VIETNAMESE - GERMAN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

AIoT for Smart Traffic Light Control in Central Ho Chi Minh City

Instructor: Dr. Vo Bich Hien

Final Report

Student 1: Nguyen Gia Phuong –10422113
Student 2: Le Duc Thanh Kim –10422105
Student 3: Dang Dinh Tu Anh –10422086
Student 4: Tran Nguyen Minh Tri –10422119

HO CHI MINH CITY, 10/2025

Approval Page

This project report is submitted to the Department of Computer Science, Faculty of Engineering, Vietnamese-German University, as part of the requirements for the course module **Project Course**. The report has been reviewed and approved by the undersigned.

Copyright

Authors: Nguyen Gia Phuong; Le Duc Thanh Kim; Tran Nguyen Minh Tri; Dang Dinh Tu Anh

Course Module: Project Course
Department: Computer Science, Faculty of Engineering
Institution: Vietnamese-German University
Location: Ho Chi Minh City, Vietnam

All source code is published here. It is available and public under MIT license for research purposes and further development.

https://github.com/ppngg/ESP32Traffic.git

Motorbike Computer Vision Dataset (Roboflow):
https://universe.roboflow.com/carcounting-wcidm/motorbike-rcwkd

System Demo & Field Testing Videos (Google Drive):
https://drive.google.com/file/d/1d7DUd_DvWiB8wctv14AtKub2RzeOprS7/view?usp=drivesdk

Our fine-tuned model on collective dataset:
https://drive.google.com/file/d/1zyIHK2BcL_hNi34KyIryjksd4vGLv1GC/view?usp=drive_link

Ho Chi Minh City, October 2025

Acknowledgements

We would like to express our sincere gratitude to our supervisor, **Dr. Vo Bich Hien**, for his valuable guidance, constructive feedback, and continuous support throughout this project. His expertise and direction greatly helped us shape the research approach and improve the quality of this report.

We also thank the **Department of Computer Science, Faculty of Engineering, Vietnamese-German University (VGU)** for providing an academic environment, resources, and opportunities that supported our work.

Finally, we would like to thank our classmates, friends, and families for their encouragement and motivation during the project period. Their support helped us stay focused and complete this report.

The Authors

Content

List of Figures

List of Tables

1 Introduction

1.1 Background

Ho Chi Minh City (HCMC) is known as the economic center of Vietnam and one of the most active cities in the country. It contributes a large part to Vietnam's GDP and attracts millions of people for work and study. However, this fast development has also created many serious problems, especially in transportation. By early 2026, the number of registered vehicles in the city had exceeded 9.5 million (Quốc Hùng & Yên, 2024), which puts huge pressure on the existing road system. In addition,HCMC has a special type of "mixed traffic," where motorcycles are the main way people travel and make up more than 90% of total traffic volume (Sở Giao thông Vận tải TP.HCM, 2024; VietNamNet, 2024). This enormous volume of vehicles puts pressure on a road network that was not designed to handle such high traffic volumes.

Though the Vietnamese government has tried a lot to improve public transport, the road infrastructure has not grown fast enough to handle the increasing number of vehicles. Road density remains very low, leading to frequent traffic jams during peak hours. By the end of 2025, traffic congestion in the city had increased by about 24% compared to the previous year, highlighting the failure of static management systems to handle peak demand (Cẩm Tuyết, 2025; Sở Giao thông Vận tải TP.HCM, 2025). This situation not only wastes time and causes economic losses, but also makes daily life more stressful for citizens. Therefore, there is a clear need for better and smarter traffic management solutions, especially those using AI technology, to help control traffic flow and improve transportation efficiency in HCMC.

1.2 Problem Statement

Ho Chi Minh City has grown very fast in recent years, and this has caused many serious problems for the city's traffic system (World Bank, 2023). Although the city is now large and very crowded, traffic management still uses many old and ineffective methods (Tuấn Kiệt, 2025). In many important areas, traffic lights work with fixed time cycles and do not change based on the real number of vehicles on the road (Cẩm Tuyết, 2025). Furthermore, traditional monitoring systems struggle to accurately detect and manage the complex mixed-traffic flow dominant in the city (Mandal & Adu-Gyamfi, 2020; Tuấn Kiệt, 2025). As a result, by the end of 2025, traffic congestion increased by 24%, showing that the city urgently needs smarter, AI-based traffic solutions (Cẩm Tuyết, 2025; Sở Giao thông Vận tải TP.HCM, 2025).

1.3 Objective

The primary goal of this project is to develop a cost-effective system, called "Smart Light Traffic",designed to dynamically optimize traffic signal intervals by prioritizing high-density lanes for extended green light durations.To achieve this, the project focuses on the following specific objectives:

- **Deploy Edge-AI for Density**: To implement a machine learning model You Only Look Once (YOLO) on the ESP32-S3 to detect motorbikes in real-time. The main goal of this project is to use image processing to understand traffic better. Instead of only counting the number of vehicles, a custom algorithm called Horizontal Fill (HFill) is used to estimate how much of the road is actually occupied. This method is more suitable for extremely congested traffic conditions, where vehicles are close together and difficult to count individually.

- **Propose a Formula-Based Adaptive Control**: The primary goal of this work is to create a simple method to control traffic signals automatically. By using the Basic Stability Condition and proportional logic, the system can decide how long the green light should stay on. This decision is based on real-time traffic data, such as how many vehicles are arriving at the intersection (q) and how many vehicles can pass through the road at full capacity (s). By using this method, traffic lights can change in a smarter way based on real traffic, instead of fixed time. This can help reduce waiting time and make driving less stressful for people.
- **Implement Cloud-IoT Communication Infrastructure**: The goal of this part is to build a simple and reliable communication system using the MQTT protocol and AWS IoT Core (Hunkeler et al., 2008). This system helps edge devices, such as traffic light controllers, send and receive data with the cloud in an efficient way using a publish–subscribe model. MQTT is chosen because it uses very little bandwidth and low power, which makes it suitable for real-time communication in IoT systems with limited resources (Al-Fuqaha et al., 2015). Additionally, AWS IoT Core is used to keep the data transfer safe and stable. It will connect the local traffic light systems to one main monitoring dashboard. Because of this, the traffic data can be sent and updated all the time without many problems.
- **Validate Performance via Empirical Benchmarking**: The goal of this part is to show that the proposed AI-based system can improve traffic performance. The system is expected to increase the traffic clearance rate by about 25%. To show this improvement, the results from the AI traffic control system will be compared with real traffic data collected from busy roads in Ho Chi Minh City, such as Dien Bien Phu, Truong Dinh and Nguyen Van Troi road.

