JURNAL TEKNOLOGI DAN OPEN SOURCE
Vol. 9, No. 1, June 2026, pp. 335~345
e-ISSN: 2622-1659, accredited Four Grade by Kemenristekdikti, Decree No: 152/E/KPT/2023
DOI: https://doi.org/10.36378/jtos.v9i1.5723

Development of a Vehicle Detection and Classification System Using YOLO and Real-Time API With The Rapid Application Development (RAD) Method

Rizqo Sahala Putra[1], Muhammad Asep Subandri[2]
[1,2]Department of Computer Engineering, Bengkalis State Polytechnic, Indonesia

Article Info

Article history:

Received 05 01, 2026
Revised 06 10, 2026
Accepted 06 24, 2026

Keywords:

YOLOv8
Vehicle Detection
Rule-Based Post-Processing
ByteTrack
CCTV
Computer Vision

ABSTRACT

This study developed a real-time, computer vision-based traffic monitoring system to address safety concerns regarding uncontrolled public vehicle access within the Politeknik Negeri Bengkalis campus environment. As the existing CCTV infrastructure lacked intelligent analytical capabilities, an automated solution was required to extract objective traffic data. The system utilized the YOLOv8n model for vehicle detection and the ByteTrack algorithm for multi-object tracking to prevent duplicate counting across frames. To overcome classification inaccuracies inherent in pre-trained models without relying on resource-intensive retraining, a rule-based post-processing method was implemented. This method evaluated bounding box geometries, including pixel area, aspect ratio, and a vertical camera perspective factor, to filter raw detections. Evaluation results demonstrated that the rule-based approach significantly minimized false positives, achieving a total precision of 93.48% and an overall accuracy of 86.00%, which vastly outperformed both baseline and fine-tuned configurations. Furthermore, comparative tests across model variants confirmed that YOLOv8n provided the most stable CPU execution, maintaining the highest frame rate and lowest inference latency. The integrated system successfully tracked vehicle counts and estimated relative speeds, providing a reliable, GPU-independent analytical tool to support data-driven campus traffic management policies.

Corresponding Author:

Rizqo Sahala Putra
Department of Informatics Engineering
Politeknik Negeri Bengkalis
Bengkalis, Indonesia
Email: rizqosp26@gmail.com

1. Introduction

Efficient and safe traffic management is a crucial necessity in high-mobility environments, such as university campuses. At Politeknik Negeri Bengkalis, public access roads crossing the campus remain open to the public due to its geographical layout, creating potential hazards for students, lecturers, and staff who must navigate through uncontrolled general traffic. The discourse on closing or restricting road access has been considered by the campus planning department as an option to improve safety. However, implementing

such policies requires objective supporting data, such as real-time vehicle volumes, types, and speeds. Unfortunately, the existing monitoring system still relies on conventional CCTV cameras that lack intelligent analysis capabilities, meaning structured data cannot be directly extracted to inform decision-making.

To address these challenges, an automated system based on computer vision is needed. The Convolutional Neural Network (CNN) architecture, popularized by the landmark AlexNet model, has proven highly effective in large-scale image classification tasks [1]. It has been successfully applied to design vehicle detection and classification systems, achieving an accuracy of 91.4% [2]. However, for real-time monitoring needs, the You Only Look Once (YOLO) algorithm is considered superior due to its ability to perform object localization and classification simultaneously. Various versions of YOLO have been extensively researched; for instance, the YOLOv5 model was capable of detecting and classifying vehicle types in Indonesia with 90% accuracy [3], while a system utilizing the YOLOv8 algorithm for real-time vehicle counting achieved an accuracy reaching 97.5% [4].

Further research has explored various combinations and iterations of YOLO to tackle specific traffic monitoring scenarios. For example, Kurniawan et al. combined the YOLACT framework with the k-nearest neighbor (KNN) method to classify vehicle types and colors in real-time, achieving 91.67% accuracy specifically for small or distant objects [5]. In more confined environments, YOLOv1 has been effectively implemented to monitor parking areas, accurately identifying four types of vehicles with a 98.66% success rate [6]. Other studies focused on granular classification, such as utilizing YOLO to detect specific car brands and models with 92% accuracy [7], or combining CNN and YOLO to advance object detection capabilities on public highways [8]. Subsequent iterations continue to show strong performance in varying conditions; YOLOv7 achieved 86% accuracy in classifying highway vehicles [9], while YOLOv8 has been utilized specifically to automate the calculation of passing vehicle volumes [10]. Additionally, YOLOv4 has been successfully simulated for detecting and counting vehicles at intersections via CCTV, maintaining a high real-time accuracy of 94.3% [11].

Recent advancements in object detection have evolved from the original YOLO framework introduced by Redmon et al. [12], which enabled real-time object detection through a single-stage architecture. Subsequent developments, including YOLO9000 [13], YOLOv3 [14], YOLOv4 [15], and YOLOX [16], have significantly improved detection accuracy, localization capability, and computational efficiency. These improvements have encouraged the adoption of YOLO-based systems in intelligent transportation applications, particularly for vehicle detection and traffic monitoring in dynamic environments.

In addition to object detection, reliable vehicle counting requires robust multi-object tracking techniques. Traditional tracking approaches such as SORT [18] and DeepSORT [19] have demonstrated strong performance in maintaining object identities across sequential frames. Recent studies have also highlighted the importance of vehicle speed estimation and small object detection in intelligent transportation systems. Accurate speed estimation utilizing deep learning approaches enables automated traffic evaluation under various conditions [21], while comprehensive reviews demonstrate that robust vehicle detection systems are critical for real-time distance estimation [22]. Furthermore, recent advancements highlight that integrating deep learning with edge computing devices significantly empowers automated traffic monitoring platforms to support data-driven municipal policies [26].

