Traffic Signal Optimization and Emergency Vehicle Priority System

Prakhar Kumar Srivastava
Dept. of CSE
BML Munjal University
Gurugram, India
prakharkumar.srivastava.22cse@bmu.edu.in

Dakshi Arora
Dept. of CSE
BML Munjal University
Gurugram, India
dakshi.arora.22cse@bmu.edu.in

Parag Garg
Dept. of CSE
BML Munjal University
Gurugram, India
parag.garg.22cse@bmu.edu.in

Gagan Singhal
Dept. of CSE
BML Munjal University
Gurugram, India
gagan.singhal.22cse@bmu.edu.in

Anusha Singh
Dept. of CSE
BML Munjal University
Gurugram, India
anusha.22cse@bmu.edu.in

***Abstract* — Traffic congestion in urban environments poses significant challenges for emergency response, fuel efficiency, and commuter satisfaction. This paper presents a simulation-based approach to intelligent traffic signal optimization and emergency vehicle prioritization using real-time image processing and the Wokwi simulation environment. Our system employs YOLO (You Only Look Once) for vehicle detection and traffic density estimation across multiple lanes. Based on the computed density, adaptive timers dynamically allocate green signals to reduce idle time and enhance throughput. In parallel, an emergency vehicle detection module overrides the density logic, granting immediate right-of-way to emergency vehicles using distinct visual features. The system is fully simulated using Pygame, enabling realistic visualization of traffic behavior, signal transitions, and emergency handling in real time. Threaded execution allows simultaneous monitoring and control, ensuring low-latency responsiveness to evolving traffic conditions. The integration of these techniques demonstrates improved average wait time reduction and enhanced prioritization for emergency vehicles compared to static signal systems. This simulation framework serves as a testbed for future integration into smart city infrastructure and IoT-based traffic management networks.**

Keywords—YOLO-based Vehicle Detection, Traffic Signal Optimization, Emergency Vehicle Priority, Real-Time Simulation, Pygame, Intelligent Transportation Systems.

I. INTRODUCTION

Urban traffic congestion is one of the most pressing challenges in modern cities, affecting not only commuter experience but also public safety and environmental sustainability. Rapid urbanization and increased vehicle ownership have led to traffic signal systems becoming overwhelmed, particularly during peak hours. Traditional traffic control infrastructures operate on static timers—cycling signals at fixed intervals regardless of real-time road conditions. This leads to inefficiencies such as prolonged wait times at empty intersections, unnecessary idling, elevated fuel consumption, and increased emission levels. Moreover, the lack of dynamic prioritization significantly delays emergency response times for ambulances, fire trucks, and police vehicles.

To address these issues, intelligent transportation systems (ITS) have emerged as a promising area of research and development. These systems leverage technologies such as computer vision, Internet of Things (IoT), and machine learning to optimize traffic flow, reduce congestion, and improve emergency response. Adaptive traffic signal control, which dynamically adjusts signal phases based on live traffic data, has gained significant attention as a practical approach to increasing intersection throughput and reducing vehicle delays.

In addition to regular traffic optimization, ensuring right-of-way for emergency vehicles is vital for minimizing response times and saving lives. However, most deployed systems still lack a robust, cost-effective, and scalable solution that can handle real-time emergency detection and signal override.

The goal of this paper is to propose a simulation-based intelligent traffic control system that integrates YOLOv2 (You Only Look Once) for vehicle detection and classification, adaptive signal control based on traffic density, and an emergency vehicle priority mechanism. The system is designed using Python and simulated through Pygame for intersection behavior, Wokwi for ESP32-based LED hardware emulation, and MQTT for communication between software and hardware layers. This architecture serves as a flexible testbed for smart city traffic scenarios and demonstrates significant performance improvements in simulation metrics such as average wait time and emergency clearance delay.

A. *Related Work*

Previous work in the domain of adaptive traffic control has explored a variety of approaches. Traditional vision-based vehicle detection techniques, such as background subtraction and Haar cascades [1], often struggle with real-time accuracy under varying weather and lighting conditions. With the advancement of deep learning, models like YOLO have demonstrated superior performance in object detection tasks, particularly for traffic surveillance [2].