1.4 Scopes

The scope of this project is defined by technical, functional, and environmental boundaries to ensure the feasibility of the AIoT prototype for Ho Chi Minh City's unique traffic conditions.
A. In-Scope (Technical & Functional)

- **Optimal Environmental Conditions**: The system is designed to work in the morning time because at that time, the light is better. In the morning, the camera can see more clearly, which helps the system detect traffic more easily and reduce errors.
- **Target Road Topology**: Technical implementation is focused on one-way urban corridors where vehicle occlusion is minimal, ensuring higher reliability for the HFill algorithm.
- **Real-World Testing Environment**: Field testing is conducted on actual public roads, specifically on Dien Bien Phu, Truong Dinh and Nguyen Van Troi road., to validate the system under live traffic conditions.
- **Operational Compliance**: All testing and deployment procedures are executed strictly to ensure they do not violate general traffic rules or impede public safety.
- **Outdoor Deployment Validation**: The system's hardware and software are tested in outdoor environments to evaluate stability, heat management, and data synchronization with AWS IoT Core.

B. Out-of-Scope

- **Industrial Hardware**: Deployment of high-performance GPU servers or industrial-grade enclosures for extreme weather conditions is excluded.
- **Legal and Regulatory Permits**: Obtaining government authorization or official legal permits for physical installation on urban light poles is beyond the current scope.
- **Vehicle Classification**: Detailed identification of specific vehicle brands or license plate recognition is not required, as the focus remains solely on density approximation (Mandal & Adu-Gyamfi, 2020).

2 Literature Review

This section explores the significant academic and technical foundations that justify the transition from static traffic management to an AI-driven, decentralized architecture in motorcycle-dominant urban centers.

2.1 Mixed Traffic Flow in Motorcycle-Dominant Cities

Traditional traffic models are mainly made for cars, so they do not work well in Ho Chi Minh City, where motorcycles make up 93% of all traffic (Vu & Nguyen, 2015). Because of this, lane-based traffic analysis is not suitable when there are too many motorcycles on the road (Vu & Nguyen, 2015). A similar situation is observed in large cities in Indonesia (such as Jakarta and Bandung), where motorcycles account for 60%–71% of the vehicle population. In this city, motorcycles often move between cars, which creates traffic flow higher than what car-based models can predict (Sutandi, 2017). In Bangkok, the large number of motorcycles causes irregular road use, making fixed-time traffic lights very inefficient during peak hours (Rongviriyapanich & Suppattrakul, 2005). Studies in Taipei also show that motorcycles do not move one by one like cars, but instead travel together in groups (Hsu et al., 2003). These studies from different cities confirm that measuring road density is more appropriate than counting individual vehicles in motorcycle-dominated traffic conditions such as Ho Chi Minh City (Hsu et al., 2003).

2.2 Comparative Analysis of Vehicle Counting Algorithms

Deep learning has significantly improved video-based vehicle counting in recent years. In Metro Manila, manual "car counters" are still utilized, but these methods are labor-intensive and lack real-time responsiveness (Agence France-Presse (AFP), 2024). Research by (Mandal & Adu-Gyamfi, 2020) evaluated detectors such as CenterNet, Detectron2, YOLOv4, and EfficientDet. These systems rely on tracking algorithms including Intersection over Union (IoU), Simple Online and Realtime Tracking (SORT), and Deep SORT. While accurate, these models face significant challenges in occluded or low-light environments (Mandal & Adu-Gyamfi, 2020). Furthermore, deep learning computation requires high-end GPUs that are financially expensive and consume massive amounts of energy. Hardware like the NVIDIA GTX 1080Ti requires up to 36 hours for training and possesses thermal requirements unfit for decentralized light pole deployment (Neil C. Thompson et al., 2020). The NVIDIA Jetson Nano has been explored as a low-cost alternative for edge-based computer vision. However, the Jetson Nano still encounters power and heat issues in standalone outdoor installations (Othman et al., 2025). Therefore, this project optimizes traffic algorithms for the ESP32-S3 to meet extreme resource and cost constraints.

2.3 Density Estimation and Discrete Counting

Traditional counting-by-detection methods fail in Ho Chi Minh City because high-density motorcycle occlusion makes separating individual bounding boxes nearly impossible (Vu & Nguyen, 2015). Mandal & Adu-Gyamfi (2020) observed that existing density models often perform poorly in wide-angle perspectives and lack the tracking capabilities necessary for traffic flow analysis. Furthermore, high-performance discrete detectors require expensive, high-power GPUs (Neil C. Thompson et al., 2020) that are not feasible for decentralized edge deployment on urban light poles. Consequently, density estimation serves as a superior alternative by measuring total "road pressure" rather than identifying individual vehicles. This method is specifically optimized for the saturated motorcycle flows where traditional lane-based analysis is ineffective

(Vu & Nguyen, 2015). By focusing on occupancy metrics, the system can give traffic information in real time for adaptive signal control without the massive computational costs of deep learning detectors (Neil C. Thompson et al., 2020).

2.4 Adaptive Signal Control and Stability

The math used to improve traffic light timing comes from the Webster Model, which helps decide the optimum signal cycle length to reduce vehicle waiting time (Webster, 1958). To make sure the system become unstable, it follows the Basic Stability Condition, where traffic demand must not exceed the intersection's saturation capacity to avoid persistent queue growth (Vu & Nguyen, 2015). The system gives priority to directions with higher road pressure, but it still keeps minimum safety times for traffic lights. By doing this, the traffic flow can stay more stable even when the number of motorcycles changes a lot, which is very common in Ho Chi Minh City (Vu & Nguyen, 2015). This strategy prevents the failures of fixed-time signal systems by keeping intersection operation within stable limits at all times.

3 Methodology

3.1 Traffic light green cycle calculation

For the primary purpose of this project, our group focuses entirely on estimating the traffic light green dynamically. To instruct an estimation workflow, we follow up the standard signal capacity relationship, where **the green ratio** must be larger or equal than required demand. This leads to following this condition:

$$\frac{g}{C} \geq \frac{q}{s} \tag{1}$$

Where:

- g **(Green time)** Duration of the green light for a specific phase in seconds.
- C **(Cycle length)** Total time of a complete signal cycle.
- q **(Demand flow)** Traffic arrival rate at the intersection during a phase.
- s **(Saturation flow rate)** Maximum capacity of lanes per hour under continuous green light.

which q is defined as:

$$q = k \cdot v \tag{2}$$

with k defined as Density, a concentration of traffic on a road segment.

