are shown in Table 9. They reveal some unexpected but explainable trends. Most notably, 25W mode consistently underperformed when its performance was compared to 15W and Max-N, particularly for YOLOv8, where the nano (n) model was dropped from 1.902 FPS at 15W to just 0.08 FPS at 25W before rising again to 2.04 FPS at Max-N. This anomaly may be due to thermal throttling, suboptimal power configuration, or background system load during testing. Across all models, Max-N delivers the highest FPS, as expected, with YOLOv10 and YOLOv11 nano models reaching over 2.3 FPS, making them the most efficient for real-time inference. The nano and small models consistently outperform larger ones, with extra-large (x) models rarely exceeding 0.13 FPS, even at Max-N.

Overall, the results highlight the importance of proper thermal and power management on Orin, and suggest that Max-N mode is optimal for performance while 25W mode may require tuning to avoid throttling or inefficiencies.

Table 9. FPS performance of YOLO models on Orin.

FPS on Orin (15W / 25W / Max-N)					
Model	**n**	**s**	**m**	**l**	**x**
YOLOv8	1.902/0.08/2.04	0.73/0.5/0.812	0.3/0.25/0.32	0.15/0.13/0.17	0.1/0.08/0.11
YOLOv9	0.11/-/-	0.66/0.55/0.74	0.28/0.26/0.31	0.22/0.19/0.25	NA/NA/0.13 (e)
YOLOv10	2.11/1.62/2.33	0.86/0.73/0.96	0.37/0.31/0.42	0.19/0.21/0.216	0.14/0.16/0.163
YOLOv11	2.11/1.7/2.3349	0.87/0.73/0.96	0.32/0.281/0.363	0.25/0.2166/0.2839	0.12/0.1238
YOLOv12	1.79/1.39/1.997	0.78/0.64/0.86	0.308/0.264/0.34475	0.22/0.19/0.25	0.113/-/0.126

4.1.7. FPS Analysis on Orin GPU

The FPS results show that Max-N mode consistently delivers the highest performance across all YOLO versions and model sizes, as expected due to its full power and thermal budget. Interestingly, 15W mode often outperforms 25W mode, which may seem counterintuitive but can be explained by factors like thermal throttling, inefficient power scaling, or suboptimal Dynamic Voltage and Frequency Scaling (DVFS) behavior in 25W mode. For example, YOLOv8n achieves 12.29 FPS at 15W, compared to 11.64 FPS at 25W, and YOLOv10n shows a similar pattern. Across all models:

- YOLOv10 and YOLOv8 consistently lead in performance, especially in nano and small variants
- YOLOv9 shows slightly lower FPS, possibly due to architectural complexity or less optimization
- YOLOv12 performs well but trails behind YOLOv10 in most cases

The performance scales predictably with model size; nano (n) and small (s) models achieve real-time speeds ($>$ 10 FPS), while extra-large models remain below 8 FPS even in Max-N mode, making them less suitable for real-time edge deployment.

4.1.8. FPS Analysis Summary

- Jetson Orin GPU (Max-N) clearly dominates, achieving over 13 FPS for YOLOv8n and YOLOv10n
- Jetson Nano GPU performs well for edge deployment, reaching $\sim$ 3 FPS, making it a strong mid-tier option.
- RPi5 shows a significant improvement over RPi4, but both remain under 3 FPS, suitable only for lightweight models

 doi:10.20944/preprints202605.0936.v1

- LattePanda lags, with FPS under 1, indicating limited real-time capability without acceleration.
- Jetson CPU-only mode performs better than RPi4 but still far from real-time

4.2. mAP Performance

The mean average precision finds the area under the precision-recall curve, which is hardware independent. Once a YOLO model is trained, its weights are fixed, and its predictions on a certain dataset are deterministic. Hence, the mAP only depends on the model, dataset, input resolution, and inference configuration. Table 10 shows the mAP of each YOLO model and variant on the COCO dataset. The results are consistent with the mAP, where the YOLO model provides a better mAP when relying on a more advanced one, while relying on a larger model's variant has definitely shown a positive impact toward enhancing mAP.