Although YOLO technology has been widely utilized with high accuracy rates, the majority of previous studies have focused on open highways, parking areas, or intersections. Few studies have implemented an end-to-end system specifically designed to monitor traffic within a campus environment to support the evaluation of road access restriction policies. Furthermore, applying deep learning models in real-world CCTV conditions still faces several challenges, such as misclassification between vehicle types with similar visual features (e.g., pickup trucks and regular trucks), camera perspective variations, and calculation duplication caused by a single vehicle being detected across multiple frames.

Therefore, this study aims to develop a YOLOv8-based vehicle detection, classification, and counting system optimized for real-world traffic conditions. The system is designed to address challenges such as varying lighting conditions, vehicle occlusion, and fluctuating traffic density while maintaining high detection accuracy and efficiency. The novelty of this research lies in the integration of a pre-trained YOLOv8 model with a rule-based post-processing approach that utilizes bounding box geometric features, including object area, aspect ratio, and spatial position, to improve vehicle classification accuracy without requiring model retraining. This approach enhances classification performance while reducing computational complexity and development costs. In addition, the system incorporates the ByteTrack algorithm for multi-object tracking, enabling consistent object identification and ensuring that each vehicle is counted only once through a stable track ID mechanism. The system is developed using the Rapid Application Development (RAD) method and integrated with a real-time API, allowing traffic data to be visualized through a web dashboard to support campus traffic management and decision-making..

2. Research Method

2.1. Research Procedure

This study implements the Rapid Application Development (RAD) method, which prioritizes fast, adaptive, and iterative prototype development. The research workflow is systematically divided into four core structural phases: requirements planning, system design, development, and implementation. During the requirements planning phase, an in-depth analysis was conducted through field observations and direct interviews with campus planning authorities to specify the practical operational needs of a university-scale traffic monitoring framework. The system design phase focused on modeling a modular software architecture that seamlessly links intelligent computer vision processing with real-time user interfaces. Subsequently, the system development phase involved coding the fundamental object detection algorithms, multi-object tracking mechanisms, relative speed estimation heuristics, and asynchronous API data pipelines. Finally, the implementation phase verified the comprehensive functional performance of the integrated software using real-world video testing streams and live CCTV feeds across the campus network.

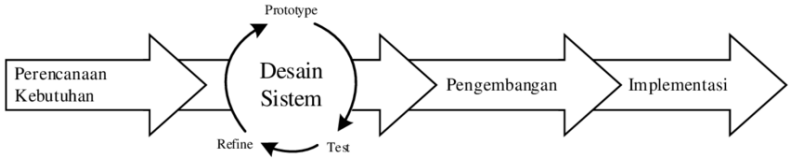

Figure 1. Stages of the RAD Method

2.2. Data Acquisition and Research Tools

To ensure rigorous validation and benchmarking of the traffic monitoring platform, data collection utilizes three primary pipelines:

1. *COCO Pre-trained Dataset*: This large-scale public dataset serves as the foundational database integrated into the baseline YOLOv8n model, enabling the instant recognition of standard vehicle classes including cars, motorcycles, buses, trucks, and bicycles without requiring a resource-intensive retraining process from scratch.
2. *Local Dataset*: A specific localized dataset comprising approximately 4,000 annotated images was compiled using mobile device cameras at Politeknik Negeri Bengkalis alongside specific public domain traffic repositories from Roboflow. Rather than using this data for model training, it is strictly deployed as an independent validation and testing benchmark to evaluate model reliability under localized environmental factors.
3. *Field Observation Data*: Direct operational parameters were gathered through manual field surveys to assess optical camera vantage points, analyze traffic density variations, and establish verified visual counts to act as the evaluation ground truth.

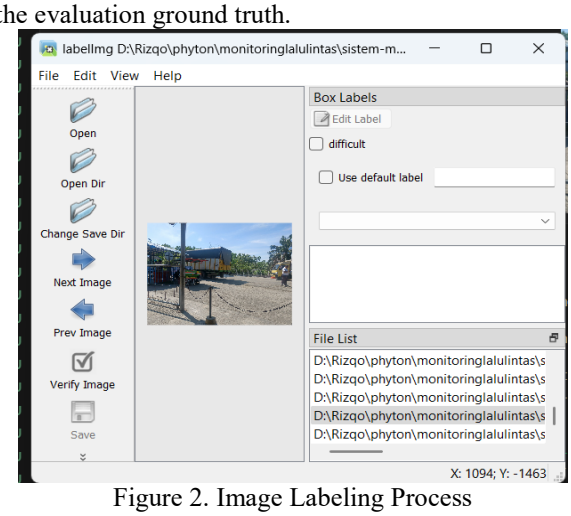

Figure 2. Image Labeling Process

The computational experimentation and end-to-end framework execution were carried out on a hardware setup featuring an Intel Core i5-1334U processor, Intel Iris Xe Graphics, 16 GB of RAM, a 512 GB solid-state drive (SSD), and an external high-definition CCTV/webcam infrastructure. On the software side, the entire system environment was programmed within Visual Studio Code on a Windows 11 operating system. The primary software libraries and backend infrastructure include the OpenCV library for advanced image manipulation and frame preprocessing, the Ultralytics YOLOv8n framework for high-speed object

detection, Laravel 11 for constructing the secure backend RESTful API architecture, Laragon to host the local Apache web server and structured MySQL database, and Draw.io for drafting the systemic structural diagrams.