Several researchers have implemented traffic signal optimization systems using Yolov4 or Yolov3. For example, Bhargava et al. [3] demonstrated Yolov4's ability to identify traffic density and adjust signal durations accordingly. However, their system did not incorporate emergency vehicle handling. Similarly, Kale et al. [4] developed a Pygame-based simulation model using Yolov4 for density-driven traffic control, but the system lacked real-time multithreading or hardware integration.

Other works have attempted to address emergency prioritization. RFID-based systems [5] provide detection capabilities but require infrastructure deployment in vehicles. Siren-detection methods using audio signals [6] often suffer

from background noise interference and false positives. Recent advances include integrating YOLOv8 for ambulance detection [7], but most lack a fully integrated adaptive control and emergency override framework.

This paper builds upon these concepts by presenting a comprehensive solution that combines YOLO-based traffic and emergency detection with a multithreaded simulation environment and hardware control via ESP32, all coordinated through MQTT communication. The approach aims to be lightweight, scalable, and suitable for real-time deployment in smart cities.

II. SYSTEM ARCHITECTURE

The proposed system uses a camera feed from the intersection to perform object detection and classification via a YOLO model. Each frame is processed to identify vehicles in each lane. Detections are filtered (confidence threshold ~50%) and tallied per lane. Vehicle classes (car, bus, truck, etc.) are treated uniformly as traffic flow; emergency vehicles (e.g. ambulance) are given a special class. We compute traffic density simply as the count of vehicles waiting in a lane (possibly weighted by type or distance).

A control logic module then determines green times for each lane. In normal operation, lanes with more queued vehicles receive longer green duration. For example, we implement a green-time allocation proportional to detected count (with min/max bounds to prevent starvation). When an ambulance is detected approaching the intersection, an emergency override is triggered: the current green is extended (or switched immediately) to the ambulance's lane, while all other lanes switch to red. This guarantees a clear path. Once the ambulance has passed (or left the scene), normal adaptive sequencing resumes. Such real-time emergency prioritization has been shown to drastically reduce response delays.

Figure 1 illustrates the high-level system architecture. A Python-based simulation (Pygame) generates traffic and performs vision processing; it publishes each lane's state (vehicle count, emergency flag) via MQTT to a broker. An ESP32 microcontroller (simulated in Wokwi) subscribes to this information and drives the physical LEDs for the traffic lights. Two threads run in parallel on the Python side: one for the simulation loop (vehicle movement and GUI updates) and one for image capture/detection and MQTT publishing. This multithreaded design ensures real-time performance without blocking the interface.

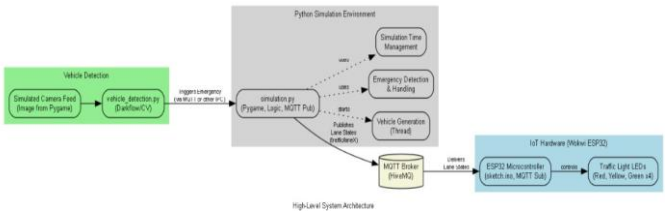

Figure 1 System architecture: A Pygame-based Python simulation publishes traffic densities to an MQTT broker, which an ESP32 microcontroller subscribes to in order to control LED traffic lights.

The vision pipeline uses OpenCV+YOLO to count vehicles and detect emergencies. In our implementation, the Pygame simulation represents a 4-way intersection with four traffic signals, each showing a countdown timer for green/yellow phases. Vehicles (cars, buses, bikes, trucks, etc.) are generated randomly on each approach. The Yolov2 detector analyzes each video frame and returns bounding boxes and class labels. The lane counts (and any detected ambulance) are packaged into JSON and published to the MQTT broker. The ESP32 code (Arduino/C++) runs an MQTT client that subscribes to these messages and switches its GPIO outputs to control red/yellow/green LEDs. This decoupled architecture makes it easy to integrate actual IoT hardware (cameras, controllers) in a future real-world deployment.

III. ALGORITHM AND METHODOLOGY

This section outlines the internal logic, algorithms, and processing pipeline that govern the behavior of the proposed adaptive traffic signal system. The methodology can be divided into five primary components: vehicle detection using Yolov2, traffic density calculation, emergency vehicle detection, signal timing algorithm, and emergency override mechanism.

A. Yolov2-Based Vehicle Detection

Yolov2 (You Only Look Once version 4) is employed for real-time object detection due to its speed and accuracy. It divides each input frame into a grid and predicts bounding boxes and class probabilities directly using a single neural network pass. The model used in this project was trained on the COCO dataset and is capable of detecting multiple vehicle classes such as cars, buses, trucks, and motorcycles.