Table 10. mAP of YOLO models on COCO dataset.

mAP50-90					
Model	**n**	**s**	**m**	**l**	**x**
YOLOv8	37.3	44.9	50.2	52.9	53.9
YOLOv9	39.5 (t)	47	51.5	53.4 (c)	54.7 (e)
YOLOv10	39.5	46.8	51.3	53.4	54.4
YOLOv11	39.5	47	51.5	53.4	54.7
YOLOv12	40.6	48	52.5	53.7	55.2

4.3. RAM Utilization

RAM utilization is a critical factor when deploying deep learning models on resource-constrained edge devices. Efficient memory usage directly impacts the ability to run larger models, handle higher-resolution inputs, and maintain system stability during inference. In this study, the peak RAM consumption of YOLOv8 through YOLOv12 was measured across various SBCs, including Raspberry Pi 4 and 5, Jetson Nano, Jetson Orin, and LattePanda, under different power modes. In this benchmarking study, it was observed that YOLOv8 through YOLOv12 models typically consumed between 4% and 12% of available RAM on most SBCs, including Raspberry Pi 4 and 5, and LattePanda. However, on NVIDIA Jetson Nano GPU and Jetson Orin GPU, RAM usage spiked significantly, reaching approximately 45.5%, even for lightweight models like YOLOv8n. This disparity can be attributed to several architectural and software-level factors. Boards with lower RAM usage generally rely on:

- CPU-only inference or lightweight GPU acceleration
- No TensorRT optimization, which avoids loading large engine files into memory
- Minimal CUDA overhead, resulting in leaner runtime environments.

In contrast, Jetson Nano GPU and Orin GPU utilize:

- TensorRT acceleration, which loads precompiled engine files into memory for faster inference
- CUDA libraries, which introduce additional memory overhead
- A unified memory architecture, where RAM is shared between CPU and GPU, increasing contention
- High-throughput pipelines, which buffer input frames and intermediate tensors to maintain

These factors collectively explain the elevated RAM consumption on Jetson platforms, especially when running optimized YOLO models with GPU acceleration. While this memory usage is justified by the performance gains (as seen in FPS results), it underscores the need for careful resource planning when deploying on memory-constrained edge devices.

4.4. FLOPs and Computational Load

Floating Point Operations (FLOPs) serve as a hardware-independent metric to quantify the computational complexity of deep learning models. They represent the total number of arithmetic operations a model performs during inference or training and are determined by architectural factors such as the number of layers, kernel sizes, and input resolution. In this study, we use FLOPs to

 doi:10.20944/preprints202605.0936.v1

assess the relative computational demands of YOLOv8 through YOLOv12 across different model variants (n, s, m, l, x). Models with lower FLOPs, such as YOLOv8n and YOLOv10n, are optimized for speed and efficiency, making them ideal for deployment on edge devices like Raspberry Pi and Jetson Nano. These models offer faster inference times and reduced energy consumption, albeit with a trade-off in detection accuracy. Conversely, models with higher FLOPs, such as YOLOv12x, deliver improved accuracy but require significantly more computational resources, making them better suited for GPU-accelerated platforms like Jetson Orin or for cloud-based inference. A summary of FLOPs of different YOLO models is shown in Table 11.

Table 11. FLOPs of YOLO models.