3.2 Designed Finite State Machine

In this project, our group will adjust control parameters based on separate traffic conditions. For describing these changes, Figure 1 depicts an action of ones. In the state *S1* (normal flow), our group sets saturation flow as base level **1900** (Vu & Nguyen, 2015), average speed **30km/h**, and short cycle **90**. When density is over 20%, the system switches to state *S2* (congestion flow), where it applies a lower saturation flow **1500**, reduced average **20km/h** but with a higher cycle base for getting more green time.

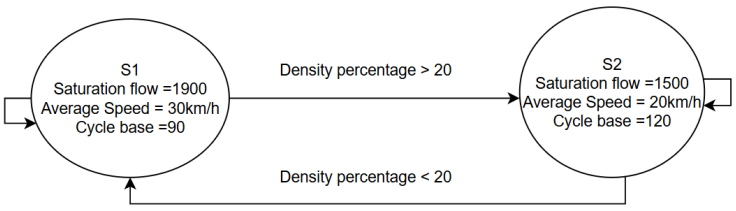

Figure 1: Finite State Machine work based on different situation on road

3.3 Discussion on problems for implementing in reality

However,the formula Equation 2 will depend strongly on k (density) for computing the green cycle. In practice, estimating k would require reliable vehicle detection and tracking from the roadway. With our current proposal, that is not feasible because of two listed reasons:

1. Traffic in Ho Chi Minh City is heterogeneous and dense, causing unusual occlusions and overlapping objects that make vehicle-by-vehicle counting unreliable.
2. Accurate vision-based counting typically relies on deep learning models, which require high modern computational device GPU and energy consumption for inference and training (Neil C. Thompson et al., 2020) .These challenges are out of reach on edge computing with small size and capacity for computing deep accurate learning models.

To overcome these problems, our group introduces an unconventional post-processing algorithm for estimating the percentage of objects in images in Figure 19. After this step on the edge device, the final result will return a particular percentage which we treat as a normalized congestion index. From the local report on traffic at the end of 2024, Ho Chi Minh City's roads experienced an average car density of 236 vehicles per kilometer (VietNamNet, 2024). The same report published shows the proportion of motorbikes takes up to 93% (Vu & Nguyen, 2015). Based on this reference, we set 219 vehicles/km as the 100% level and estimate local density as:

$$k_{\text{est}} = \frac{p}{100} \times 219 \tag{3}$$

where:

k_{est}	Estimated density (vehicles/km)
p	Density percentage from the output of the edge unit (%)

4 System Design

This project adopts a distributed Artificial Internet of things (AIoT) architecture, where sensing and controlling at the edge and coordinate via cloud services. For each intersection ESP32-SEEED Camera runs an inference to produce a vehicle density percentage. This result will be sent directly to the AWS IoT Core service on the topic name `esp32/traffic/density_now`. An ESP32 controller will process this message through the task AWS loop controller and update the LED timer.

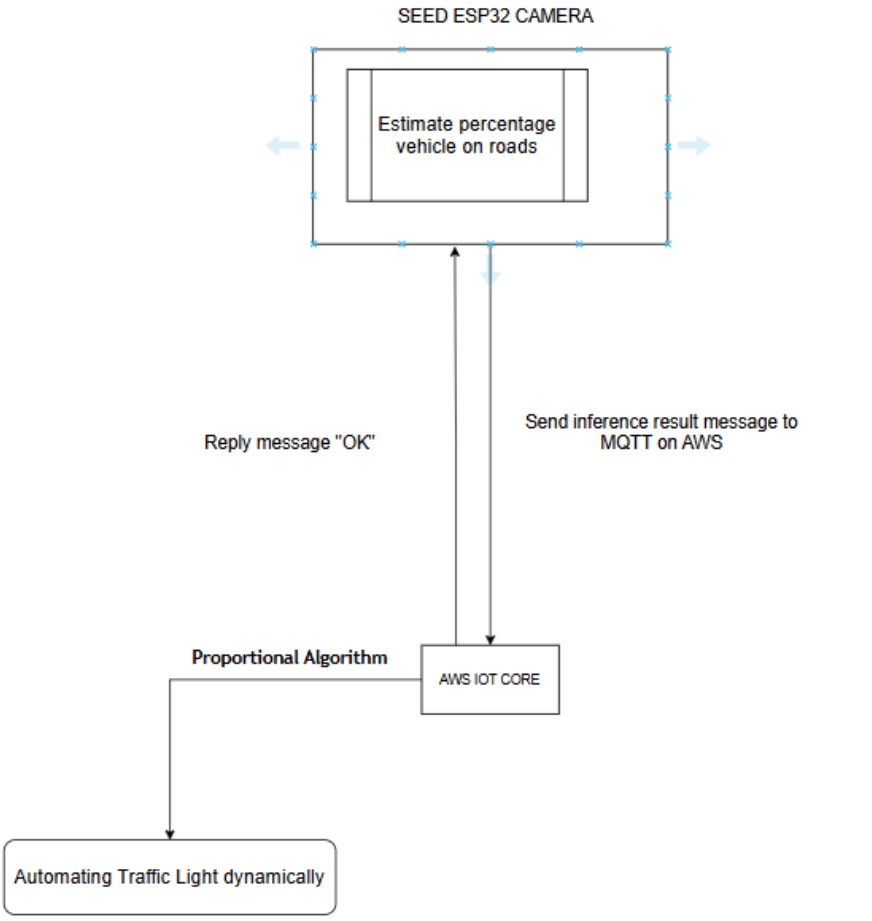

Figure 2: This diagram for illustrating how the system cooperate

5 Result Analysis and Dicussion

5.1 Dataset preparation and collecting method

5.1.1 Dataset preparation

To train and evaluate the vision-based vehicle detection model, we collected a real-world traffic image dataset using a Seeed Studio XIAO ESP32-S3 Sense board with an OV2640 camera module. In this one our group collect data during 3 days on Nguyen Van Troi and Au Co street during a day.

Figure 3: Seeed Studio XIAO ESP32S3 Sense board with an OV2640 camera module.

5.1.2 Dataset Overview and Labeling process

As our group adjusted this camera to take a picture per minute, during the 8 hours of data collection, there were 766 images in total with a 240x240 resolution. To avoid overfitting of the model and increase its generalization, we split this data into a train and test set with an 80/20 ratio. In addition, our group utilized the Roboflow platform for manually labeling these captured images

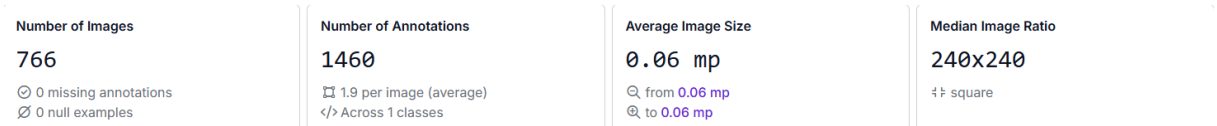

Figure 4: Dataset overview included 766 for training with 240x240 pixels and in total over 1400 annotated images.

