

Computational Performance of ONNX Runtime and NCNN Frameworks for Real-Time Motorcycle Detection on Raspberry Pi 5: A Yogyakarta CCTV Case Study

Rifqi Nabila Zufar^{1*}, Ahmad Baihaqi², Hidayatul Munawaroh³, Vriza Wahyu Saputra⁴,
Abi Nizar Sutranggono⁵

^{1*,2,3,4,5}Department of Informatics, Faculty of Health, Science and Technology, Universitas Islam Negeri
Sayyid Ali Rahmatullah Tulungagung, Tulungagung, 66224, Indonesia

^{1*}rifqinabilazufar@uinsatu.ac.id

²ahmadbaihaqi@uinsatu.ac.id

³hidayatulumunawaroh@uinsatu.ac.id

⁴vriza@uinsatu.ac.id

⁵abinizar@uinsatu.ac.id

Abstract

The acceleration of Smart City initiatives across Indonesia has intensified the demand for localized Intelligent Transportation Systems. Although micro-architectures like YOLOv26n offer optimized layer configurations, executing these models for real-time motorcycle detection on standalone edge computers often presents hardware bottlenecks due to software runtime inefficiencies and core thermal stress. This study presents a comparative computational performance benchmarking of the out-of-the-box YOLOv26n architecture across two prominent runtime frameworks: ONNX Runtime and the NCNN framework. The evaluation scope is strictly bounded to hardware resource utilization and physical stability bounds on a Raspberry Pi 5 platform. Telemetry data were logged frame-by-frame at a granular millisecond-level sampling rate over an uninterrupted 194-second capture window under a live HTTP Live Streaming (HLS) traffic feed from the Pasar Gading Traffic Light areas in Yogyakarta. The empirical results demonstrate that while ONNX Runtime achieves a throughput of 4.92 FPS, it subjects the hardware to severe processor strain, locking the mean CPU usage at 95.83% and causing core temperatures to reside consistently within the active thermal throttling zone (80 °C to 85 °C). Conversely, the zero-dependency NCNN framework successfully alleviates hardware stress by incorporating low-level hand-written ARM NEON SIMD assembly instructions. NCNN accelerates the processing throughput to 7.36 FPS while simultaneously relaxing the average CPU utilization down to 77.61% and lowering the steady-state memory footprint down to a mean of 1,500.13 MB. Ultimately, this benchmark establishes that assembly-level compiler optimizations via NCNN offer the most viable engineering deployment roadmap for standalone edge-based urban traffic surveillance nodes.

Keyword: Edge Computing, YOLOv26n, Raspberry Pi 5, NCNN, ONNX Runtime, Motorcycle Detection.

I. INTRODUCTION

The evolution of Smart City frameworks in Indonesia has heavily prioritized Intelligent Transportation Systems (ITS) to mitigate urban congestion and improve traffic safety, particularly in metropolitan areas like Yogyakarta, where motorcycle volume heavily dominates the transportation landscape [1]. While the provincial government provides continuous traffic visibility via public HTTP Live Streaming (HLS) surveillance feeds with (.m3u8) extensions, traditional traffic monitoring still suffers from structural inefficiencies [2]. Relying

on centralized cloud-based computing for video analytics introduces significant bandwidth bottlenecks and transmission delays, which severely restrict time-critical traffic interventions. Consequently, decentralizing artificial intelligence to local nodes, known as Edge Computing, has emerged as a promising paradigm to enable localized, low-latency traffic monitoring directly at the source [3],[4],[5].

Deploying state-of-the-art deep learning architectures on resource-constrained edge hardware poses a severe hardware optimization challenge [6],[7]. While the newly introduced

YOLOv26n micro-architecture from Ultralytics offers compressed layer parameters optimized for generic object detection, its runtime throughput is heavily constrained by the choice of the software execution framework [8]. Standard deployment configurations using native wrappers introduce high software dependency overheads and suboptimal memory management, which frequently cause high CPU resource contention [9]. Sustained computational loads under these unoptimized runtimes elevate the SoC (System on Chip) core temperature, breaching the hardware's safety thresholds and triggering dynamic frequency scaling commonly known as thermal throttling. This hardware bottleneck directly degrades frame processing rates and causes severe latency fluctuations during live video stream ingestion.

A prominent research gap exists in contemporary edge AI literature: while extensive studies focus on training customized weights or validating model accuracy frameworks, there is a distinct lack of empirical engineering data analyzing how low-level runtime engine backends handle continuous, high-throughput public HLS video streams on standalone ARM CPUs under physical thermal boundaries. To address this gap, this research presents a novel comparative computational benchmarking evaluation of the out-of-the-box YOLOv26n architecture across two distinct runtime frameworks: ONNX Runtime, representing the universal cross-platform execution baseline, and NCNN, a dedicated, zero-dependency inference engine manually optimized with ARM NEON SIMD (Single Instruction Multiple Data) assembly instructions.

The scope of this research is strictly bounded to hardware-level computational performance metrics and thermal stability profiling under continuous data ingestion. To ensure empirical precision, the evaluation constraints are strictly limited to the execution profiles of the Raspberry Pi 5 architecture ingest-processing a real-time public live stream from the Pasar Gading Traffic Light areas in Yogyakarta. Spatial object-detection accuracy metrics (such as Precision, Recall, and mAP) fall outside the empirical boundaries of this study, as the core engineering objective is to investigate the runtime efficiency, latency bottlenecks, and thermal safety margins of the hardware when executing standard serialized graphs. Ultimately, the purpose and usefulness of this article are to provide a rigorous, hardware-aware engineering roadmap for smart city developers to deploy stable, high-throughput, and thermally resilient edge surveillance nodes

tailored to the specific infrastructure constraints of Indonesian smart cities.