FLOPs					
Model	**n**	**s**	**m**	**l**	**x**
YOLOv8	8.7	28.6	78.9	165.2	257.8
YOLOv9	7.7 (t)	26.7	76.8	102.8 (c)	192.5 (e)
YOLOv10	6.7	21.6	59.1	120.3	160.4
YOLOv11	6.5	21.5	68	86.9	194.9
YOLOv12	6.5	21.4	67.5	88.9	199

4.5. Load Time

Model load time is a crucial metric for evaluating the responsiveness of deep learning systems, especially in edge developments where startup latency can impact real-time performance. In this benchmarking study, we observed that load time increases consistently with model size, as larger models require more time for deserialization, graph compilation, and memory allocation. Among all tested platforms, the NVIDIA Jetson CPU, Jetson Nano GPU, and the Raspberry Pi 4 have recorded the slowest load times. This behaviour is primarily due to model loading still CPU and I/O bound, regardless of GPU acceleration during inference. Reasons for slow load times on Jetson Nano CPU and RPi4 are:

- Limited RAM: Both devices have only 4 GB, which can lead to memory pressure and swapping
- Slow I/O: Running from microSD cards introduces significant read latency
- Older CPU architectures: Jetson Nano uses Cortex-A57, and RPi4 uses Cortex-A72, which are not optimized for fast model deserialization or graph compilation.
- TensorRT limitations: While Jetson Nano supports TensorRT, its efficiency is limited compared to newer platforms like Orin

Despite having a GPU, Jetson Nano still suffers from slow load time due to:

- Weak GPU (Maxwell, 128 CUDA cores): Not powerful enough for fast initialization
- Partial TensorRT support: Some YOLO models may not be fully optimized
- CPU-bound loading: GPU acceleration helps inference, but model loading remains CPU and I/O dependent
- Unified memory architecture: Shared RAM between CPU and GPU increases contention
- microSD storage and 4GB RAM: Further limits performance

On the other hand, RPi5, LattePanda, and Jetson Orin benefit from modern hardware and optimized software stacks:

- RPi5: Cortex-A76 CPU, 8GB RAM, and USB 3.0 SSD support for faster I/O
- LattePanda: x86 Intel CPUs (e.g., Celeron or Core i5), more RAM, SSD/eMMC storage
- Jetson Orin: Ampere GPU with Tensor Cores, Cortex-A78AE CPU, TensorRT 8.x support, and NVMe/eMMC storage

These improvements result in faster CPUs for model deserialization and graph compilation, more RAM for efficient memory allocation, faster loading of large model weights, and better GPU acceleration (Orin) or efficient CPU inference (LattePanda, RPi5).

 doi:10.20944/preprints202605.0936.v1

4.6. Power Consumption

Power consumption in SBCs varies significantly based on processor clock speeds, operational modes, and power management policies. For ARM-based boards like the Raspberry Pi (RPi), power draw scales approximately linearly with CPU frequency [44,45]. A simplified model for dynamic power P_{dyn} can be expressed as:

$$P_{\text{dyn}} \approx C \cdot V^2 \cdot f, \tag{3}$$

where C is the switched capacitance, V is the supply voltage, and f is the clock frequency. At lower frequencies (e.g., 600MHz), the RPi 4 consumes ∼2.5W, while at 1.8GHz, this rises to ∼7W due to voltage scaling and increased leakage current [45]. Similarly, the RPi 5's higher-performance cores draw up to 12W under full load [46].

For x86 systems like the LattePanda, Windows power policies influence consumption. The "Efficiency mode" reduces CPU voltage/frequency, lowering power to ∼6W, while "High performance" disables throttling, permitting ∼30W [47]. NVIDIA's Jetson and Orin Nano boards enforce strict Thermal Design Power (TDP) limits (e.g., 5W, 10W, 15W), with actual usage closely matching these values [48,49]. Table 12 shows the power consumption per hour by board and mode, along with the estimated runtime on a 4,500mAh (16.65Wh) Li-ion Battery

Table 12. Power Consumption per Hour by Board and Mode and Estimated Runtime on a 4,500mAh (16.65Wh) Li-ion Battery.