Table 1. Comparison of YOLOv8 Model Variants

Parameter	YOLOv8n	YOLOv8s	YOLOv8m
Parameter Count	~3.2M	~11.2M	~25.9M
Model Size	~6MB	~22MB	~52MB
FPS (CPU)	17-31	8-16	2-7
Inference Speed	Very Fast	Moderate	Slow
Detection Accuracy	Good	Better	Best
Best For	Real-time CPU	Balanced	High Accuracy
Parameter Count	~3.2M	~11.2M	~25.9M

2.3. Research Procedure

The object detection component employed the Ultralytics YOLOv8 architecture, integrating an anchor-free detector approach—a paradigm shift shown to improve localization accuracy and computational efficiency in modern detectors like YOLOX [16].YOLOv8 adopts a streamlined detection head and optimized feature extraction backbone, making it particularly suitable for real-time deployment on CPU-constrained environments while maintaining competitive detection performance.

The selection of YOLOv8 was motivated by its anchor-free detection mechanism and efficient feature extraction capability, which provide competitive performance for real-time traffic monitoring applications. Previous studies have demonstrated that YOLO-based detectors consistently outperform traditional object detection approaches such as Faster R-CNN and SSD in terms of inference speed while maintaining high detection accuracy [23], [24], [25].

Real-time visual object detection is executed utilizing the lightweight YOLOv8n model architecture. Raw video input frames captured from the CCTV stream are programmatically resized to a uniform resolution of 640×640 pixels to fit the network input layer. The convolutional backbone extracts critical visual features such as object edges, textures, and geometric profiles, which are subsequently aggregated across multiple feature scales by the Path Aggregation Network (PANet) neck layer to safeguard detection sensitivity for both distant and nearby vehicles. The anchor-free detection head then yields the raw tensor predictions consisting of class probabilities, objectness scores, and coordinate bounding boxes simultaneously.

To circumvent the inherent classification missteps of the general pre-trained model and effectively drive down false positive counts, this research introduces a rule-based post-processing method. This architectural layer serves as an efficient, lightweight alternative to full model fine-tuning, which often triggers extreme overfitting when exposed to restricted, class-imbalanced local datasets. This method extracts the native spatial metrics of the raw bounding boxes specifically tracking the bounding box pixel area, the aspect ratio (W/H), and the vertical centroid position (center_y) relative to the complete frame height (frame_height). These logical rules integrate a dynamic camera perspective factor to algorithmically adjust and normalize the bounding box areas of vehicles that appear physically smaller near the upper horizon of the camera viewport. Furthermore, because the standard COCO dataset lacks an explicit class for pickup trucks, a class simplification logic is enforced to programmatically merge detected pickup profiles into the broader truck class based on shared light-cargo functional profiles, stabilizing classification performance over complex CCTV viewpoints.

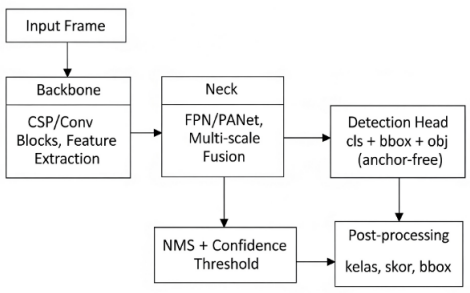

Figure 3. Detection and Post-Processing Approach Flow Diagram

2.4. Research Procedure

To maintain object identity consistency across video frames, this study utilized the ByteTrack multi-object tracking algorithm proposed by Zhang et al. [17]. ByteTrack associates both high-confidence and low-

confidence detections, allowing the tracker to recover temporarily lost objects and reduce identity fragmentation caused by occlusion, motion blur, and abrupt vehicle movement. This characteristic makes ByteTrack particularly effective for traffic monitoring scenarios involving dense and continuous vehicle flows.

Standard frame-by-frame object detection models are fundamentally limited by a lack of temporal awareness, resulting in severe vehicle counting duplications because a single vehicle is repeatedly identified as a new entity in every individual frame it traverses. To eliminate this flaw, the ByteTrack algorithm is integrated into the system core as the multi-object tracking engine. ByteTrack effectively maps bounding boxes across sequential frames by evaluating motion predictions and spatial overlap, assigning a permanent, distinct Track ID to every unique vehicle entering the scene. To bolster system resilience against temporary occlusion or brief detection dropouts, a strict minimum-frame activation threshold is applied, meaning an entity is only logged into the database after maintaining a stable tracking history across multiple consecutive frames.

Once a vehicle's unique identity is securely stabilized by the multi-object tracker, the system estimates its velocity through a heuristic centroid tracking approach across temporal frames relative to the active processing frame rate (FPS). Because this specialized deployment functions without complex physical camera lens calibration or real-world pixel-to-meter coordinate transformation mapping, the computed values denote a relative velocity index rather than a legally binding absolute speed measurement. The underlying algorithm combines a fixed vehicle-type base velocity constant with a variable box area scaling factor and a vertical frame perspective coefficient. These speed metrics are designed to operate purely as statistical indicators to map traffic flow velocity trends and spot potentially dangerous speeding patterns within the campus perimeter, rather than serving as a precise instrument for automated traffic law enforcement.