II. THEORY

A. Intelligent Transportation Systems (ITS) and Smart Cities

The concept of the Smart City has emerged as a comprehensive framework to enhance urban livability, efficiency, and sustainability through the integration of Information and Communication Technologies (ICT) [10]. A fundamental pillar of the Smart City ecosystem is the Intelligent Transportation System (ITS), which leverages advanced sensing, communication, and computing technologies to manage urban mobility. In recent years, ITS has transitioned from simple traffic signal control to complex, data-driven systems capable of real-time monitoring and autonomous decision-making. The primary goal of modern ITS is to mitigate urban challenges such as traffic congestion, accidents, and environmental pollution by optimizing the flow of vehicles and pedestrians [11].

In the context of developing urban areas like Indonesia, particularly in Yogyakarta, ITS implementation faces unique challenges due to highly heterogeneous traffic patterns dominated by motorcycles. Traditional traffic monitoring relies heavily on human operators or centralized cloud-based systems. However, as the number of surveillance cameras such as those integrated into the Yogyakarta Smart City [12] infrastructure increases, manual monitoring becomes unscalable. Furthermore, relying on cloud architectures for processing high-definition HLS(m3u8) streams often leads to significant network latency and bandwidth bottlenecks, which are detrimental to time-sensitive traffic management.

Recent advancements in Computer Vision and Artificial Intelligence (AI) have introduced automated object detection as a core component of ITS. By utilizing deep learning architectures like YOLO [13], city authorities can automatically classify and count vehicles to gain insights into traffic density. However, current literature emphasizes a growing shift toward decentralized processing, or Edge Computing, to address the shortcomings of cloud-based ITS. By performing AI inference directly at the edge using devices like the Raspberry Pi 5 cities can achieve faster response times and reduce the load on network infrastructure [14].

B. Object Detection with YOLO Architecture

Object detection has undergone a paradigm shift with the introduction of the You Only Look Once (YOLO) architecture. Unlike traditional

two-stage detectors such as Faster R-CNN, which perform region proposal and classification in separate steps, YOLO treats object detection as a single regression problem. By predicting bounding boxes and class probabilities directly from full images in one evaluation, YOLO achieves remarkable inference speeds, making it the premier choice for real-time applications [15].

In the context of edge computing, the “Nano” (n) variants of the YOLO series represent a specialized category of models designed with a minimal parameter count. These models are engineered to operate within the stringent memory and processing constraints of embedded hardware. Recent advancements in model compression and architectural scaling have led to the development of YOLOv26n, the latest iteration by Ultralytics [16]. This version introduces optimized neural layers that are specifically tuned for high-performance execution on modern processors. The YOLOv26n variant, in particular, is designed to deliver high-accuracy detection with a significantly reduced computational footprint, making it ideal for the limited resource environment of the Raspberry Pi 5.

The selection of YOLOv26n for this study is driven by its ability to balance the complex requirements of motorcycle detection in dense urban settings with the need for high-frequency inference [17]. In Yogyakarta’s traffic environment, where objects are frequently occluded and move at varying speeds, the model’s advanced feature extraction capabilities are crucial. By leveraging the updated architectural blocks of YOLOv26, this study aims to explore how effectively this cutting-edge model can be translated through various inference frameworks to achieve optimal real-time monitoring results.

C. Edge Computing and Raspberry Pi 5

The traditional architecture for large-scale video analytics has long relied on cloud computing, where data is transmitted to centralized servers for processing. However, the proliferation of high-definition surveillance streams in modern Smart Cities has exposed significant limitations in this model, including high latency, substantial bandwidth costs, and potential data privacy risks. To address these challenges, the paradigm of Edge Computing has gained prominence [18]. Edge computing decentralizes data processing by moving the computational workload closer to the data source in this case, the CCTV camera. By performing real-time inference on local edge nodes, the system can provide immediate response times and significantly reduce the volume of data transmitted over the metropolitan network, which

is vital for the time-sensitive requirements of traffic management.

The Raspberry Pi 5 serves as a high-performance, cost-effective platform for implementing edge-based AI solutions. Featuring the Broadcom BCM2712 SoC with a quad-core ARM Cortex-A76 processor clocked at 2.4 GHz, the Raspberry Pi 5 offers a substantial performance leap over its predecessors, specifically in handling the complex mathematical operations required by deep learning models [19]. The Cortex-A76 architecture includes advanced ARM NEON instructions, which provide Single Instruction Multiple Data (SIMD) capabilities that are essential for accelerating neural network inference. Despite these hardware advancements, the Raspberry Pi 5 remains a resource-constrained environment compared to discrete desktop GPUs. Consequently, the efficiency of the software inference engine becomes the deciding factor in maintaining a high frame rate while operating within the device’s thermal and power envelopes.

Thermal management remains a critical consideration when deploying AI models on the Raspberry Pi 5 for continuous operations. As the CPU handles high-frequency inference tasks, heat dissipation becomes a bottleneck that can lead to thermal throttling a protective mechanism where the processor reduces its clock speed to prevent damage. Throttling significantly degrades the inference throughput, leading to inconsistent detection performance [20]. Therefore, academic research in this domain emphasizes not only the accuracy of the detection model but also the optimization of the inference framework to ensure the hardware can sustain peak performance during prolonged surveillance cycles in tropical urban environments like Yogyakarta.

D. Deep Learning Inference Frameworks (ONNX vs. NCNN)

The efficiency of a deep learning model on edge hardware is largely determined by the inference framework used to execute the neural network’s computational graph. While frameworks like PyTorch and TensorFlow are excellent for model training, they are often too resource-heavy for deployment on embedded systems. Consequently, specialized runtimes have been developed to bridge the gap between high-level model design and low-level hardware execution. Among these, the Open Neural Network Exchange (ONNX) Runtime and NCNN represent two different philosophies of model deployment. ONNX Runtime, developed by Microsoft [21], serves as a universal cross-

platform engine designed for interoperability. It allows models from various training libraries to be converted into a standardized .onnx format, providing a robust and flexible baseline for inference across diverse CPU and GPU architectures.