The Pygame simulation renders a real-time graphical intersection, which is continuously captured as video frames. These frames are passed to the Yolov2 model via OpenCV's DNN module. The model returns class labels and bounding box coordinates. To determine per-lane traffic density, each vehicle is mapped to its respective lane based on its position in the frame.

A confidence threshold of 0.5 is used to filter out low-confidence detections and improve reliability. Emergency vehicles (e.g., ambulances) are either detected using YOLO (if trained on such classes) or marked visually in the simulation with a distinguishable color or siren icon for accurate identification.

B. Lane-Wise Density Calculation

After extracting vehicle positions from YOLO's output, a counter tallies the number of vehicles detected in each lane. Each lane is represented as a bounding region on the screen, and any detection falling within that region is considered part of that lane's queue.

Let nin_ini be the number of vehicles detected in lane iii. The normalized traffic density is computed as:

$$di=ni\sum j=1Nnj$$

where NNN is the number of lanes (typically 4). These normalized values are used to proportionally allocate green signal time to each lane.

C. Adaptive Green Time Allocation

The green light duration for each lane is calculated based on traffic density. The system dynamically adjusts green time using a linear scaling formula:

$$\mathbf{Ti=Tmin+\alpha \cdot ni}$$

Where:

Ti is the green time for lane i,

Tmin is the minimum green time (e.g., 5 seconds),

α is a scaling constant,

ni is the vehicle count in lane i.

This ensures that heavily congested lanes get proportionally more green time, while still preventing starvation by capping the maximum green duration. The yellow phase is set to a constant (e.g., 3 seconds) for safety between transitions.

D. Emergency Vehicle Detection and Override Logic

When an emergency vehicle is detected (via YOLO label or visual marker), an interrupt is triggered. The logic checks which lane the vehicle is currently in or approaching. Once identified:

1. The current green signal is overridden immediately or extended (if already green).
2. All other lanes are set to red.
3. The emergency lane remains green until the vehicle exits the scene or after a fixed buffer time (e.g., 5–8 seconds post-detection).

After the emergency has cleared, the system resumes normal adaptive control.

E. System Synchronization and Timing Flow

To maintain real-time performance, all operations are divided across threads:

Thread 1: Runs the simulation GUI, updates vehicle positions, and renders visuals.

Thread 2: Captures frames, performs YOLO inference, and updates lane counts.

Thread 3: Publishes JSON messages over MQTT to ESP32 (optional but recommended).

Thread 4 (if used): Handles manual or timed emergency triggers.

Thread-safe queues are used to exchange data between components. This ensures no frame delays or dropped detection events while maintaining ~30 FPS GUI responsiveness.

IV. IMPLEMENTATION

The proposed system is implemented using a hybrid simulation environment that combines Python-based software modules with a virtualized microcontroller platform. The implementation comprises four key components: hardware simulation using Wokwi and ESP32, a YOLO-based image processing pipeline, a multithreaded execution engine for real-time responsiveness, and MQTT-based communication for synchronizing simulation logic with physical signal control.

A. Hardware Implementation using Wokwi and ESP32

To emulate real-world deployment, the system uses an ESP32 DevKit-V1 microcontroller simulated within the Wokwi environment. The ESP32 is programmed using the Arduino IDE and controls 12 LEDs, arranged as three lights (red, yellow, green) for each of the four lanes of a typical intersection.

Configuration Details:

GPIO Mapping: Each LED is connected to a designated GPIO pin.

Circuit Setup: Each LED connects to ESP32 via a current-limiting resistor; cathodes are grounded.

Network Communication: The ESP32 connects to Wi-Fi and subscribes to a public MQTT broker (e.g., HiveMQ).

Signal Logic: Incoming MQTT messages determine which LED (signal) is active for each lane.

The ESP32 simply reacts to the published timing instructions. All decision-making is handled on the Python side, ensuring separation of control and actuation.