3. Result and Discussion

3.1. System Overview

The research successfully implemented an end-to-end computer vision-based traffic monitoring system capable of executing real-time vehicle detection, classification, multi-object tracking, and relative speed estimation. The system architecture was strategically developed to run optimally on client devices without a dedicated Graphics Processing Unit (GPU), ensuring that all deep learning inference phases are executed entirely on the Central Processing Unit (CPU). The local user interface was constructed using the desktop-based Tkinter library, whereas the real-time video stream distribution was handled by a lightweight Flask-based streaming server utilizing the Motion JPEG (MJPEG) protocol. The generated traffic data is periodically sent via asynchronous POST requests through a Laravel 11 REST API layer to a structured MySQL database and visualized on a web dashboard. This modular software framework was engineered to mitigate the performance drop typically observed in default pre-trained object detectors due to changing camera perspectives, small object scales, and challenging outdoor lighting variations.

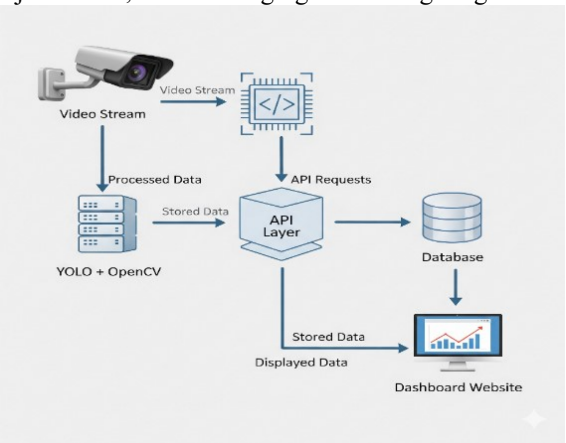

Figure 4. System architecture

Equations should be placed at the center of the line and provided consecutively with equation numbers in parentheses flushed to the right margin, as in (1). The use of Microsoft Equation Editor or MathType is preferred.

3.2. Evaluation of Limited Fine-Tuning Scenarios

In the early experimental stages, a fine-tuning approach was conducted using a localized dataset from Politeknik Negeri Bengkalis combined with custom public traffic domains from Roboflow, totaling

approximately 4,000 annotated images. The bounding box distributions and class balances were systematically evaluated prior to the training process.

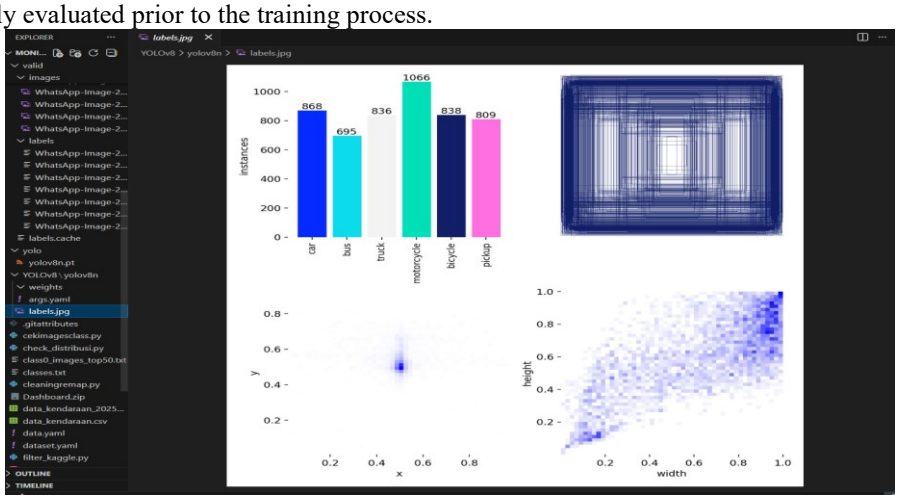

Figure 5. Visualization of Fine-Tuning Dataset Distribution

The empirical results indicated that limited fine-tuning failed to yield significant performance gains, particularly for localized cargo vehicles such as pickup trucks and heavy trucks. This sub-optimal performance stemmed from critical constraints: extreme sample scarcity in specific cargo classes that induced model overfitting, high environmental variance (lighting changes and atypical camera mounting angles) that could not be fully generalized by a compact dataset, and hardware limits on CPU-only training that restricted the epoch size. Consequently, full model retraining was bypassed, and performance optimization was directed toward a logical rule-based post-processing system executed immediately after raw tensor detection.

3.3. Implementation of Rule-Based Post-Processing

To resolve the systematic classification bottlenecks without a resource-heavy retraining process, a rule-based post-processing framework was implemented to filter the raw bounding boxes produced by the YOLOv8n network. This custom processing tier directly exploits the native spatial metrics of the raw detections: bounding box pixel area, aspect ratio (w/h), and the vertical centroid position (center_y) relative to the complete frame height (frame_height). To harmonize the raw outputs with a static camera perspective, a linear geometric correction function was applied to normalize the bounding box bounds. The perspective adjustment factor is computed via the following equation:

$$perspective\_factor = \max\left(0.6, \frac{center\_y}{frame\_height}\right) \quad (1)$$

The normalized object area (adjusted_area), which accounts for physical depth scaling, is subsequently derived according to the equation below

$$adjusted\_area = \frac{area}{perspective\_factor} \quad (2)$$

Equations (1) and (2) guarantee that distant vehicles situated near the upper horizon of the camera viewport are not falsely eliminated due to their compressed pixel footprints. Furthermore, because the generic COCO dataset lacks an explicit class designation for pickup trucks, a class simplification logic was established to programmatically merge detected pickup truck profiles into the broader truck class based on shared light-cargo functional roles and spatial visual profiles from an overhead view.