5.2 Model Training and Result

In this section, our group will present our quantized and fine-tuned model on our collected dataset.Our selected model is Yolo-Swift. A model has been introduced by Seeed Studio built

entirely on Yolo-V5 backbone (Seeed Studio, 2022) and served mostly on running inference AI model on small-ram-on-chip devices.

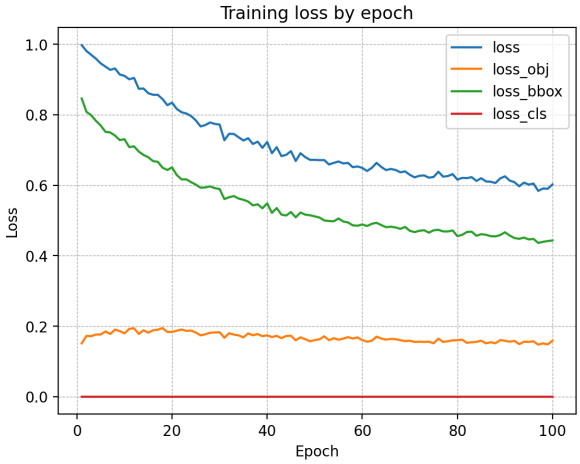

Figure 6: Training loss during training 100 epochs

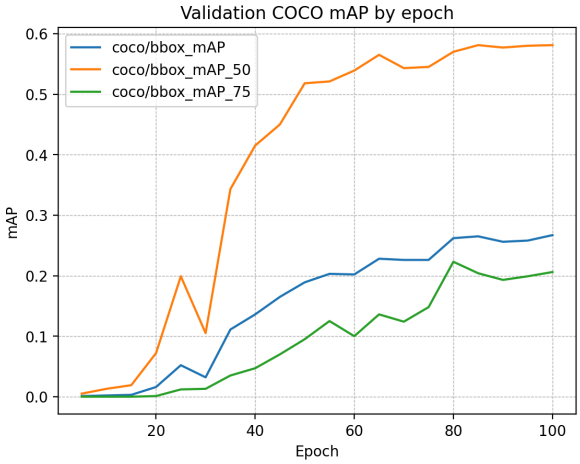

Figure 7: Validation on test set during training

In general, stable convergence and increasing performance are shown in Figure 6, Figure 7. Better localization is shown by a steady decrease in training loss, primarily due to a reduction in bounding-box loss. Validation mAP grows and then plateaus, with $mAP@0.50 \approx 0.58$ significantly higher than $mAP@0.75 \approx 0.20$.

5.3 Traffic Light Timer and our comparison

5.3.1 Background and Motivation

To validate our proposed methodology, we also investigated our traffic lights to reality at different times of day in 2 well-known destinations. Firstly, our group picked Nguyen Van Troi - Nguyen Trong Tuyen junction, a key connection corridor linking District 1 - the heart of economics and finance in Ho Chi Minh City to the Cong Hoa gateway-a famous location for traffic congestion at the end of the workday.

Figure 9: Nguyen Van Troi road (satellite view).

Figure 10: Nguyen Van Troi road (our camera).

Figure 8: Comparison between satellite and on-site views of Nguyen Van Troi road.

Secondly, we also did a survey of the famous road of inner-Ho Chi Minh city Truong Dinh-Dien Bien Phu connecting the central district and commercial areas. This traffic congestion usually happens in this area because of the large volume of vehicles every day.

Figure 12: Dien Bien Phu road

Figure 13: Truong Dinh road

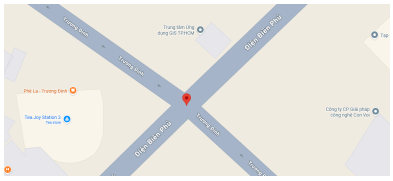

Figure 14: Intersection (Google Maps)

Figure 11: Left: on-site capture; middle: on-site capture; right: Google Maps view of the intersection.

5.3.2 Analysis Result

Cycle / Phase	Nguyen Van Troi road	Our proposal light
Green Cycle	45s	30s
Yellow Cycle	3s	3s
Red Cycle	27s	47s

Table 1: Traffic light comparison on Nguyen Van Troi road at 16h30

Cycle / Phase	Dien Bien Phu road	Truong Dinh road	Our proposal light
Green Cycle	30s	35s	52s
Yellow Cycle	3s	3s	3s
Red Cycle	40s	35s	25s

Table 2: Traffic light comparison on Truong Dinh and Dien Bien Phu road at 12h40 am

From result of Table 1 and Table 2, an increase trend for a lane has more traffic density. Particularly, at 12:40 am on Dien Bien Phu road, manual observation indicated increased vehicle volume, and the proposed schedule correspondingly extends the green phase to improve traffic flow. Our green light proposal is increase over 20s comparing traffic light at Dien Bien Phu street, which leads to a decrease in red time for phrase. In contrast, on Nguyen Van Troi road at 16:30, our group observed that inbound flow toward the inner city was lower at that moment, so the proposed timing slightly reduces the green phase from 45s to 30s and increases the red phase from 27s to 47s.

5.3.3 Testing and Difficulties during the implementation phase

As show in Table 3 and Table 4, our group just applied a device with limit on computing power and storage on RAM with effective price. That is a reason why our fine-tuned model faced a big trouble during deploying in a dense-complex traffic on reality. Our group will show some test results and failure during implementation phrase.

Figure 16: High occlusion

Figure 17: Low image quality and resolution

Figure 18: Model hallucination for detecting object with high loss

Figure 15: Limitations of the vehicle detection model: low-resolution inputs and heavy occlusion reduce detection reliability, and web-based testing can produce hallucinated predictions (false bounding boxes on non-vehicle regions)

5.4 Post-processing algorithm for this difficulties

Because of above limitations,our system cannot reliably detect and count motorbikes individually, so per-vehicle counting is impractical. Instead, we estimate congestion by measuring how much of the region of interest (ROI) is covered by detected bounding boxes.This produces a single normalized traffic-density indicator.

This section presents the post-processing pipeline that converts each raw camera frame into one scalar value representing traffic density. Rather than using explicit vehicle counts, the pipeline estimates horizontal road occupancy by motorbikes, which is better suited to side-view camera setups where occlusion is frequent. The design prioritizes robustness and computational efficiency, enabling real-time operation on resource-constrained embedded hardware.