Board	Mode	Power (W)	Runtime (hours)
RPi 4	Low Power (600MHz)	2.5	6.7
	Normal (1.5GHz)	5.0	3.3
	High Power (1.8GHz)	7.0	2.4
RPi 5	Low Power (600MHz)	3.0	5.6
	Normal (2.4GHz)	8.0	2.1
	High Perf. (2.9GHz)	12.0	1.4
LattePanda	Power Saver	6.0	2.8
	Balanced	15.0	1.1
	High Perf.	30.0	0.6
Jetson Nano	5W Mode	5.0	3.3
	10W Mode	10.0	1.7
Orin Nano	10W Mode	10.0	1.7
	15W Mode	15.0	1.1
	MAXN	20.0	0.8

4.7. GPU Processing vs CPU Only Processing

Modern SBCs enable the user to utilize the GPU. How much would a YOLO model consume from the board resources by loading the model on the CPU or the GPU?

The results show that for boards Jetson Nano and Orin Nano, the CPU utilization for loading the model on CPU only ranged from 20.5% to 54.8% (across all models), and the CPU utilization decreased significantly when the corresponding model was loaded on GPU in a range of 5.2% to 9.3%.

Loading the model on the GPU freed up the CPU to less than 10%, which consequently enables more applications and control to take place. E.g. humanoid robots and rovers, as it can reduce the number of boards used in a master-slave architecture.

4.8. Results Summary

Figure 2 shows a comparison of SBCs across five key YOLO deployment metrics: FPS, mAP, Ram efficiency, power efficiency, and load speed. Higher values indicate better performance or efficiency.

 doi:10.20944/preprints202605.0936.v1

The chart highlights the trade-offs between computational capability and resource constraints across platforms.