3.4. Performance Comparison: Baseline, Fine-Tuning, and Post-Processing

Comparative benchmark tests were performed using localized static test video data from the campus sector to evaluate the performance of the three distinct architectural setups. The empirical metrics captured across each development setup are structured cleanly below.

Table 2. Comparative Performance Evaluation of YOLOv8 Configurations

Methodology	Ground Truth	Detection Count	TP	FP	FN	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)

Baseline YOLOv8 (No Post-Processing)	47	103	47	57	0	45.19	100.00	62.25	45.19
YOLOv8 with Fine-Tuning	47	9	9	2	38	81.82	19.15	31.03	18.37
YOLOv8 with Rule-Based Post-Processing	47	46	43	3	4	93.48	91.49	92.47	86.00

Based on the quantitative metrics, the default pre-trained YOLOv8 model achieved a perfect total recall of 100.00% but registered an extremely poor precision rate of 45.19%. This indicates an overwhelming accumulation of false positives due to the absence of spatial filters, which caused objects to be redundantly counted across multiple frames. Conversely, the limited fine-tuning method triggered severe model degradation, dragging the total recall down to a critical 19.15% due to overfitting on the small localized sample set. The robust rule-based post-processing method outpaced both setups, driving down false positive counts and elevating the total precision rate to 93.48% with an overall system accuracy of 86.00% under ideal conditions. The obtained precision of 93.48% demonstrates competitive performance compared with previous YOLO-based vehicle monitoring studies, which reported accuracy levels ranging from 90% to 97.5% [3], [4], [10]. The integration of rule-based post-processing contributed to reducing misclassification between visually similar vehicle categories while maintaining computational efficiency.

3.5. Environmental Scenario Analysis

The system was stress-tested across three field video streams reflecting distinctly different weather and ambient lighting conditions:

1. *Test Video 1 - Campus Site Daytime (Clear Weather)*: Represents an ideal setup characterized by optimal ambient sunlight. Without post-processing, system precision sat at 45.19% due to unchecked frame-by-frame redundancies. Applying the rule-based post-processing layer increased the precision rate by +48.29% to reach 93.48%, maintaining an outstanding F1-score of 92.47%.
2. *Test Video 2 - Campus Site Nighttime (Heavy Rain)*: Presents an extreme challenge due to low illumination and significant visual noise from headlight reflections on wet asphalt. Without filters, the system registered a poor accuracy of 12.24% with high false-positive tracking on ghost car profiles. The post-processing logic lifted the recall rate from 27.27% to 43.18%, though precision remained bottlenecked at 15.97% due to the severe optical degradation caused by rain.
3. *Test Video 3 - Urban Roads Daytime (Rainy Weather)*: Tests the system's resilience in a complex urban environment with dense background noise. Without post-processing, precision plummeted to 24.90% with a massive false positive accumulation of 187 entities. Integrating the rule-based spatial adjustments significantly minimized false alarms, yielding a precision of 88.52%, a recall of 87.10%, and a total system accuracy of 78.26%.

Table 3. Environmental Characteristics and Evaluation Dataset Properties

Evaluation Scenario	Location Context	Enviromental & Lighting Conditions	Key Traffic Flow Metrics & Class Distribution
Test Video 1	Polbeng Internal Campus Area	Fair Daytime, Clear Weather, High Illumination, Static Camera Perspective	Ground Truth: 47 vehicles (Dominated by Motorcycles [65%] and Cars [25%])
Test Video 2	Polbeng Internal Campus Area	Nighttime, Active Precipitation (Rain), Low Illumination with Shadow Reflections	Ground Truth: 44 vehicles (High motorcycle density during local student commutes)
Test Video 3	Public Roads (Bengkalis City Center)	Daytime, Adverse Weather (Heavy Rain), Complex Visual Occlusions	Ground Truth: 62 vehicles (Higher presence of standard commercial utility vans and cars)
Live CCTV Stream	Active Polbeng Network Nodes	Varied real-time lighting transitions (Continuous deployment testing environment)	Unstructured real-world continuous traffic flow tracking with low latency

As compiled in Table 3, the evaluation framework deliberately encompasses varying illumination thresholds, geometric perspectives, and ambient interferences to thoroughly verify the algorithm's operational elasticity. The target class distribution heavily favors two-wheeled motorbikes, which directly

mirrors the local student demographic commuting trends inside Politeknik Negeri Bengkalis. These multi-environmental datasets provide the empirical boundaries necessary to configure the spatial definitions inside the post-processing filter layer without running into overfitting limitations.

Table 4. Detection Results for Test Video 1 (**Clear Daytime**) - **Without Post-Processing**

Vehicle Type	Ground Truth	Detection Count	TP	FP	FN	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
Motorcycle	41	68	41	27	0	60.29	100.00	75.23	60.29
Car	4	24	4	20	0	16.67	100.00	28.57	16.67
Truck	2	11	2	9	0	18.18	100.00	30.77	18.18
Bus	0	0	0	0	0	0.00	0.00	0.00	0.00
Bicycle	0	0	0	0	0	0.00	0.00	0.00	0.00
Total	47	103	47	57	0	45.19	100.00	62.25	45.19

Table 5. Detection Results for Test Video 1 (**Clear Daytime**) – **With Post-Processing**