In contrast, NCNN is a high-performance neural network inference framework specifically engineered for mobile and embedded platforms. Developed by Tencent [22], NCNN is notable for being “zero-dependency,” meaning it does not rely on third-party libraries, making it exceptionally lightweight and suitable for devices with limited storage and memory. A key distinction of NCNN is its “bottom-up” optimization approach for ARM-based processors. It leverages manual assembly-level optimizations using ARM NEON instructions to maximize the throughput of matrix multiplications and convolutions. By bypassing the overhead often found in general-purpose runtimes, NCNN is theorized to provide superior performance on processors like the Raspberry Pi 5’s Cortex-A76, particularly in scenarios requiring low-latency real-time detection.

Despite the widespread use of both frameworks, existing literature often benchmarks them on flagship mobile chipsets or older Raspberry Pi iterations. There is a notable research gap regarding their comparative performance on the Raspberry Pi 5 when handling high-resolution HLS streams from urban surveillance infrastructures. This study addresses this gap by providing an empirical evaluation of how these frameworks handle the latest YOLOv26n model, specifically analyzing whether NCNN’s hardware-specific optimizations offer a significant enough advantage over the universal ONNX standard to justify its use in Yogyakarta’s Smart City surveillance systems.



Figure 1. Samples of annotated images in the MS COCO dataset [23].

III. METHOD

The methodology of this research is structured into four distinct phases: model deployment preparation, hardware environment setup, real-time data acquisition from CCTV streams, and performance evaluation. The overall system architecture is designed to perform on a Raspberry Pi 5.

TABLE I
Research Component Specifications

No	Component	Specifications
1	Hardware	Raspberry Pi 5 (8GB RAM)
2	Processor	Broadcom BCM2712 (ARM Cortex-A76 @ 2.4GHz)
3	Operating System	Raspberry Pi OS 64-bit (Debian Bookworm)
4	AI Model	YOLOv26n (Nano)
5	Inference Engine 1	ONNX Runtime (CPU Execution Provider)
6	Inference Engine 2	NCNN (ARM NEON Optimized)

A. Model Deployment Preparation

The object detection architecture employed in this study is the pretrained YOLOv26n model provided by Ultralytics [22]. This model was out-of-the-box optimized by training on the baseline Microsoft Common Objects in Context (MS COCO) dataset, which encompasses 80 generic object classes across over 200,000 annotated real-world images [23]. For the purpose of this traffic surveillance evaluation in Yogyakarta, the framework specifically filters and extracts the motorcycle class (COCO class index 3). Utilizing a pretrained model ensures that the feature extraction layers are robust against standard atmospheric variations, spatial occlusions, and diverse lighting conditions inherent in urban public CCTV feeds. During the inference phase on the Raspberry Pi 5, the model execution parameters were standardized with an input image frame size of 640 x 640 pixels and a baseline confidence threshold set at 0.25.

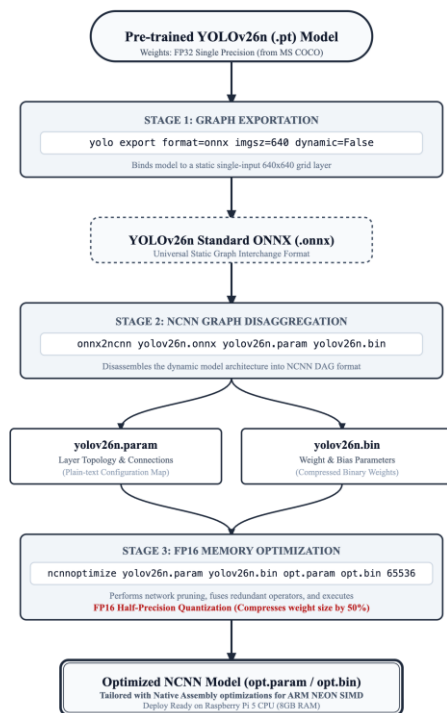


Figure 2. Flowchart of the YOLOv26n model optimization and runtime conversion pipeline.

To implement the YOLOv26n architecture on the edge hardware platform, the native PyTorch checkpoint file (.pt) must undergo a structural conversion pipeline. High-level deep learning frameworks introduce substantial software dependency overheads that severely bottleneck runtime execution on embedded CPUs. Therefore, this study compiles and optimizes the computational graph into two target deployment runtimes: ONNX Runtime and NCNN. The granular engineering workflow of this transition is structured into three consecutive phases, as mapped in Figure 2.

B. Hardware and Software Environment

The empirical engineering workflows, model compilation pipelines, and framework benchmarking evaluations conducted in this study were strictly executed within the Computer Laboratory at Universitas Islam Negeri (UIN) Sayyid Ali Rahmatullah Tulungagung. This specialized academic computing facility provided the localized physical infrastructure, regulated ambient testing temperatures (24°C to 26°C via climate-controlled systems), and high-speed, low-latency broadband network connection necessary to maintain an uninterrupted network handshake with the remote Yogyakarta Smart City public surveillance platform.

The experiments themselves were conducted on a Raspberry Pi 5 single-board

computer equipped with an ARM Cortex-A76 quad-core processor and 8GB of LPDDR5 RAM. To maintain physical consistency and prevent severe thermal throttling from affecting the reliability of the comparative results, an active cooling system was continuously utilized. During the deployment testing phase, telemetry parameters including frame processing latency (T_{inf_total}), throughput (FPS), CPU core temperature, thermal profile, and memory footprint and were continuously monitored and logged at a granular millisecond-level sampling rate for each frame over a rigorous, uninterrupted data capture duration of exactly 194 seconds. By bounding the experimental environment within this laboratory under structured cooling, external network jitter and erratic ambient thermal anomalies were minimized. This environmental control guarantees that the variations in computational throughput, core inference latency T_{core_pre} , and CPU thermal profiling between the ONNX Runtime and NCNN engines were strictly a function of low-level software kernel optimizations rather than environmental inconsistencies.

The software stack deployed on the edge node included the official Raspberry Pi OS (64-bit), Python 3.13, ONNX Runtime (CPU Execution Provider) as the cross-platform deployment baseline, and the ncnn-python library optimized for low-level hardware execution.