Input: Detections $\mathcal{D}$, ROI $\left(x_1^R, y_1^R, x_2^R, y_2^R\right)$, threshold τ, top-K
Output: $h \in [0,1]$ and optional count $|\mathcal{I}|$

1. Sort $\mathcal{D}$ by score s in descending order
2. $\mathcal{I} \leftarrow \emptyset$
3. **for** each detection $b = (x_1, y_1, x_2, y_2, s)$ in $\mathcal{D}$ **do**
    1. **if** $s < \tau$ **or** $|\mathcal{I}| \geq K$ **then**
        1. **break**
    2. $x_{1'} \leftarrow \max\left(x_1, x_1^R\right), x_{2'} \leftarrow \min\left(x_2, x_2^R\right)$
    3. $y_{1'} \leftarrow \max\left(y_1, y_1^R\right), y_{2'} \leftarrow \min\left(y_2, y_2^R\right)$
    4. **if** $x_{2'} \leq x_{1'}$ **or** $y_{2'} \leq y_{1'}$ **then**
        1. **continue**
    5. Append interval $[x_{1'}, x_{2'}]$ to $\mathcal{I}$
4. **if** $\mathcal{I} = \emptyset$ **then return** 0
5. Sort intervals in $\mathcal{I}$ by left endpoint (x_1) ascending
6. $L \leftarrow 0$; $(a, b) \leftarrow$ first interval in $\mathcal{I}$
7. **for** each next interval (c, d) in $\mathcal{I}$ **do**
    1. **if** $c \leq b$ **then**
        1. $b \leftarrow \max(b, d)$
    2. **else**
        1. $L \leftarrow L + (b - a)$
        2. $(a, b) \leftarrow (c, d)$
8. $L \leftarrow L + (b - a)$
9. $W \leftarrow x_2^R - x_1^R$
10. **if** $W \leq 1$ **then return** 0
11. **return** $\min\left(1, \max\left(0, \frac{L}{W}\right)\right)$

Figure 19: An unconventional post-processing algorithm named as Horizontal Fill (`compute_hfill`)

The Horizontal Fill (hfill) metric is an unconventional post-processing step designed to estimate road occupancy from side-view camera frames. This method is used to give an approximate percentage of how much the motorbikes are occupying the ROI while avoiding redundant counting caused by overlapping bounding boxes or false positives. After capturing 320×240 RGB565 images and performing center-cropping and quantization for the SWIFT-YOLO model, the system extracts motorbike detections.

Given a set of detections $\mathcal{D}$, where each detection consists of a bounding box and an associated confidence score. The set is then sorted by confidence score in ascending order, ensuring reliability. Detections with scores below the defined threshold ($\tau = 0.65$) are discarded and only top-K ($K = 10$) detections are included, making sure only high-quality detections are considered in the final computation and allow the embedded hardware to operate with acceptable latency. For each retained detection, we compute the intersection between the detection box and the ROI. Detections that do not intersect the ROI are excluded. For intersecting detections, only the horizontal extent of the overlap is preserved in the form of an x-axis interval. For intersecting detections, only the horizontal span of the overlap matters for our calculation of interval [x1', x2']. This projection reduces the two-dimensional overlap problem to a one-dimensional interval coverage problem, which is sufficient when horizontal occupancy within the ROI is our primary interest. The resulting set of x-axis intervals is then sorted by their left endpoints and merged to eliminate overlapping or adjacent segments. This merging step is

critical to prevent double-counting regions covered by multiple detections. The total horizontal coverage length L is obtained by summing the lengths of the merged intervals. Then, we sort our set of x-axis intervals by left endpoint and merge them together. This helps eliminate double-counting overlapping detections. The final hfill value is the ratio of this union length to the total ROI width, providing a normalized congestion measure ($hin[0, 1]$) that is robust to perspective distortion and camera distance.

The Figure 20 showcase the use of our Horizontal Fill algorithm in a real-life scenario. The green box depicts the Region of Interest which is pre-configured, while the red box represents the percentage of motorbikes covering the predefined region. The red region is the result of summing up all the detected motorbike bounding boxes and projecting that onto the x-axis.

Figure 20: Visualization of Horizontal Fill algorithm from images captured on ESP32S3

We also acknowledge that our algorithm poses some serious limitations. The Region of Interest requires manual configurations during deployment because factors such as the camera angle, the road structure affect how the ROI should be set, resulting in a severe time consumption and waste of human resources. Furthermore, how the objects appear on the camera view can significantly affect the final output of the algorithm as objects that appear closer to the camera will account for a larger covered proportion of the ROI. In the Figure 21 and Figure 22, one motorbike can cover 32.3% of the ROI, while another motorbike at the same time only occupies 26.5% of the road. This inconsistency makes it difficult to estimate the true density of the road, consequently affecting how the traffic light control algorithm will operate during deployment. Although this issue can be partially mitigated through proper camera setup, better road structure that is suitable for the algorithm, improved ROI configuration, doing so requires additional resources that are beyond the scope of this project. There are also common errors with object detection systems that we encountered during testing such as false detections, multiple overlapping bounding boxes which in turn also affect the resulted percentage of traffic density

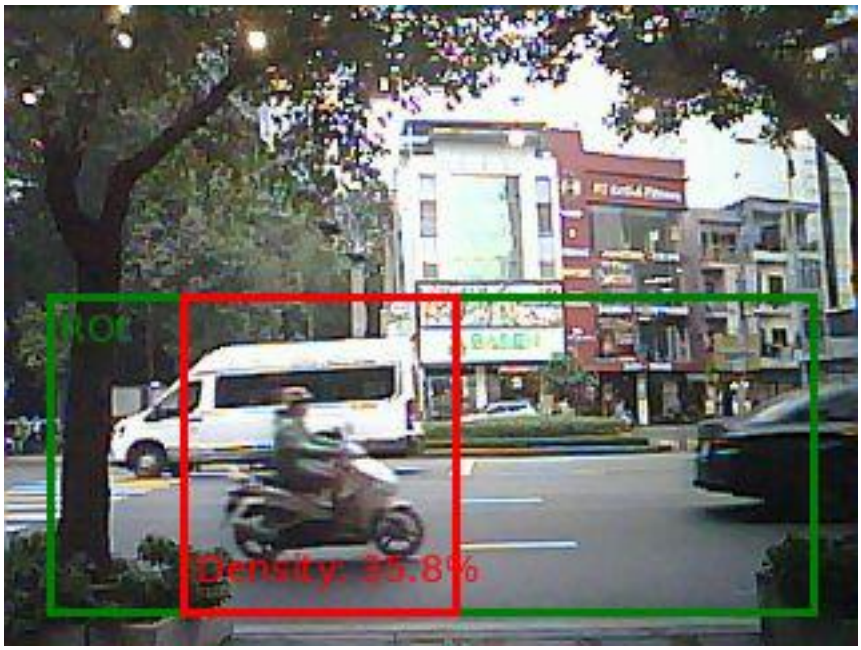

Figure 21: A motorbike taking up 32.3% of the Region of Interest due to overlapping bounding boxes, which inflate the final horizontal coverage used for traffic estimation