Vehicle Type	Ground Truth	Detection Count	TP	FP	FN	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
Motorcycle	41	37	37	0	4	100.00	90.24	94.87	90.24
Car	4	7	4	3	0	57.14	100.00	72.73	57.14
Truck	2	2	2	0	0	100.00	100.00	100.00	100.00
Bus	0	0	0	0	0	0.00	0.00	0.00	0.00
Bicycle	0	0	0	0	0	0.00	0.00	0.00	0.00
Total	47	46	43	3	4	93.48	91.49	92.47	86.00

Table 6. Detection Results for Test Video 2 (**Nighttime Heavy Rain**) – **Without Post-Processing**

Vehicle Type	Ground Truth	Detection Count	TP	FP	FN	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
Motorcycle	44	12	12	0	32	100.00	27.27	42.86	27.27
Car	0	53	0	53	0	0.00	0.00	0.00	0.00
Truck	0	2	0	0	0	100.00	100.00	100.00	0.00
Bus	0	0	0	0	0	0.00	0.00	0.00	0.00
Bicycle	0	0	0	0	0	0.00	0.00	0.00	0.00
Total	44	65	12	53	32	18.18	27.27	21.82	12.24

Table 7. Detection Results for Test Video 2 (**Nighttime Heavy Rain**) – **With Post-Processing**

Vehicle Type	Ground Truth	Detection Count	TP	FP	FN	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
Motorcycle	44	19	19	0	25	100.00	43.18	60.32	43.18
Car	0	99	0	99	0	0.00	0.00	0.00	0.00
Truck	0	2	0	0	0	80.00	100.00	88.89	80.00
Bus	0	0	0	0	0	0.00	0.00	0.00	0.00
Bicycle	0	0	0	1	0	0.00	0.00	0.00	0.00
Total	44	118	19	100	25	15.97	43.18	23.31	13.19

Table 8. Detection Results for Test Video 3 (**Daytime Rainy Weather**) – **Without Post-Processing**

Vehicle Type	Ground Truth	Detection Count	TP	FP	FN	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
Motorcycle	42	69	42	27	0	60.87	100.00	75.68	60.87
Car	16	96	16	80	0	16.67	100.00	28.57	16.67
Truck	4	83	4	79	0	4.82	100.00	9.20	4.82
Bus	0	1	0	1	0	0.00	0.00	0.00	0.00
Bicycle	0	0	0	0	0	0.00	0.00	0.00	0.00
Total	62	249	62	187	25	24.90	100.00	39.87	24.90

Table 9. Detection Results for Test Video 3 (**Daytime Rainy Weather**) – **With Post-Processing**

Vehicle Type	Ground Truth	Detection Count	TP	FP	FN	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
Motorcycle	42	34	34	0	8	100.00	80.95	89.47	80.95
Car	16	22	16	6	0	72.73	100.00	84.21	72.73
Truck	4	5	4	1	0	80.00	100.00	88.89	80.00
Bus	0	0	0	0	0	0.00	0.00	0.00	0.00
Bicycle	0	0	0	0	0	0.00	0.00	0.00	0.00
Total	62	61	54	7	8	88.52	87.10	87.80	78.26

As displayed in Table 4-9, the implementation of the proposed rule-based post-processing method significantly alters the classification performance metrics. Under the baseline configuration, the raw YOLOv8n model delivers a perfect recall of 100.00% but suffers from an excessively low total precision of 45.19% due to 57 recorded false-positive events. This occurs because a single vehicle triggers continuous multi-frame overlapping detections without any geometric constraints.

By introducing the rule-based post-processing filtering layer, the overall precision drastically escalates to 93.48% (a +48.29% surge), and the macro accuracy jumps to 86.00% (a +40.81% increase). Although the stricter boundary logic slightly compromises the recall to 91.49% due to four true vehicles falling just outside the rigid filter thresholds, the overall F1-Score substantially climbs from 62.25% to 92.47%. This statistical evolution validates that rule-based geometric filtering serves as a powerful and CPU-friendly substitute for extensive model retraining. Furthermore, regarding tracking-specific metrics, the multi-object tracking (MOT) module utilizing ByteTrack effectively locks down the unique Track IDs across sequential streaming updates. It eliminates counting duplications and boundaries issues by mapping trajectory paths instead of evaluating isolated bounding box frames.

3.6. Multi-Object Tracking Performance and Speed Indexing

To resolve the frame-by-frame counting redundancies inherent in standard object detectors, the ByteTrack multi-object tracking engine was integrated into the pipeline. ByteTrack permanently maps object boxes across sequential video frames to assign a distinct Track ID to each passing vehicle. To avoid registering temporary visual artifacts or noise, a minimum frame activation threshold was introduced. Empirical results showed that the tracking identities remained highly stable under normal traffic speeds, with track loss occurring exclusively during prolonged physical occlusions lasting longer than 3 seconds. The stable object identities observed during testing are consistent with findings reported in ByteTrack and DeepSORT studies, where robust multi-object tracking significantly reduced duplicate counting and identity switching problems in crowded traffic scenes [17], [19].

Once tracking stability was achieved, vehicle speeds were evaluated using a heuristic spatial approach based on centroid movement displacement per frame relative to the system's processing frame rate (FPS). The relative velocity data generated from the live CCTV monitoring stream is organized systematically below.