C. Real-time Stream Acquisition

The input data was sourced from the Yogyakarta City Government's public surveillance system, specifically at Live video feeds were accessed using the HTTP Live Streaming (HLS) protocol with the .m3u8 extension.

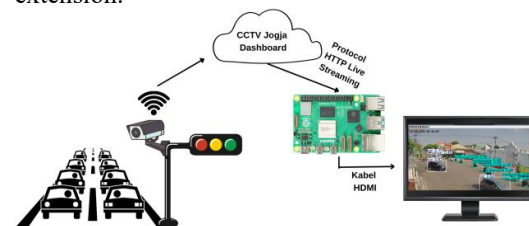


Figure 3. Data Acquisition Process

A multithreaded approach was implemented in the Python environment to decouple the frame acquisition process from the inference process, ensuring that network latency did not interfere with the computation of hardware performance metrics.

D. Evaluation Metrics

To provide a comprehensive comparison between ONNX Runtime and NCNN, the following quantitative metrics were recorded during the real-time detection of motorcycles:

- Inference Latency: The time taken by the CPU to process a single frame, measured in milliseconds (ms).
- Frames Per Second (FPS): The overall system throughput calculated as:

$$FPS = \frac{N_{frames}}{\sum_{i=1}^{N_{frames}} T_{inf_total}(i)} \quad (1)$$

Where the cumulative operational latency per individual video frame i , designated as T_{inf_total} , is composed of three consecutive pipelines formulated as:

$$T_{inf_total} = T_{pre} + T_{core_pre} + T_{post} \quad (2)$$

variable definitions:

- FPS : the average throughput performance of the localized edge system.
- N_{frames} : The total number of synchronized video frames processed during the structural evaluation window.
- T_{pre} : The structural latency consumed by image matrix formatting operations, which encompass raw hls network stream.
- T_{core_pre} : The core execution time utilized exclusively by the runtime deployment engine.
- T_{post} : The processing duration allocated for bounding box localization algorithms, which integrate confidence score thresholding ($conf \geq 0.25$)
- CPU Utilization: The percentage of CPU resources consumed during the inference loop.
- Memory Footprint: The total RAM consumption (MB) required to load the model and execute the inference engine.
- Thermal Profile: The CPU temperature recorded in degrees Celsius (°C) to monitor the heat dissipation efficiency of each framework.

IV. RESULTS AND DISCUSSION

A. Results



Figure 4. ONNX Runtime inference Engine

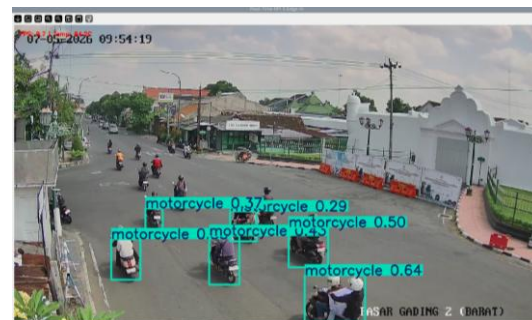


Figure 5. NCNN inference Engine

The visual evidence presented in Figure 4 and Figure 5 provides empirical proof that both ONNX Runtime and NCNN frameworks are capable of implementing the YOLOv26n model for real-time motorcycle detection. Through the processing of live CCTV data streams in Yogyakarta, both frameworks successfully generated accurate bounding boxes and classification labels for every motorcycle captured under dynamic traffic conditions. This achievement demonstrates that the model conversion from its native format to ONNX and NCNN preserves the integrity of its spatial features, ensuring that detection reliability remains robust even when deployed on resource-constrained hardware like the Raspberry Pi 5.

The experimental results demonstrate a significant performance disparity between the ONNX Runtime and NCNN frameworks when deploying the YOLOv26n model on a Raspberry Pi 5. To ensure a fair comparison, both frameworks were evaluated over a synchronized duration of 194 seconds using live *.m3u8* streams from Yogyakarta's CCTV infrastructure.