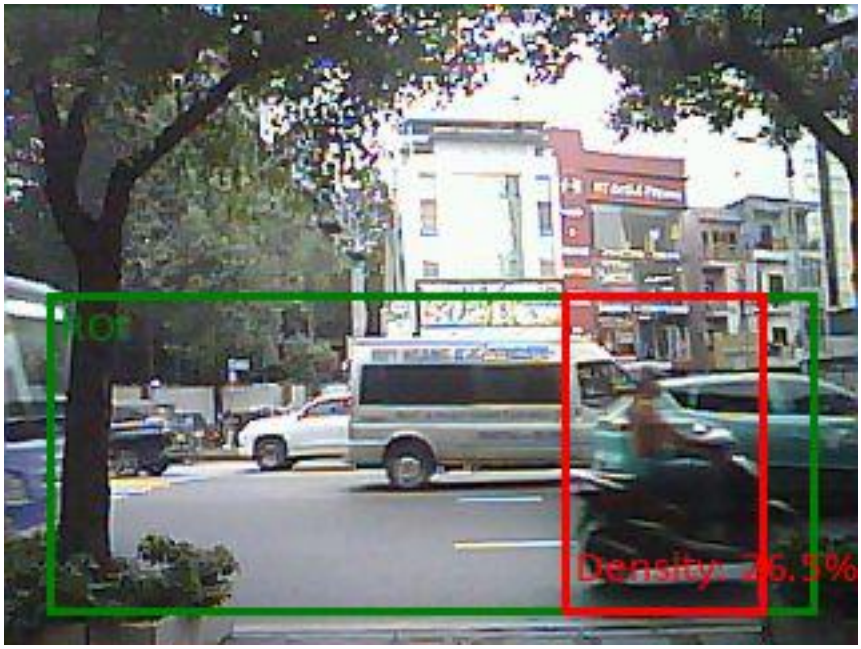

Figure 22: Another case showcasing the perspective-induced inconsistency. The motorbike only accounts for 26.5% of coverage as the bounding box matches up with the object's size, leading to a more accurate percentage of traffic density

5.4.1 Motivation and Background

Conventional traffic estimation methods often rely on counting the amount of bounding boxes. However, in side-view camera configurations, these approaches are sensitive to occlusion and perspective effects. In contrast, horizontal road occupancy provides a more direct and interpretable measure of congestion. At the same time, side-view setup offers a more accessible method to traffic monitoring while traditional top-down view requires complex installation and expensive costs. Prior studies have shown that combining region-of-interest selection with occupancy-based metrics yields more stable traffic density estimates than full-frame or count-

based approaches, particularly under occlusion and perspective distortion (Fachri et al., 2023; Trần & Nguyễn, 2024).

5.4.2 Design Considerations and Advantages

The proposed pipeline is suitable for embedded deployment due to its low computational and memory overhead. Its dominant operation is sorting, resulting in an overall $O(n \log n)$ complexity, which is widely regarded as efficient for real-time embedded vision and intelligent transportation systems (Gonzalez & Woods, 2018; Koesdwiady et al., 2016). Memory usage remains minimal, as only a bounded number of detections and one-dimensional intervals are stored, consistent with ROI-based processing strategies commonly used in resource-constrained vision systems (Gonzalez & Woods, 2018).

Robustness is achieved through layered filtering and aggregation mechanisms, including ROI restriction, confidence threshold, top-K selection, burst sampling, and median aggregation. Such techniques are well established for reducing noise, suppressing false detections, and stabilizing outputs in real-world traffic scenes with occlusion and variable lighting (Dollar et al., 2012). Median-based aggregation in particular is known to be effective against outliers in noisy measurement streams (Gonzalez & Woods, 2018).

Compared to vehicle counting or bounding-box area summation, which are sensitive to occlusion, scale variation, and perspective distortion (Gomaa et al., 2019), region-level aggregated metrics provide more stable and interpretable estimates of traffic conditions, especially in side-view camera setups (Guo et al., 2021). This makes the horizontal fill metric a practical and reliable choice for density estimation under real-world deployment constraints.

6 Conclusion

The project introduces an affordable AIoT-based traffic management system that developed and designed for the motorcycle-dominated urban environment of Ho Chi Minh City. Our system has the ability to estimate traffic density in real-time despite hardware limitations by implementing the YOLO model on ESP32-S3 hardware and introducing the novel Horizontal Fill algorithm. To validate the system's ability to dynamically adjust traffic light cycles based on actual road conditions, we conducted field testing on Nguyen Van Troi, Dien Bien Phu, and Truong Dinh roads. The results showed that green light durations on our simulated Traffic Light Prototype adapted appropriately to the measured traffic density. The integration of MQTT protocol with AWS IoT Core enabled reliable cloud communication, allowing for centralized monitoring and control of distributed traffic nodes. While the prototype faced challenges related to image quality, occlusion, and environmental sensitivity, it proved the fundamental viability of edge-based AI for adaptive traffic control in resource-constrained settings. The system's modular architecture and low deployment cost of approximately 2.15 million VND per intersection make it a practical starting point for scaling intelligent traffic management across Vietnam's rapidly growing cities. Overall, this work demonstrates that even with limited computational resources, AI system can meaningfully address urban congestion challenges in developing nations.

7 Limitations and Future work

7.1 Limitations

Although the project shows a working prototype for smart traffic management, there are still many technical problems that limit its performance:

- **Suboptimal Hardware Infrastructure**: The system uses the ESP32-S3 micro-controller, which is not strong enough to run complex AI tasks. Its processing speed is quite low, so the system cannot work well when doing real-time AI analysis. In internal tests, the model performance was considered "poor" when running directly on the device.
- **Low Image Fidelity and Sharpness**: Because of hardware and memory limits, the camera only captures images with a low resolution of 240×240 pixels. These images are not sharp, so it is hard for the AI model to clearly see and separate individual vehicles. This is a big problem in Ho Chi Minh City, where traffic is very dense.
- **Occlusion and Overlap Errors**: Many motorcycles often overlap and block each other in the image. Because the images are blurry, the model cannot handle this situation well, which leads to wrong vehicle counting and traffic density estimation (Mandal & Adu-Gyamfi, 2020).
- **Environmental Sensitivity**: The vision model is currently optimized only for morning hours with stable natural light . It lacks the robustness required for night-time operation or during heavy tropical rainfall, where visibility is significantly reduced (Mandal & Adu-Gyamfi, 2020).