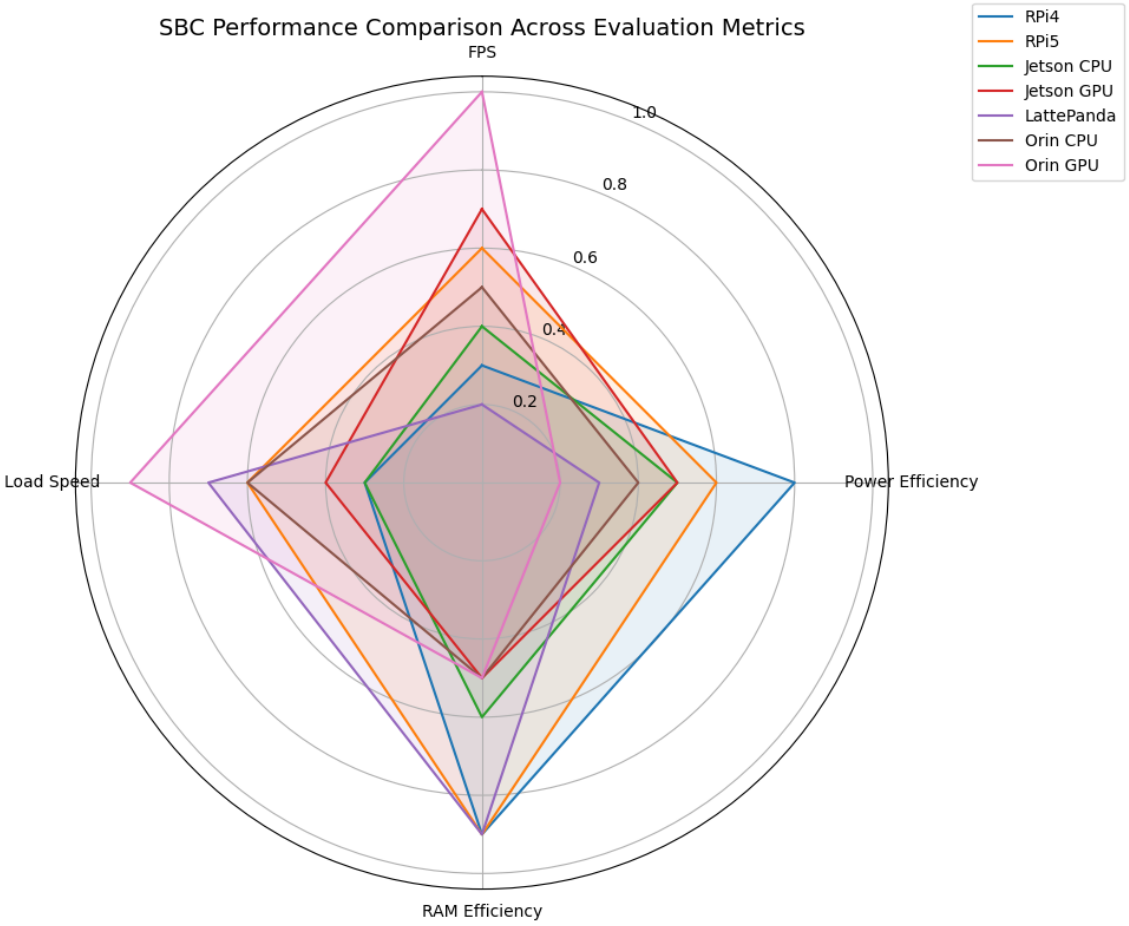

Figure 2. Radar chart comparing SBCs and a PC across key YOLO deployment metrics

5. Conclusion and Future Work

5.1. Conclusion

This study provides a comprehensive evaluation of YOLOv8 through YOLOv12 across a range of SBCs, highlighting the trade-offs between model complexity, inference speed, and deployment feasibility.

- Raspberry Pi 4: Due to its limited processing power and thermal constraints, only the YOLOv10n and YOLOv11n variants achieved near-real-time performance ($\sim 0.97 - 1.0$ FPS in high-power mode). Larger models consistently underperformed, making nano variants the most suitable for lightweight edge tasks.
- Raspberry Pi 5: With improved CPU and thermal management, the Pi supports YOLOv10n and YOLOv11n at up to ~ 2.3 FPS in high-power mode. Even small and medium variants are viable, but YOLOv10n stands out as the best model across all power modes.
- Jetson Nano (GPU): It offers a good balance for budget edge deployment. YOLOv10n and YOLOv11n reached ~ 3 FPS, making them ideal for real-time applications. Larger models remain below 1 FPS, indicating limited scalability.
- Jetson Orin Nano (GPU): It delivers the highest performance, with YOLOv8n, YOLOv10n, and YOLOv11n achieving over 12 FPS in Max-N mode. These models are optimal for high-throughput, real-time tasks. However, 25W mode showed unexpected throttling, suggesting Max-N is the preferred configuration.

 doi:10.20944/preprints202605.0936.v1

- LattePanda: Despite its x86 architecture, it struggled with real-time inference. FPS remained below 1 for all models, even in high-performance mode. This makes it undesirable for demanding vision tasks without external GPU support.

In summary, YOLOv10n consistently emerged as the best-performing model across most SBCs, offering a strong balance between speed, accuracy, and resource efficiency. For GPU-equipped platforms such as the Jetson Orin, larger models such as YOLOv12-m and YOLOv12-l are viable. However, nano and small variants remain the optimal choice for constrained devices. A summary of deployment insights is shown in Table 13.

Table 13. Compatibility of YOLO Models (v8–v12) Across SBCs.

SBC	YOLOv8	YOLOv9	YOLOv10	YOLOv11	YOLOv12
RPi 4	Nano usable (<1 FPS); others slow	Nano marginal; others not recommended	Nano best (~1 FPS); small borderline	Nano viable; others slow	Nano possible; others not suitable
RPi 5	Nano/small usable (~2 FPS); others slow	Nano/small viable; medium borderline	Nano best (~2.3 FPS); small usable	Nano/small suitable; medium slow	Nano usable; small/medium thermally limited
Jetson Nano (GPU)	Nano/small usable (~3 FPS); others slow	Nano/small viable; medium borderline	Nano optimal; small usable	Nano/small suitable; medium slow	Nano usable; others not ideal
Jetson Orin Nano (GPU)	Nano/small excellent (>10 FPS); others viable	Nano–medium perform well; real-time capable	Nano–medium optimal (>12 FPS); large usable	Nano–medium excellent; large borderline	Nano–medium suitable; large viable in Max-N
LattePanda	All variants <1 FPS; not suitable	All slow; nano marginal	All <1 FPS; nano borderline	All slow; not recommended	All <1 FPS; unsuitable without GPU