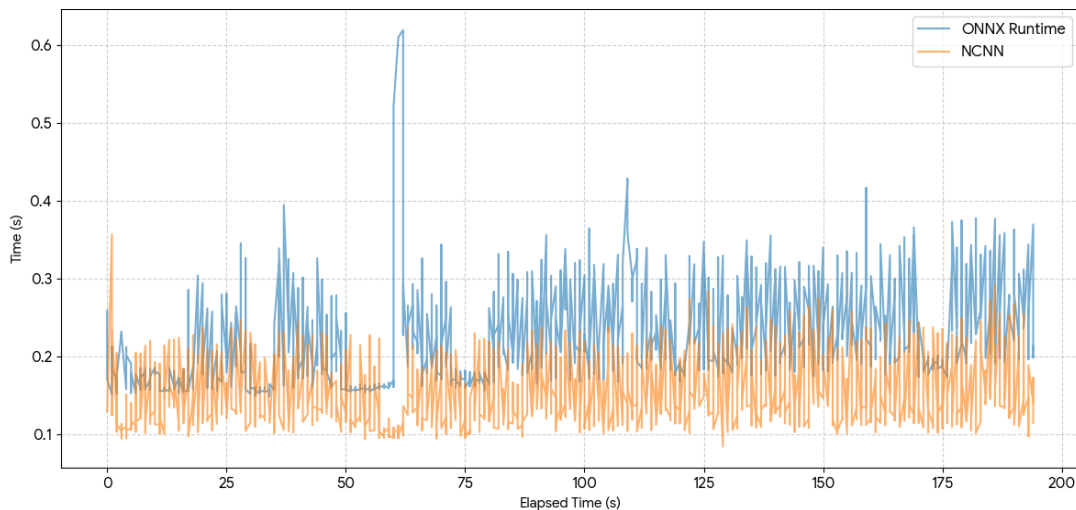


Figure 6. Comparative Inference Latency

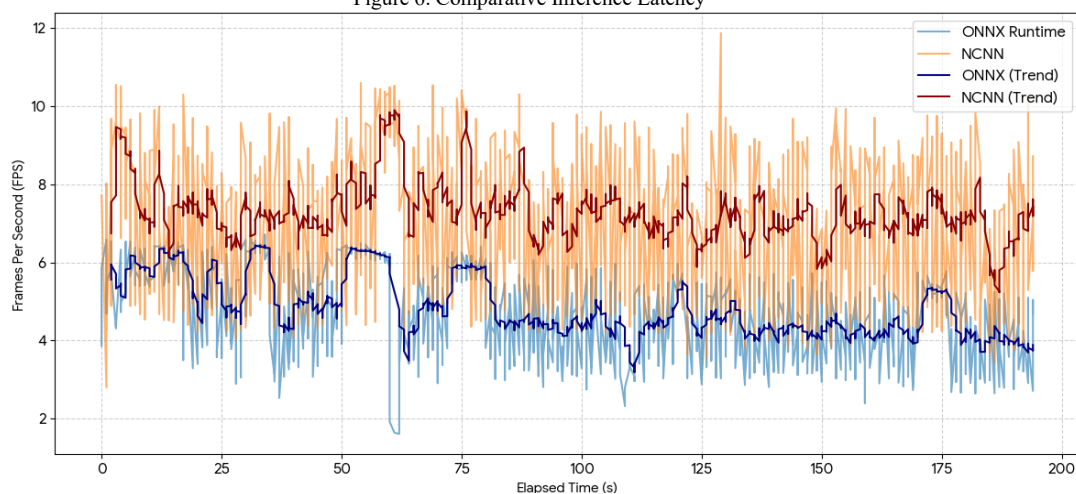


Figure 7. Real-time System Throughput (FPS)

TABLE II
Summary Comparison Table

Evaluation Metrics	ONNX Runtime	NCNN	Difference / Efficiency
Average Inference Time (s)	0.216	0.145	32.9% (Faster)
Average System FPS	4.92	7.36	49.4% (Higher)
Average CPU Usage (%)	95.83%	77.61%	18.2% (Lower Load)
Average RAM Usage (MB)	1543.8	1500.1	43.7 MB (Lower)
Average Temperature (°C)	80.5	78.7	1.8 °C (Cooler)
Peak Temperature (°C)	86.5	85.3	1.2 °C (Lower)

During the observation period, the NCNN framework and ONNX runtime successfully processed a total of 842 frames. In terms of temporal efficiency, NCNN achieved an average

inference latency of 0.145 seconds per frame, which is approximately 32.9% faster than ONNX Runtime's average of 0.216 seconds. This improvement in latency directly corresponds to the system throughput, where NCNN reached an average of 7.36 FPS compared to 4.92 FPS for ONNX Runtime.

From a hardware resource perspective, NCNN exhibited superior efficiency despite its higher processing speed. The average CPU utilization for NCNN was recorded at 77.61%, significantly lower than the 95.83% observed in ONNX Runtime. Furthermore, NCNN maintained a slightly more favorable thermal profile, with an average temperature of 78.7°C, compared to 80.5°C for ONNX Runtime. The peak temperature reached during the experiment was 86.5°C for ONNX Runtime and 85.3°C for