7.2 Future Work

To transition this prototype into a high-performance urban solution, the following upgrades are proposed:

- **Hardware and Optical Upgrades**: We plan to migrate the system to a more powerful edge platform (such as a Raspberry Pi or NVIDIA Jetson) to support high-definition optical sensors (Mandal & Adu-Gyamfi, 2020). This will allow for sharper images and more precise vehicle tracking
- **Advanced Model Refinement**: Future work will involve training more robust models, such as YOLOv8 or YOLOv10, using a larger and more diverse dataset to improve detection accuracy under various lighting conditions (Mandal & Adu-Gyamfi, 2020).
- **Algorithm Optimization**: We intend to refine the HFill algorithm to better handle high-density saturation, ensuring that the estimated density more closely matches the reality of motorcycle-dominant traffic flows (Vu & Nguyen, 2015).
- **Expanded Operational Scope**: The project will be extended to support multi-lane intersections and night-time traffic management by integrating infrared sensors and low-light detection algorithms (Mandal & Adu-Gyamfi, 2020).
- **Smart Network Integration**: Using the established MQTT and AWS IoT Core pipeline, we aim to connect multiple intersection nodes, allowing for a synchronized "Green Wave" flow that can reduce wait times across the entire city grid

References

Agence France-Presse (AFP). (2024, October 18). *Car counters in Metro Manila help address 'world's worst traffic'*. Inquirer.net. https://newsinfo.inquirer.net/1994042/car-counters-in-metro-manila-help-address-worlds-worst-traffic

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, *17*(4).

Cẩm Tuyết. (2025, January 4). *TPHCM: Ùn Tắc Giao Thông Tăng 24% so Với Cùng Kỳ Năm 2024*. Báo Sài Gòn Giải Phóng. https://www.sggp.org.vn/tphcm-un-tac-giao-thong-tang-24-so-voi-cung-ky-nam-2024-post826941.html

Dollar, P., Wojek, C., Schiele, B., & Perona, P. (2012). Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(4), 743–761.

Fachri, M., Hikmah, N., & Chusna, N. Lu'lu ul. (2023). Vehicle Counter in Traffic Using Pixel Area Method with Multi-Region of Interest. *BIRCI Journal*.

Gomaa, A., Abdelwahab, M. M., Abo-Zahhad, M., Minematsu, T., & Taniguchi, R. (2019). Robust Vehicle Detection and Counting Algorithm Employing a Convolution Neural Network and Optical Flow. *Sensors*, *19*(20), 4588.

Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th ed.). Pearson.

Guo, H., Yao, S., Yang, Z., Zhou, Q., & Nahrstedt, K. (2021). CrossROI: Cross-camera region of interest optimization for efficient real time video analytics at scale. *Arxiv (Cornell University)*. https://doi.org/10.48550/arxiv.2105.06524

Hsu, T.-P., Mohd Sadullah, A. F., & Nguyen, X. D. (2003). *A Comparative Study on Motorcycle Traffic Development of Taiwan, Malaysia and Vietnam* (Vol. 5).

Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008). *MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks*. 3rd International Conference on Communication Systems Software and Middleware (COMSWARE).

Koesdwiady, A., Soua, R., Karray, F., & Kamel, M. S. (2016). Recent trends in driver safety monitoring systems: State of the art and challenges. *Machine Vision and Applications*, *28*, 1–21.

Krizhevsky, Alex, Sutskever, Ilya, Hinton, & Geoffrey E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems.

Mandal, V., & Adu-Gyamfi, Y. (2020). Object detection and tracking algorithms for vehicle counting: A comparative analysis. *Arxiv*.

Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, & Gabriel F. Manso. (2020). *The Computational Limits of Deep Learning*. https://arxiv.org/abs/2007.05558

Othman, N., Saleh, Z., & Ibrahim, B. (2025). *A Low-Cost Embedded Car Counter System by using Jetson Nano Based on Computer Vision and Internet of Things*. ResearchGate.

Quốc Hùng, & Yên, T. (2024, December 24). *Nâng Cấp Hạ Tầng, Công Nghệ Quản Lý Giao Thông*. Báo Sài Gòn Giải Phóng. https://www.sggp.org.vn/nang-cap-ha-tang-cong-nghe-quan-ly-giao-thong-post777835.html

Rongviriyapanich, T., & Suppattrakul, C. (2005). *Effects of Motorcycles on Traffic Operations on Arterial Streets* (Vol. 6).

Seeed Studio. (2022,). *yolov5-swift: YOLOv5 in PyTorch > ONNX > CoreML > TFLite > UF2*. https://github.com/Seeed-Studio/yolov5-swift

Sutandi, A. C. (2017). Evaluation of exclusive stopping space for motorcycle at signalized intersections in large cities in Indonesia. *Journal of Urban Infrastructure*.

Sở Giao thông Vận tải TP.HCM. (2024). *Báo Cáo Tổng Kết Công Tác Quản Lý Hạ Tầng và Phương Tiện Giao Thông Năm 2024*. Sở Giao thông Vận tải TP. Hồ Chí Minh.

Sở Giao thông Vận tải TP.HCM. (2025). *Thông Cáo Báo Chí về Tình Hình Trật Tự An Toàn Giao Thông và Các Giải Pháp Kéo Giảm Ùn Tắc*. Sở Giao thông Vận tải TP. Hồ Chí Minh.

Transportation Research Board. (2010). *Highway Capacity Manual* (6th ed.). National Research Council.

Trần, M. Đ., & Nguyễn, T. H. (2024). Improvement of Vehicle Detection and Classification Performance with Region of Interest. *ITEJ*.

Tuấn Kiệt. (2025, January 13). *Ho Chi Minh City to reorganize traffic signals to tackle congestion*. VietnamNet. https://vietnamnet.vn/en/ho-chi-minh-city-to-reorganize-traffic-signals-to-tackle-congestion-2362641.html

VietNamNet. (2024, October 28). *Vietnam Leads the World in Motorcycle Usage with Over 77 Million Registered*. https://vietnamnet.vn/en/vietnam-leads-the-world-in-motorcycle-usage-with-over-77-million-registered-2338732.html

Vu, A. T., & Nguyen, D. V. M. (2015). *Mixed Traffic Saturation Flows of Signalized Intersections in Motorcycle Dominant Cities*.