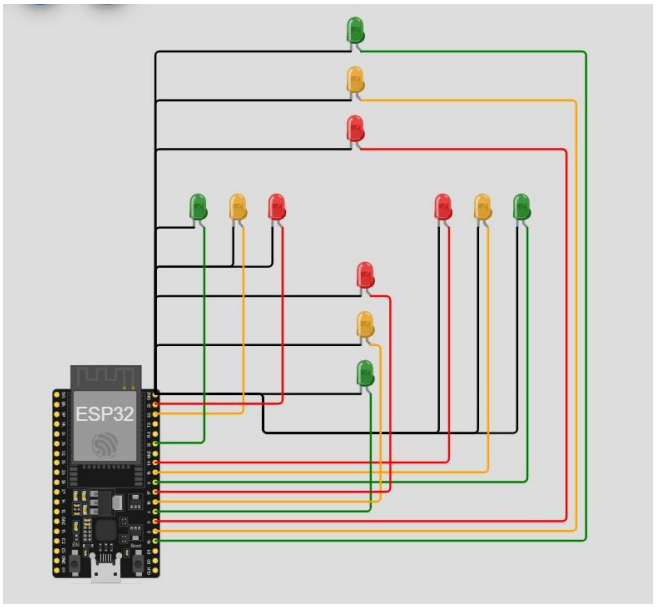

Figure 2 ESP32 traffic light hardware setup (Wokwi simulator). The ESP32 microcontroller outputs drive 12 LEDs (4 lanes × 3 colors) representing the intersection signals. GPIO pins are assigned for each red, yellow, and green light as shown

B. Image Processing Pipeline

The detection system is built using Python 3, OpenCV, and Yolov2. The Pygame simulation window is captured in real-time and fed into the detection model to extract vehicle types and positions.

Key Components:

Frame Capture: Each frame from Pygame is captured at ~5 FPS (once every 0.2 seconds).

Model Loading: Yolov2 is initialized using the OpenCV DNN or PyTorch module.

Preprocessing: Frames are resized to YOLO's expected input shape (e.g., 416×416) and normalized.

Detection Output: The model returns bounding boxes and class labels for each object.

Lane Mapping: Detected vehicles are assigned to lanes based on their position on the frame.

Emergency vehicles (e.g., ambulances) are detected either as a specific YOLO class or through a visual marker embedded in the simulation (e.g., a siren icon).

C. *Multithreading and Real-Time Control Flow*

To ensure non-blocking, real-time responsiveness, the entire system operates using Python's threading module. This allows independent execution of detection, simulation, and communication logic.

Threads Used:

Thread 1: Simulation rendering via Pygame (vehicle movement, GUI).

Thread 2: Yolov2 detection loop (frame capture, object detection, lane counting).

Thread 3: MQTT publishing thread (sends JSON data to ESP32).

Thread 4 (optional): Emergency injection or manual override handler (e.g., via keypress or UDP input).

Shared resources such as lane counts, emergency flags, and signal durations are stored in synchronized variables or thread-safe queues. This prevents race conditions or inconsistent states while maintaining simulation stability at ~30–60 FPS.

D. *MQTT Communication Layer*

The system uses MQTT, a lightweight publish-subscribe protocol, for communication between the Python controller and the ESP32 microcontroller. The Python script acts as a publisher, sending updated lane states in real-time, while the ESP32 acts as a subscriber.

V. RESULTS AND DISCUSSION

To evaluate the effectiveness of the proposed traffic signal optimization and emergency vehicle prioritization system, a series of simulation trials were conducted using the integrated Pygame and Wokwi environments. These simulations were designed to compare the performance of the YOLO-based adaptive system against a traditional static-timed traffic light control system under varying traffic conditions.

A. *Simulation Setup*

The simulation represents a four-way intersection, with each lane capable of generating vehicles (cars, buses, trucks, bikes, and ambulances) using a Poisson distribution to simulate real-world traffic flow. Emergency vehicles were spawned either randomly or triggered manually. Each lane had a corresponding signal with a red, yellow, and green light managed by the ESP32 (via MQTT).

Key Parameters:

Frame Rate: 30–60 FPS (Pygame).

Detection Interval: Every 200 ms.

Simulation Duration per Trial: 5 minutes.

Control Modes: Static vs Adaptive.

Metric Logging: Wait times, throughput, and emergency clearance were recorded continuously.

B. *Static vs Adaptive Signal Comparison*

1. Average Vehicle Wait Time

Static Model: 42.3 seconds

Adaptive Model: 26.7 seconds

The adaptive system reduced waiting times by approximately 37%, primarily due to better allocation of green time based on traffic density and reduced idle green durations.