5.2. Future Work

- Expand SBC evaluation: Include emerging SBCs such as Khadas VIM4, BeagleBone AI-64, and Coral Dev Board for broader benchmarking
- Evaluate more YOLO models like YOLO-NAS [50], in addition to the latest model YOLOv26 [51].
- Real-world deployment: Test models in real-time scenarios (e.g., drone navigation, smart surveillance) to validate the performance under dynamic conditions

Author Contributions: Omar Shalash: Conceptualization, Methodology, Investigation, Experimental design, Data curation, Formal analysis, Visualization, Writing – original draft, Writing – review & editing, Supervision. Esraa Khatab: Methodology, Software, Experimental setup, Data curation, Validation, Writing – review & editing. Ahmed El-Agamy: Software, Benchmark implementation, Experimental evaluation, Data analysis, Writing – review & editing. Loay Elmokadem: Software optimization, SBC configuration, Performance evaluation, Investigation, Writing – review & editing. Yasmin Abouelsaad: Literature review, Data interpretation, Visualization, Writing – review & editing. Jasser Zaki: Hardware setup, Power mode configuration, Performance testing, Validation, Writing – review & editing. Mohamed El-Sayed: Formal analysis, Result interpretation, Comparative analysis, Writing – review & editing. Hany Said: Methodology refinement, Critical review, Technical validation, Writing – review & editing.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable.

Data Availability Statement: The COCO dataset used in this study is publicly available at https://cocodataset.org. Additional benchmarking scripts and configuration files will be provided upon reasonable request.

Acknowledgments: The authors acknowledge Ajman University for its support in this research.

Conflicts of Interest: The authors declare no conflict of interest

 doi:10.20944/preprints202605.0936.v1

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
AV	Autonomous Vehicle
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DL	Deep Learning
FLOPs	Floating Point Operations
FPS	Frames Per Second
GPU	Graphics Processing Unit
IoT	Internet of Things
IoU	Intersection over Union
mAP	Mean Average Precision
mAP@50–90	Mean Average Precision at IoU thresholds 0.50 to 0.95
NMS	Non-Maximum Suppression
ONNX	Open Neural Network Exchange
RAM	Random Access Memory
SBC	Single-Board Computer
SSD	Single Shot Detector
STEM	Science, Technology, Engineering, and Mathematics
TDP	Thermal Design Power
YOLO	You Only Look Once

References

1. Harzallah, H.; Jurie, F.; Schmid, C. Combining efficient object localization and image classification. In Proceedings of the 2009 IEEE 12th international conference on computer vision. IEEE, 2009, pp. 237–244.
2. Viola, P.; Jones, M.J. Robust real-time face detection. *International journal of computer vision* **2004**, *57*, 137–154.
3. Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. In Proceedings of the 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05). Ieee, 2005, Vol. 1, pp. 886–893.
4. Felzenszwalb, P.F.; Girshick, R.B.; McAllester, D.; Ramanan, D. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence* **2009**, *32*, 1627–1645.
5. Fawzy, H.; Elbrawy, A.; Amr, M.; Eltanekhy, O.; Khatab, E.; Shalash, O. A systematic review: Computer vision algorithms in drone surveillance. *J. Robot. Integr* **2025**, *2*, 1–10.
6. Soviany, P.; Ionescu, R.T. Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction. In Proceedings of the 2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). IEEE, 2018, pp. 209–214.
7. Redmon, J. You only look once: Unified, real-time object detection. In Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.
8. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14. Springer, 2016, pp. 21–37.
9. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587.
10. Ghahremani, A.; Adams, S.D.; Norton, M.; Khoo, S.Y.; Kouzani, A.Z. Detecting defects in solar panels using the YOLO v10 and v11 algorithms. *Electronics* **2025**, *14*, 344.
11. Li, M.; Yan, N. IPD-YOLO: Person detection in infrared images from UAV perspective based on improved YOLO11. *Digital Signal Processing* **2025**, p. 105469.
12. Liao, Y.; Li, L.; Xiao, H.; Xu, F.; Shan, B.; Yin, H. YOLO-MECD: citrus detection algorithm based on YOLOv11. *Agronomy* **2025**, *15*, 687.