Table 10. Relative Speed Estimation Metrics for Campus Traffic

Vehicle Type	Sample Size	Min Speed (km/h)	Max Speed (km/h)	Average Speed (km/h)
Motorcycle	28	30	70	50
Car	4	25	60	45
Bicycle	5	5	20	15

Field verification confirmed that the calculated average speed of 50 km/h for motorcycles and 45 km/h for cars aligned closely with real-world observations within the restricted speed limits of the campus perimeter. The quantitative error calculation comparing the system's velocity tracking (Vsystem) against manual visual ground truth (Vfield) was modeled through the following equation:

$$Error(\%) = \frac{|V_{system} - V_{field}|}{V_{field}} \times 100\% \quad (3)$$

The results of equation (3) showed that the speed estimation error for medium-speed vehicles like motorcycles remained well within an acceptable observational range at 23.42%. However, the percentage error scaled up to 58.66% for low-speed entities such as bicycles. This inflation is primarily a mathematical artifact caused by the extreme sensitivity of percentage error formulas to small base denominators, as the absolute speed deviation remained within a minor, acceptable margin for statistical traffic monitoring.

3.7. CPU Evaluation of YOLOv8 Model Variants

The final phase of this research benchmarked three Ultralytics model sizes YOLOv8n (nano), YOLOv8s (small), and YOLOv8m (medium) on a client CPU infrastructure. The operational parameters, confidence limits, and post-processing adjustments were kept identical across all runs to guarantee strict evaluation objectivity.

Table 11. Comprehensive CPU Performance Matrix of YOLOv8 Model Sizes

Metric	YOLOv8n (Nano)	YOLOv8s (Small)	YOLOv8m (Medium)
Average FPS	14.80	7.09	3.32
Inference Latency (ms)	63.10	135.97	296.02
CPU Usage (%)	35.37	41.78	46.92
RAM Usage (MB)	10,774.42	10,557.74	10,389.60
Total Processed Frames	1,500	1,500	1,500
Gross Detection Count	2,922	3,287	3,913

The comparative data demonstrated that the YOLOv8n variant achieved the highest execution speed, averaging 14.80 FPS with an inference latency of just 63.10 ms while drawing a modest 35.37% CPU load. Conversely, the YOLOv8m model captured the highest number of raw bounding box instances (3,913 detections) but crippled system performance, dropping execution speeds down to 3.32 FPS with latency scaling up to 296.02 ms. For real-time monitoring workflows, maintaining a stable and high frame processing rate is far more critical than raw instance counts per frame. Low execution speeds on the *small* and *medium* models trigger frame drops, which cause the tracking engine (ByteTrack) to lose target continuity and generate duplicate counts. Therefore, YOLOv8n was chosen as the optimal engine, offering the best balance of speed and tracking accuracy for CPU-constrained environments. Similar observations were reported in YOLOX [16], where lightweight models demonstrated better real-time performance on resource-constrained devices.

4. Conclusion

This study successfully developed a real-time vehicle detection and classification system based on the YOLOv8 architecture utilizing CCTV video streams within the campus environment. The completed platform demonstrates robust capabilities in automated vehicle detection, classification, multi-object tracking, relative speed estimation, and asynchronous database logging. Although the default pre-trained YOLOv8 model exhibited inherent misclassifications under specific edge-case scenarios particularly among light-cargo transport vehicles the implementation of the proposed rule-based post-processing framework significantly improved classification consistency without requiring a resource-heavy model fine-tuning workflow. Furthermore, the integration of the ByteTrack algorithm successfully maintained strict target identity continuity, thereby minimizing counting redundancies caused by duplicate frame detections.

The comparative evaluation among the YOLOv8n, YOLOv8s, and YOLOv8m model variants establishes that YOLOv8n delivers the most stable operational performance on CPU-bound client devices, yielding the highest frame rate (FPS) and the lowest inference latency. This high frame-rate stability directly enhances multi-object tracking precision and overall counting accuracy, confirming YOLOv8n as the optimal baseline engine for this deployment. Concurrently, the heuristic speed estimation module produces velocity metrics within realistic parameters that accurately represent the actual traffic dynamics observed within the campus perimeter. Although these speed tracking results remain relative and operate without exhaustive physical camera lens calibration, the generated observational data is sufficiently robust to support data-driven traffic monitoring and policy evaluations.

References

[1] A. Krizhevsky, I. Sutskever, And G. E. Hinton, “Imagenet Classification With Deep Convolutional Neural Networks.” [Online]. Available: Http://Code.Google.Com/P/Cuda-Convnet/

[2] R. G. Fajri, I. Santoso, And Y. A. Adi Soetrisno, “Perancangan Program Pendeteksi Dan Pengklasifikasi Jenis Kendaraan Dengan Metode Convolutional Neural Network (Cnn) Deep Learning,” *Transient: Jurnal Ilmiah Teknik Elektro*, Vol. 9, No. 1, Pp. 97–106, Mar. 2020, Doi: 10.14710/Transient.V9i1.97-106.

[3] D. Iskandar Mulyana And M. A. Rofik, “Implementasi Deteksi Real Time Klasifikasi Jenis Kendaraan Di Indonesia Menggunakan Metode Yolov5,” *Jurnal Pendidikan Tambusai*, Vol. 6, No. 3, Pp. 13971–13982, Jul. 2022, Doi: 10.31004/Jptam.V6i3.4825.

[4] A. S. Kusuma, A. I. Pradana, And B. W. Pamekas, “Pengembangan Sistem Perhitungan Jumlah Kendaraan Berdasarkan Jenis Kendaraan Menggunakan Algoritma Yolo Secara Realtime,” *Skanika: Sistem Komputer Dan Teknik Informatika*, Vol. 7, No. 2, Pp. 166–179, Jul. 2024, Doi: 10.36080/Skanika.V7i2.3201.