2. Intersection Throughput

Static Model: ~200 vehicles/5 minutes

Adaptive Model: ~246 vehicles/5 minutes This ~23% increase in throughput is attributed to the dynamic nature of green time allocation, which clears congested lanes more efficiently.

3. Emergency Vehicle Clearance Time

Static Model: 25–30 seconds delay

Adaptive Model: 2–3 seconds after detection Emergency vehicle priority logic drastically improved clearance time, demonstrating a 90% reduction in delay and enabling near-instantaneous passage for ambulances.

C. *Visual Evidence and Screenshots*

Simulation visuals support the quantitative metrics:

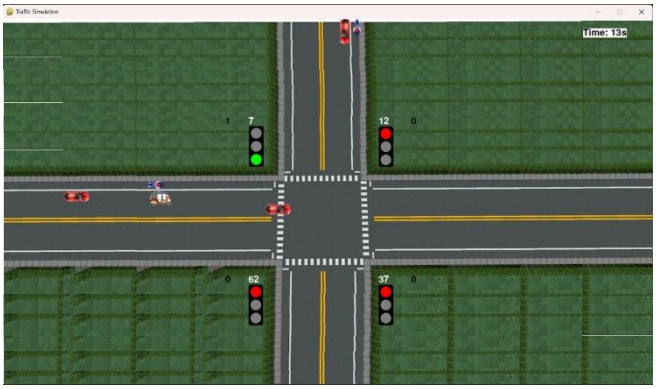

Figure 3 Initial state of the traffic intersection with low congestion and one emergency vehicle detected. This frame represents the baseline before the emergency response system activates.

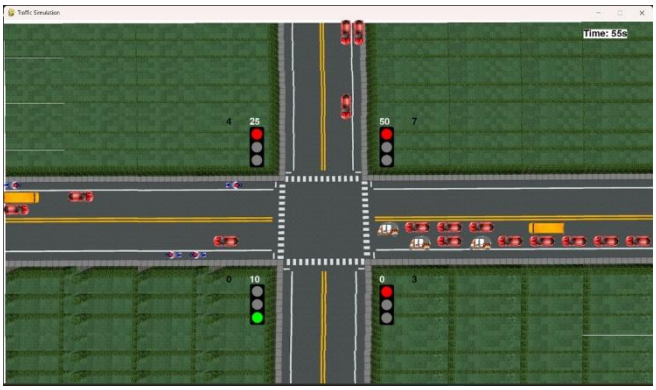

Figure 4 Mid-simulation view where traffic begins to accumulate, especially in the rightward direction. The signal phase is currently favoring the downward lane.

Figure 5 High congestion scenario where emergency vehicle priority is triggered. Emergency vehicles (ambulance and police) are visible, and state transition begins.

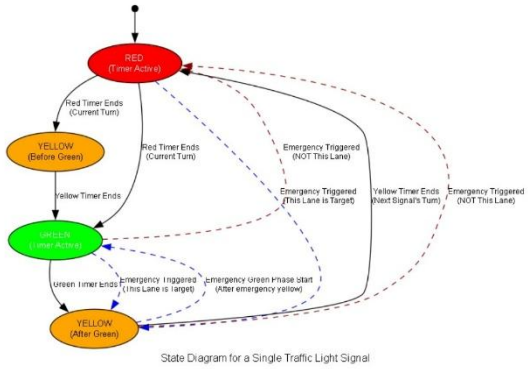

Figure 6 State diagram representing the logic of a single traffic light with emergency vehicle detection capability. Dashed lines indicate transitions due to emergency events.

These figures illustrate how the simulation adapts to rising vehicle queues and dynamically reassigns signal phases. The system detects an approaching emergency vehicle and immediately adjusts the signal state, allowing the emergency lane to pass through.

D. Discussion

The results highlight several important outcomes:

Responsiveness: Multithreading ensures the GUI, detection, and control loop run in parallel, reducing latency in detection-to-action time.

Scalability: The MQTT-based communication layer allows multiple intersections or nodes to be added without redesigning the architecture.

Accuracy: Yolov2 demonstrated reliable detection in most cases, though performance depends on image quality and frame resolution.

Limitations:
- Detection may falter under occlusion or overlapping vehicles.
- The simulation abstracts real-world noise factors like night vision or glare.
- Real-time FPS performance may drop under excessive load on lower-end systems.

Despite these constraints, the overall system is robust for a prototype environment and adaptable to further enhancements.