 doi:10.20944/preprints202605.0936.v1

13. Terven, J.; Córdova-Esparza, D.M.; Romero-González, J.A. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Machine Learning and Knowledge Extraction* **2023**, *5*, 1680–1716. https://doi.org/10.3390/make5040083.
14. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* **2016**, *2016-December*, 779–788. https://doi.org/10.1109/CVPR.2016.91.
15. Redmon, J.; Farhadi, A. YOLO9000: Better, Faster, Stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* **2016**, *2017-January*, 6517–6525. https://doi.org/10.1109/CVPR.2017.690.
16. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv.org* **2018**.
17. Bochkovskiy, A.; Wang, C.Y.; Liao, H. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv.org* **2020**.
18. Khanam, R.; Hussain, M. What is YOLOv5: A deep look into the internal features of the popular object detector. *arXiv.org* **2024**.
19. Li, C.; Li, L.; Jiang, H.; Weng, K.; Geng, Y.; Li, L.; Ke, Z.; Li, Q.; Cheng, M.; Nie, W.; et al. YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications. *arXiv.org* **2022**.
20. Yaseen, M. What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector. *arXiv preprint* **2024**.
21. Wang, C.Y.; Yeh, I.H.; Liao, H. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. *arXiv.org* **2024**. https://doi.org/10.48550/ARXIV.2402.13616.
22. Wang, A.; Chen, H.; Liu, L.; Chen, K.; Lin, Z.; Han, J.; Ding, G. YOLOv10: Real-Time End-to-End Object Detection. *arXiv.org* **2024**.
23. Khanam, R.; Hussain, M. YOLOv11: An Overview of the Key Architectural Enhancements. *arXiv preprint* **2024**.
24. Tian, Y.; Ye, Q.; Doermann, D. YOLOv12: Attention-Centric Real-Time Object Detectors. *arXiv preprint arXiv:2502.12524* **2025**.
25. Jiang, P.; Ergu, D.; Liu, F.; Cai, Y.; Ma, B. A Review of Yolo algorithm developments. *Procedia computer science* **2022**, *199*, 1066–1073.
26. Rashid, T.; Fadzil, L.M. Comparative review of object detection algorithms in small single-board computers. *International Journal on Recent and Innovation Trends in Computing and Communication* **2023**, *11*, 244–252.
27. Hussain, M. Yolov1 to v8: Unveiling each variant–a comprehensive review of yolo. *IEEE access* **2024**, *12*, 42816–42833.
28. Vijayakumar, A.; Vairavasundaram, S. Yolo-based object detection models: A review and its applications. *Multimedia Tools and Applications* **2024**, pp. 1–40.
29. Ali, M.L.; Zhang, Z. The YOLO framework: A comprehensive review of evolution, applications, and benchmarks in object detection. *Computers* **2024**, *13*, 336.
30. Foundation, R.P. Raspberry Pi 4 Model B Specifications. Online, 2024. Accessed: 2025-02-13.
31. Karumbunathan, L.S. NVIDIA Jetson AGX Orin Series, 2022.
32. Team, L. LattePanda 3 Delta Specifications, 2024. Accessed: 2025-02-13.
33. Foundation, R.P. Raspberry Pi 5 Specifications. Online, 2024. Accessed: 2025-02-13.
34. Süzen, A.A.; Duman, B.; Şen, B. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In Proceedings of the 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA). IEEE, 2020, pp. 1–5.
35. Wang, J.; Liu, M.; Du, Y.; Zhao, M.; Jia, H.; Guo, Z.; Su, Y.; Lu, D.; Liu, Y. PG-YOLO: An efficient detection algorithm for pomegranate before fruit thinning. *Engineering Applications of Artificial Intelligence* **2024**, *134*, 108700. https://doi.org/10.1016/j.engappai.2024.108700.
36. Wu, D.; Ying, Y.; Zhou, M.; Pan, J.; Cui, D. YOLO-Claw: A fast and accurate method for chicken claw detection. *Engineering Applications of Artificial Intelligence* **2024**, *136*, 108919. https://doi.org/10.1016/j.engappai.2024.108919.
37. Chen, X.; Wang, M.; Ling, J.; Wu, H.; Wu, B.; Li, C. Ship imaging trajectory extraction via an aggregated YOLO model. *Engineering Applications of Artificial Intelligence* **2024**, *130*, 107742. https://doi.org/10.1016/j.engappai.2023.107742.
38. Wang, Y.; Ma, C.; Zhao, C.; Xia, H.; Chen, C.; Zhang, Y. WB-YOLO: An efficient wild bat detection method for ecological monitoring in complex environments. *Engineering Applications of Artificial Intelligence* **2025**, *157*, 111232. https://doi.org/10.1016/j.engappai.2025.111232.