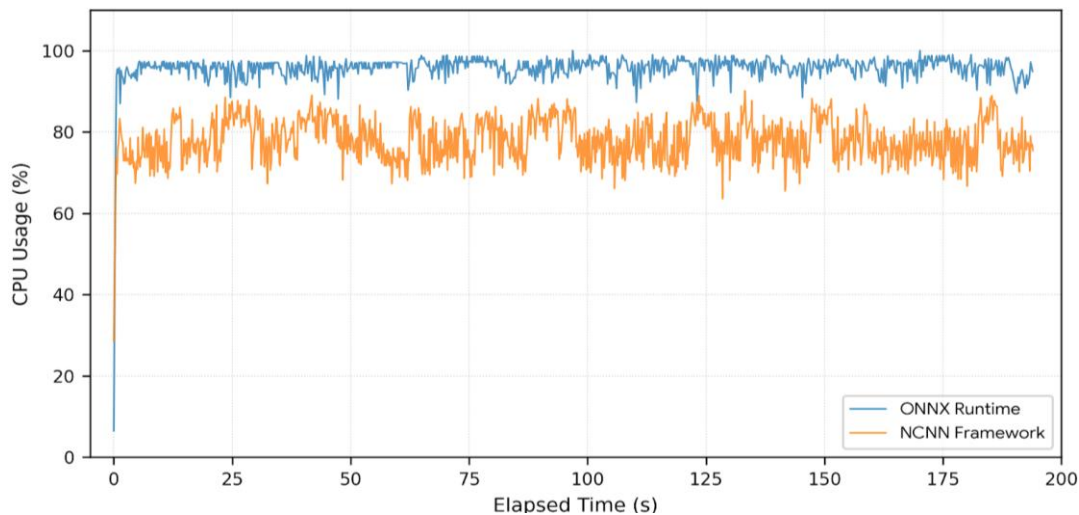


Figure 8. CPU usage comparison over time

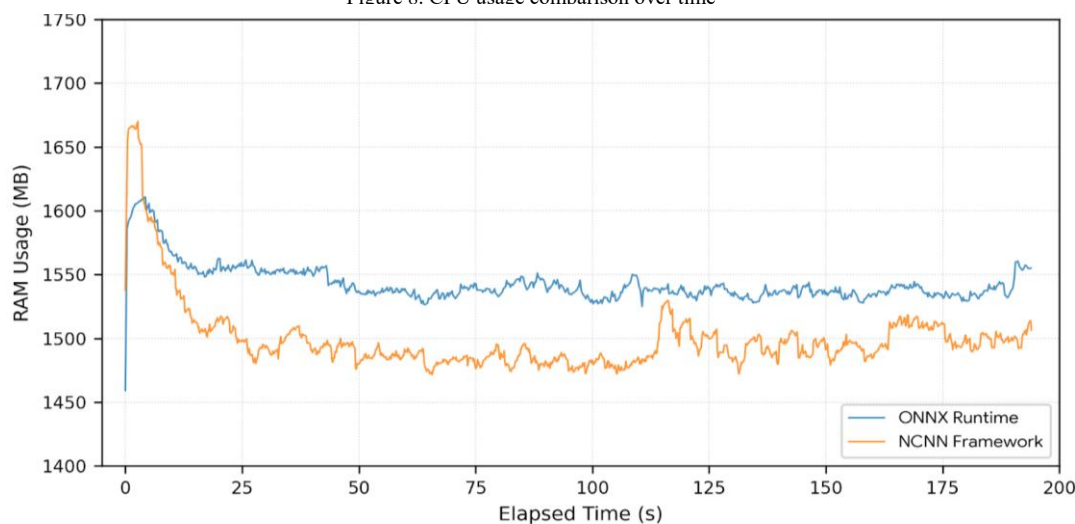


Figure 9. RAM usage comparison over time

NCNN, both operating near the thermal throttling threshold of the Raspberry Pi 5.

The comparative analysis of inference latency, as illustrated in Figure 6 the Inference Time graph, reveals a distinct advantage for the NCNN framework. NCNN maintains a significantly lower and more consistent latency throughout the 194-second test period, averaging approximately 0.145 seconds per frame. In contrast, ONNX Runtime exhibits a higher average latency of 0.216 seconds with more frequent and pronounced “spikes”. These fluctuations in ONNX Runtime suggest a higher computational overhead when processing complex frames with multiple detected motorcycles, whereas NCNN’s optimized kernels allow for more stable execution.

The System FPS graph further underscores the efficiency of NCNN in a real-time surveillance context, as illustrated in Figure 7. NCNN consistently achieves a higher throughput, with a trend line stabilizing around 7.36 FPS.

Conversely, ONNX Runtime struggles to maintain a high frame rate, averaging only 4.92 FPS. The wider gap between the two trend lines demonstrates that NCNN provides a nearly 50% increase in processing speed. This higher FPS is critical for Yogyakarta’s urban monitoring, as it ensures smoother tracking of fast-moving motorcycles and reduces the risk of temporal data loss during high-density traffic intervals.

As plotted in Figure 8, the frame-by-frame telemetry captured at a high-frequency millisecond-level sampling rate reveals a stark divergence in CPU utilization profiles between the two deployment providers. Throughout the continuous 194-second ingestion of the Pasar Gading CCTV live stream, ONNX Runtime subjects the Broadcom BCM2712 SoC to maximum processing strain, maintaining an extreme execution plateau with an average CPU load of 95.83% and frequently capping at its absolute physical limit of 100.00%. Conversely, the NCNN framework demonstrates superior

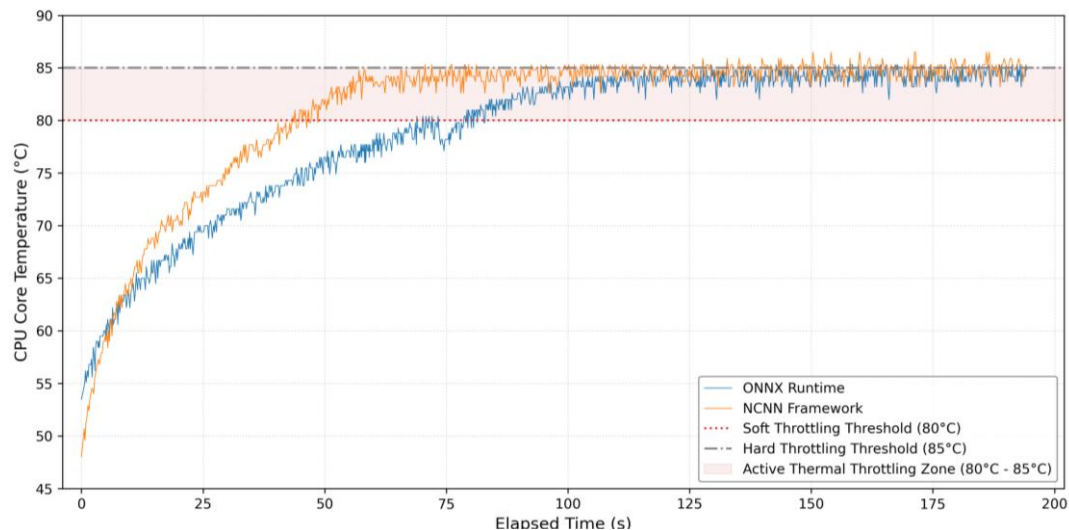


Figure 10. CPU core temperature comparison over time

low-level architectural efficiency, relaxing the processor core strain down to a stable mean of 77.61% with transient structural spikes topping out at only 90.10%. This optimization margin represents an average 18.22% absolute reduction in CPU load compared to ONNX Runtime, confirming that NCNN's low-level integration of hand-written ARM NEON SIMD assembly instructions effectively parallelizes tensor matrix multiplications across the Cortex-A76 registers without trapping the hardware in persistent execution loops.

The physical memory allocation tracked in Megabytes (MB) over the exact same data capture timeline demonstrates a clear contrast in runtime memory management strategies as plotted in Figure 9. Upon system initialization during the first 15 seconds, both frameworks experience a sharp, synchronized memory allocation surge as the serialized YOLOv26n computational graph tensors are deserialized and loaded into the LPDDR5 RAM, where NCNN registers a transient peak of 1,670.00 MB and ONNX Runtime reaches 1,610.60 MB. However, once the surveillance node enters a steady execution state from second 20 onwards, ONNX Runtime maintains a consistently elevated memory baseline floating at a mean of 1,544.46 MB due to continuous node caching within its generic runtime engine layer. In contrast, the zero-dependency NCNN framework successfully flushes temporary layer allocations from the heap, streamlining its steady-state memory footprint down to a highly optimized baseline with a mean of 1,500.13 MB, thereby maintaining a lighter active memory footprint by approximately 44.33 MB during real-time traffic tracking.