[5] L. Adi Kurniawan *Et Al.*, "Sistem Klasifikasi Jenis Dan Warna Kendaraan Secara Real-Time Menggunakan Metode K-Nearest Neighbor Dan Framework Yolact," *Jepin (Jurnal Edukasi Dan Penelitian Informatika)*, Apr. 2021.

[6] A. A. B, A. Amin, And M. W. Kasrani, "Penerapan Metode Yolo Object Detection V1 Terhadap Proses Pendeteksian Jenis Kendaraan Di Parkiran," *Jurnal Teknik Elektro Uniba (Jte Uniba)*, Vol. 6, No. 1, Pp. 194–199, Oct. 2021, Doi: 10.36277/Jteuniba.V6i1.130.

[7] A. Riansyah And A. H. Mirza, "Pendeteksi Mobil Berdasarkan Merek Dan Tipe Menggunakan Algoritma Yolo," *Smatika Jurnal*, Vol. 13, No. 01, Pp. 43–52, Jun. 2023, Doi: 10.32664/Smatika.V13i01.719.

[8] F. Jupiter, E. S. Negara, Y. N. Kunang, And M. I. Herdiansyah, "Implementasi Algoritma Cnn Dan Yolo Untuk Mendeteksi Jenis Kendaraan Pada Jalan Raya," *Explore: Jurnal Sistem Informasi Dan Telematika*, Vol. 14, No. 2, P. 110, Dec. 2023, Doi: 10.36448/Jsit.V14i2.3259.

[9] B. Aditya Pratama, S. Rahman, And A. Sembiring, "Klasifikasi Jenis Kendaraan Pada Jalan Raya Menggunakan Yolov7," *Jurnal Informatika Teknologi Dan Sains (Jinteks)*, Vol. 5, No. 4, Pp. 661–666, Dec. 2023, Doi: 10.51401/Jinteks.V5i4.3493.

[10] W. P. N. Putra, A. I. Pradana, And N. Nurchim, "Implementasi Sistem Penghitungan Volume Kendaraan Menggunakan Yolov8," *Jurnal Fasilkom*, Vol. 14, No. 2, Pp. 443–450, Aug. 2024, Doi: 10.37859/Jf.V14i2.7395.

[11] Muhammad Rosyan Amanullah, R. M. Akbar, And Y. D. Rosita, "Simulasi Deteksi Dan Hitung Jumlah Kendaraan Menggunakan Yolov4 Pada Cctv Persimpangan Jalan Raya," *Seminar Nasional Fakultas Teknik*, Vol. 2, No. 1, Pp. 96–101, Sep. 2023, Doi: 10.36815/Semastek.V2i1.128.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection,"in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Las Vegas, NV, USA, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[13] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Honolulu, HI, USA, 2017, pp. 6517–6525, doi: 10.1109/CVPR.2017.690.

[14] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv preprint arXiv:1804.02767, Apr. 2018.

[15] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," arXiv preprint arXiv:2004.10934, Apr. 2020.

[16] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO Series in 2021," arXiv preprint arXiv:2107.08430, Jul. 2021.

[17] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, "ByteTrack: Multi-Object Tracking by Associating Every Detection Box," in Proc. Eur. Conf. Comput. Vis. (ECCV), vol. 13682, Tel Aviv, Israel, 2022, pp. 1–21, doi: 10.1007/978-3-031-20047-2_1.

[18] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," in Proc. IEEE Int. Conf. Image Process. (ICIP), Phoenix, AZ, USA, 2016, pp. 3464–3468, doi: 10.1109/ICIP.2016.7533003.

[19] N. Wojke, A. Bewley, and D. Paulus, "Simple Online and Realtime Tracking with a Deep Association Metric," in Proc. IEEE Int. Conf. Image Process. (ICIP), Beijing, China, 2017, pp. 3645–3649, doi: 10.1109/ICIP.2017.8296962.

[20] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, "FairMOT: On the Fairness of Detection and Re-Identification in Multiple Object Tracking," Int. J. Comput. Vis., vol. 129, no. 11, pp. 3069–3087, Nov. 2021, doi: 10.1007/s11263-021-01513-4.

[21] M. W. Ahmed, M. Adnan, M. Ahmed, D. Janssens, G. Wets, A. Ahmed, and W. Ectors, "From Stationary to Nonstationary UAVs: Deep-Learning-Based Method for Vehicle Speed Estimation," *Algorithms*, vol. 17, no. 12, p. 558, 2024, doi: 10.3390/a17120558.

[22] M. A. Rahmat, I. Indrabayu, A. Achmad, and A. E. U. Salam, "A Thorough Review of Vehicle Detection and Distance Estimation Using Deep Learning in Autonomous Cars," *JOIV: International Journal on Informatics Visualization*, vol. 8, no. 4, pp. 2362–2369, 2024, doi: 10.62527/joiv.8.4.2665.

[23] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, Jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[24] W. Liu et al., "SSD: Single Shot MultiBox Detector," in Proc. Eur. Conf. Comput. Vis. (ECCV), Amsterdam, Netherlands, 2016, pp. 21–37, doi: 10.1007/978-3-319-46448-0_2.

[25] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Venice, Italy, 2017, pp. 2980–2988, doi: 10.1109/ICCV.2017.324.

[26] M. S. Sawah, "Advancements in UAV-based traffic monitoring: a systematic review of deep learning and edge computing," *Bulletin of Electrical Engineering and Informatics*, vol. 13, no. 1, pp. 451–460, 2024, doi: 10.11591/eei.v13i1.6521.