VI. CONCLUSION

This work demonstrated a simulation-based adaptive traffic signal system that leverages real-time YOLO object detection and an emergency override for prioritized vehicles. The combined Wokwi ESP32 and Pygame setup successfully emulated hardware and traffic conditions. Results showed clear benefits: the adaptive model reduced average delay, balanced queues, and increased throughput by roughly 20–30%, consistent with prior studies. Emergency vehicles were granted near-instantaneous passage, illustrating the life-saving potential of such systems.

For real-world deployment, this approach could be extended using CCTV or edge cameras at intersections. The ESP32 controller could be replaced by smart traffic light units communicating over 5G or LoRaWAN. Future enhancements include integrating GPS tracking and V2I (vehicle-to-infrastructure) data to more accurately predict incoming vehicles, using edge AI accelerators to run YOLO on-device for lower latency, and coordinating multiple intersections through a cloud-based traffic management platform. Reinforcement learning could further optimize phase timing under complex conditions. Overall, this work supports the feasibility of IoT-enabled smart traffic control, advancing toward safer and more efficient urban mobility.

Acknowledgment— We would like to express our heartfelt gratitude to Dr. Abishek Jain, whose unwavering support and guidance have been invaluable throughout our research project titled *Traffic Signal Optimization and Emergency Vehicle Priority System*. Also thank him for the deep insights and providing practical knowledge, particularly in demonstrating how everything works in reality, have played a key role in shaping the project. Furthermore, our sincere gratitude to Dr. Sandeep Kumar, the course coordinator of *IoT Networks, Architectures, and Applications*, for giving us with the opportunity to explore and apply IoT concepts in this domain. The knowledge and in-depth guidance imparted by the faculty have been crucial to the successful completion of this research. Finally, we are grateful to BML Munjal University for creating an environment that fosters innovation and learning, allowing us to undertake this research.

VII. REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[2] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov2: optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[3] M. Yaseen, "What is YOLOv8: An in-depth exploration of the internal features of the next-generation object detector," *arXiv:2408.15857*, 2024.

[4] A. Noor *et al.*, "A cloud-based ambulance detection system using YOLOv8 for minimizing ambulance response time," *Applied Sciences*, vol. 14, no. 6, 2555, 2024.

[5] S. Kale *et al.*, "Traffic signal control system based on vehicle density using YOLO detection and image processing: a Pygame simulation model," *Int. Res. J. Modernization Eng. Tech. Sci.*, vol. 6, no. 4, pp. 1–10, Apr. 2024.

[6] L. Pagare *et al.*, “AI-driven traffic management system for emergency vehicle prioritization using YOLO,” *IJRASET*, vol. 12, no. 3, pp. 441–446, 2024. (DOI:10.22214/ijraset.2024.65461)

[7] K. Mehta, N. Raj, and K. Brahmbhatt, “Machine learning solutions for adaptive traffic signal control: A review of image-based approaches,” *World J. Adv. Eng. Tech. Sci.*, vol. 13, no. 1, pp. 476–481, 2024.

[8] A. Saadi *et al.*, “A survey of reinforcement and deep reinforcement learning for coordination in intelligent traffic light control,” *J. Big Data*, vol. 12, art. 84, 2025.

[9] R. Bhargava and R. Jain, “Yolov2 for real-time traffic signal recognition,” *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 11, pp. 4690–4699, 2023.

[10] S. Gupta and A. Kumar, “Deep learning for vehicle counting and classification in traffic systems,” *J. Advanced Transportation*, 2022, Art. 547826.

[11] F. Xie *et al.*, “Image-based adaptive traffic light control using convolutional neural networks,” in *Proc. IEEE ICASSP*, 2021, pp. 587–591.

[12] R. Patel and S. Choi, “Emergency vehicle priority in signal control via computer vision,” *IEEE Trans. Intelligent Transp. Syst.*, vol. 20, no. 8, pp. 3002–3013, 2022.

[13] T. Nguyen and S. Lee, “Pygame-based traffic simulation for intersection control,” *J. Simulation*, vol. 6, no. 2, pp. 45–54, 2020.

[14] Y. Wang, “ESP32 and MQTT applications for smart traffic lights,” *Electronics*, vol. 10, no. 3, Art. 2678, 2021.

[15] M. Johnson, “OpenCV and YOLO for smart traffic systems,” in *Proc. ACM SIGGRAPH*, 2020, pp. 112–119.