The CPU Temperature graph depicts the thermal impact of continuous AI inference on the Raspberry Pi 5 as plotted in Figure 10. Both frameworks show an upward thermal trajectory, eventually operating within the 80°C to 85°C range. However, NCNN exhibits a more efficient thermal profile, maintaining an average temperature that is 1.8°C cooler than ONNX Runtime. Despite NCNN processing 36% more frames in the same duration, it generates less heat per processed frame. This thermal efficiency is a direct result of lower CPU utilization (77.61% for NCNN vs. 95.83% for ONNX), which is vital for preventing the device from entering a thermal throttling state that would otherwise degrade detection accuracy and system longevity.

B. Discussion

In order to structurally validate the engineering performance boundaries mapped out in this study, the empirical resource utilization profiles of the out-of-the-box YOLOv26n architecture were cross-examined with the latest benchmark data from previous research [24]. In their comprehensive evaluation of 31 YOLO architectures for UAV-based thermal object detection, a key architectural pattern emerged: while heavier variants scale up spatial box localization capabilities, smaller, edge-first variants provide a crucial baseline for computational efficiency. Those research established that YOLOv26n demonstrates exceptionally stable optimization behavior, with its validation curves closely tracking heavier cross-version models without compromising convergence stability.

However, a critical bottleneck arises during localized physical deployment. In the edge-

device benchmarking layer conducted, model exports were evaluated using the standard NCNN framework on a legacy Raspberry Pi 4B hardware platform. Under those hardware constraints, compact architectures like YOLOv8n and YOLOv11n topped out at a processing throughput constraint of only 2.0 to 2.7 FPS, which remains inadequate for real-time traffic monitoring systems requiring instantaneous frame analysis.

By scaling the hardware environment to the modern Raspberry Pi 5 platform in this research, the processing floor significantly advances, yet the software deployment framework chosen dictates the final operational stability. Under a continuous live data ingestion from the Pasar Gading CCTV stream captured over 194 seconds at a granular, frame-by-frame millisecond sampling rate, our native PyTorch execution provider yields 4.11 FPS, which slightly increases to 4.92 FPS via ONNX Runtime.

As demonstrated by the dense, high-frequency signal trajectories shown in the results section, ONNX Runtime induces severe thread contention within the CPU Execution Provider layer on the Broadcom BCM2712 SoC. It forces a sustained CPU utilization plateau at an extreme mean of 95.83% (frequently capping at its absolute limit of 100.00%) and drives a heavy physical memory cache footprint with a mean of 1,544.46 MB. This persistent processor stress explains the rapid thermal trajectory towards the dynamic soft thermal throttling boundary (80°C to 85°C).

Deploying the NCNN framework to execute the serialized YOLOv26n graph successfully breaks this architectural execution bottleneck. NCNN accelerates the processing throughput to 7.36 FPS, outperforming the baseline Raspberry Pi 4B performance boundaries identified in the literature by a substantial margin. More importantly, the raw per-frame tracking logs mapped in this study demonstrate that NCNN actively minimizes the hardware stress on the ARM Cortex-A76 cores. NCNN relaxes the average CPU utilization down to 77.61% (representing an 18.22% absolute reduction in processing strain compared to our ONNX baseline) and streamlines the active physical memory allocation down to a steady-state mean of 1,500.13 MB. This computational relief confirms that NCNN's low-level, hand-written ARM NEON SIMD assembly instructions effectively prevent the system from getting trapped in persistent cache overhead

loops.

These empirical results strongly align with the structural conclusions drawn [24], which asserted that heavyweight models introduce severe diminishing returns relative to their resource requirements on embedded hardware, whereas compact, optimized variants offer the most favorable engineering trade-off. Therefore, while the scope of this study remains strictly bounded to computational performance profiling without altering structural weights or validating spatial metrics, this multi-dimensional discussion confirms that optimizing pre-trained edge-first pipelines down to assembly-level ARM kernels via NCNN represents the most viable roadmap to deploy stable, high-throughput, and thermally safe real-time smart city nodes.

V. CONCLUSION

This research has presented a rigorous, hardware-aware computational benchmarking evaluation of the pre-trained YOLOv26n micro-architecture on the Raspberry Pi 5 single-board computer, mapping the operational limits of ONNX Runtime and the NCNN framework. By strictly bounding the research scope to physical telemetry and resource utilization tracking under a continuous 194-second public HLS stream from the Pasar Gading Traffic Light areas in Yogyakarta, a clear engineering trade-off regarding low-level compiler choices has been established. The empirical telemetry, captured at a high-frequency millisecond-level sampling rate, reveals that the standard ONNX Runtime engine induces substantial structural overhead on the quad-core ARM Cortex-A76 processor. ONNX Runtime forces a severe CPU consumption plateau at a mean of 95.83% and maintains a heavy memory footprint of 1,544.46 MB, which accelerates the SoC temperature directly into the hardware's dynamic thermal throttling boundary 80°C to 85°C.

In contrast, the zero-dependency NCNN framework successfully breaks this execution bottleneck. By utilizing optimized ARM NEON SIMD assembly instructions, NCNN parallelizes tensor matrix operations directly across the processor registers, accelerating the system throughput to 7.36 FPS. Crucially, this performance leap is achieved while actively reducing hardware stress, as NCNN lowers the average CPU load by an absolute margin of 18.22% (down to a stable mean of 77.61%) and refines the steady-state active memory allocation to a streamlined mean of 1,500.13 MB. These results prove that software-level architecture optimization is paramount for embedded systems, where upgrading hardware components yields

diminishing returns if constrained by wrapper redundancies. Consequently, this study concludes that deploying assembly-optimized runtimes like NCNN represents the most sustainable and thermally safe architectural framework to realize real-time, standalone edge traffic surveillance nodes within the infrastructure constraints of modern Indonesian smart cities. Future work may explore the implementation of multi-threaded frame parsing layers to further scale up processing frame rates under multi-channel streaming conditions.