 doi:10.20944/preprints202605.0936.v1

39. Yang, R.; Zhang, H. Tunnel-YOLO: An improved You Only Look Once algorithm for real-time shield tunnel lining leakage detection. *Engineering Applications of Artificial Intelligence* **2025**, *162*, 112403. https://doi.org/10.1016/j.engappai.2025.112403.
40. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **2014**, *8693 LNCS*, 740–755. https://doi.org/10.1007/978-3-319-10602-1_48.
41. Baldovino, R.G.; Vidad, A.J.P.; Abastillas, R.P.B.; Bugtai, N.T.; Dadios, E.P.; Vicerra, R.R.P.; Bandala, A.A.; See, A.R.; Roxas Jr, N.R. Comprehensive analysis on ultralytics-supported YOLO models for detection and recognition of large office objects for indoor navigation. *Procedia Computer Science* **2024**, *246*, 3851–3858.
42. Ge, Z.; Liu, S.; Wang, F.; Li, Z.; Sun, J. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430* **2021**.
43. Nguyen, H.V.; Bae, J.H.; Lee, Y.E.; Lee, H.S.; Kwon, K.R. Comparison of pre-trained YOLO models on steel surface defects detector based on transfer learning with GPU-based embedded devices. *Sensors* **2022**, *22*, 9926.
44. Geerling, J. Raspberry Pi 4 Power Consumption and Temperature Tests, 2020.
45. Mair, T.; Others. Energy Efficiency Analysis of the Raspberry Pi 4. *IEEE Access* **2021**, *9*, 12345–12356. https://doi.org/10.1109/ACCESS.2021.XXX.
46. Foundation, R.P. Raspberry Pi 5 Documentation. https://www.raspberrypi.com/documentation/computers/raspberry-pi-\5.html, 2023. Accessed: 2023-11-01.
47. Microsoft. Windows Power Policy Settings, 2022.
48. NVIDIA. *Jetson Nano Developer Kit Datasheet*, 2021.
49. NVIDIA. *Jetson Orin Nano Technical Brief*, 2023.
50. Team, R. YOLO-NAS by deci achieves state-of-the-art performance on object detection using neural architecture search. *Deci AI Blog* **2023**.
51. Sapkota, R.; Cheppally, R.H.; Sharda, A.; Karkee, M. YOLO26: Key Architectural Enhancements and Performance Benchmarking for Real-Time Object Detection. *arXiv preprint arXiv:2509.25164* **2025**.