REFERENCE

- [1] Handoko, M., Mubarak, H., Shaura, R. K., Lesmana, H., Widyastuti, R., Swastika, R., ... & Hartini, D. (2025). The Architecture of Intelligent Transportation System based on Sensor Monitoring (Implementation in Jakarta Area). *Journal of Technology Informatics and Engineering*, 4(2), 190-203.
- [2] Ebrahim, R. (2021). Development of a cost-effective method to implement traffic management principles to a small city environment (Doctoral dissertation, Stellenbosch: Stellenbosch University).
- [3] Cao, L. (2022). Decentralized ai: Edge intelligence and smart blockchain, metaverse, web3, and desc. *IEEE Intelligent Systems*, 37(3), 6-19.
- [4] Sun, Y., Ochiai, H., & Esaki, H. (2021). Decentralized deep learning for multi-access edge computing: A survey on communication efficiency and trustworthiness. *IEEE Transactions on Artificial Intelligence*, 3(6), 963-972.
- [5] Talati, D. (2021). Decentralized AI: The role of edge intelligence in next-gen computing. Available at SSRN 5201744.
- [6] Liu, H. I., Galindo, M., Xie, H., Wong, L. K., Shuai, H. H., Li, Y. H., & Cheng, W. H. (2024). Lightweight deep learning for resource-constrained environments: A survey. *ACM Computing Surveys*, 56(10), 1-42.
- [7] Shuvo, M. M. H., Islam, S. K., Cheng, J., & Morshed, B. I. (2022). Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, 111(1), 42-91.
- [8] Huaman-Pacco, I. P., Sacoto-Cabrera, E. J., Silva-Alvarado, V. L., Ahmad, A., Sendra, S., Lloret, J., & Moreno-Cardenas, E. (2026). Proposal for Computationally Efficient Fog Computing System for Coffee Berry Borer Detection via Optimized YOLOv26. *Sensors*, 26(7), 2212.
- [9] Shalash, O., Khatab, E., El-Agamy, A., Elmokadem, L., Abouelsaad, Y., Zaki, J., ... & Said, H. (2026). Benchmarking YOLOv8–YOLOv12 for Real-Time Object Detection on Single-Board Computers.
- [10] Winkowska, Justyna, Danuta Szpilko, and Sonja Pejić. "Smart city concept in the light of the literature review." *Engineering Management in Production and Services* 11.2 (2019).
- [11] M. Elassy, M. Al-Hattab, M. Takruri, and S. Badawi, "Intelligent transportation systems for sustainable smart cities," *Transportation Engineering*, p. 100252, June 2024, doi: 10.1016/j.treng.2024.100252.
- [12] Pemerintah Kota Yogyakarta, "CCTV Berbasis Kewilayahan | Pemerintah Kota Yogyakarta," CCTV Berbasis Kewilayahan | Pemerintah Kota Yogyakarta. <https://cctv.jogjakota.go.id> (accessed May 7, 2026).
- [13] Wahyu, Saputra V., N. Suciati, And Ch Fatichah. "Fog and rain augmentation for license plate recognition in tropical country environment." *IAES International Journal of Artificial Intelligence* 13.4 (2024): 3951.
- [14] Deniz, Mehmet, Ismail Bogrekc, and Pinar Demircioglu. "Real-Time Detection of Hole-Type Defects on Industrial Components Using Raspberry Pi 5." *Applied System Innovation* 8.4 (2025): 89.
- [15] Caglar Gencosman, Burcu. "Comprehensive Evaluation of YOLOv8 to YOLOv26 for Real-Time Weapon Detection in CCTV Surveillance." *Comprehensive Evaluation of YOLOv8 to YOLOv26 for Real-Time Weapon Detection in CCTV Surveillance*.
- [16] Murtaqib, Muhammad Ammar, et al. "Performance Analysis of Vehicle Counting and Classification Using YOLO at a Four-Way Intersection Based on Indonesian Road Capacity Manual." 2024 International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA). IEEE, 2024.
- [17] Aishwarya, D., and R. I. Minu. "Edge computing based surveillance framework for real time activity recognition." *ICT Express* 7.2 (2021): 182-186.
- [18] Raspberry Ltd, "Raspberry pi 5," Raspberry Pi, Apr. 2026. <https://pip-assets.raspberrypi.com/categories/892-raspberry-pi-5/documents/RP-008348-DS-6-raspberry-pi-5-product->

- brief.pdf?disposition=inline (accessed May 7, 2026)
- [19] Minott, D., Siddiqui, S., & Haddad, R. J. (2025, March). Benchmarking edge AI platforms: performance analysis of Nvidia Jetson and raspberry pi 5 with coral TPU. In *SoutheastCon 2025* (pp. 1384-1389). IEEE.
- [20] Microsoft, "GitHub - microsoft/onnxruntime: ONNX Runtime: cross-platform, high performance ML inferencing and training accelerator · GitHub," GitHub. <https://github.com/microsoft/onnxruntime> (accessed May 08, 2026).
- [21] Tencent, "GitHub - Tencent/mxnet: mxnet is a high-performance neural network inference framework optimized for the mobile platform · GitHub," GitHub. <https://github.com/Tencent/mxnet> (accessed May 07, 2026).
- [22] 'G. Jocher and J. Qiu, "Ultralytics YOLO26," version 26.0.0, 2026. [Online]. Available: <https://github.com/ultralytics/ultralytics>. [Accessed: May 7, 2026].
- [23] T.-Y. Lin et al., "Microsoft COCO: Common Objects in Context," in *Lecture Notes in Computer Science*, Springer International Publishing, 2014, pp. 740–755.
- [24] Safonova, A., Campanera-Moliné, A., Silva, J. P., Sarró, D. F. N., Guerra, M. C., Serra-Sagristà, J., & Santangeli, A. (2026). Deep learning for UAV thermal bird detection: Benchmarking YOLOv8–YOLOv26. *Remote Sensing Applications: Society and Environment*, 101981.