Webster, F. V. (1958). *Traffic Signal Settings*. Road Research Laboratory.

World Bank. (2023). *Vietnam Urban Development Strategy: Managing Rapid Growth in Mega-Cities*.

APPENDICES

Extra Table

In this section, we will provide extra tables during this project included hardware specifications, software implementation and cost estimation

No.	Item Description	Quantity	Unit Price (VND)	Total (VND)
1	18650 Li-ion Rechargeable Battery 3.7V 2500mAh 10C	8	40,000	320,000
2	Seeed Studio XIAO ESP32S3 Sense (Wi-Fi + BLE + Camera + Mic)	4	459,000	1,836,000
			Total Estimated Cost	**2,156,000**

Table 3: Estimated Cost of Components

Category	Specification
Microcontroller (MCU)	**ESP32-S3R8** (Xtensa® 32-bit LX7 dual-core, up to 240 MHz)
Clock Speed	**Up to 240 MHz**
Flash Memory	**8 MB Flash**
PSRAM	**8 MB PSRAM**
Wireless Connectivity	Wi-Fi 802.11 b/g/n (2.4 GHz), Bluetooth® 5.0 (LE)
USB Interface	USB Type-C; supports USB 2.0 (programming, debugging, serial communication)
Camera	OV2640 camera (1600×1200), connected via ESP32-S3 camera interface
Interface	1× UART, 1× IIC, 1× IIS, 1× SPI, 11× GPIO (PWM), 9× ADC, 1× User LED, 1× Charge LED, 1× B2B connector (with 2 additional GPIO), 1× Reset button, 1× Boot button
Power	Input voltage (Type-C): 5 V Input voltage (BAT): 4.2 V
Circuit Operating Voltage	Type-C: 5 V @ 38.3 mA Battery: 3.8 V @ 43.2 mA
Dimensions	21 × 17.8 × 15 mm
Working Temperature	−40°C – 65°C

Table 4: Specification Table — XIAO ESP32S3 Sense

Extra Figures

In this section, we will provide extra figures relating to our prototype,our hardware and software.

Battery

Each board is powered by **two single-cell 3.7 V Li-ion batteries** connected through the **XIAO ESP32S3 Sense's onboard JST-PH port**, which also supports **integrated charging via USB-C**. The **onboard power management system automatically handles both charging and discharging**, enabling **fully autonomous battery operation** for several days depending on battery capacity and duty cycle.

Figure 23: 18650 Li-Ion Rechargeable Battery

Compute/Cam - Seeed XIAO ESP32S3 Sense

The XIAO ESP32S3 Sense features a dual-core Xtensa LX7 processor (240 MHz) with 8 MB PSRAM and 8 MB Flash, capable of **performing on-device AI inference** and **real-time image processing** for traffic counting tasks. Its **built-in vector instructions** and **AI libraries (ESP-NN, ESP-DSP)** provide sufficient compute throughput to run **TinyML models** such as Edge Impulse FOMO or MobileNet Tiny at several frames per second, while maintaining **low power consumption** for battery-based deployments.

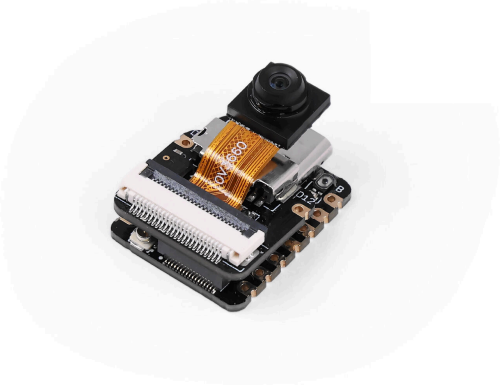

Figure 24: Seeed Studio XIAO ESP32S3 Sense

Prototype for traffic light

The ESP32-S3 is the main compute and control unit in our prototype. It receives traffic-density updates from AWS IoT (via MQTT over Wi-Fi), parses the incoming JSON payload, and then drives the traffic-light state machine to update the red/yellow/green outputs in real time.

Designed for embedded IoT applications, the ESP32-S3 integrates a dual-core Xtensa LX7 processor (up to 240 MHz), on-chip Wi-Fi (2.4 GHz) and Bluetooth Low Energy, and a rich set of GPIO and peripheral interface.This combination allows it to handle networking, message processing, timing/scheduling, and hardware control on a single low-power chip. In our system, periodic tasks (e.g., MQTT loop, scheduler tick, and light state transitions) run reliably with deterministic timing, while the wireless stack maintains a stable connection to AWS IoT. The

ESP32-S3 also supports edge AI acceleration and external memory, which makes it suitable for future extensions such as on-device vision inference and local traffic-density estimation.

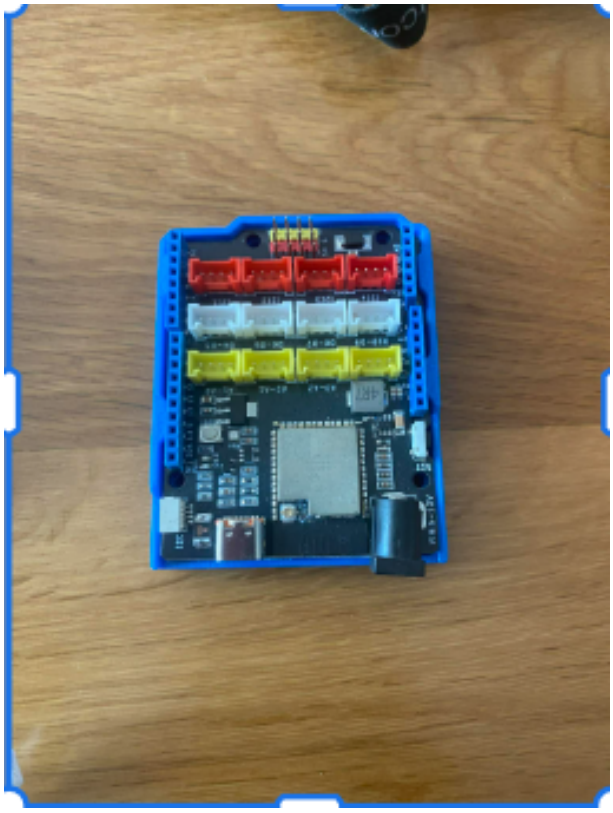

Figure 25: ESP32-S3 for controlling 2 traffic lights

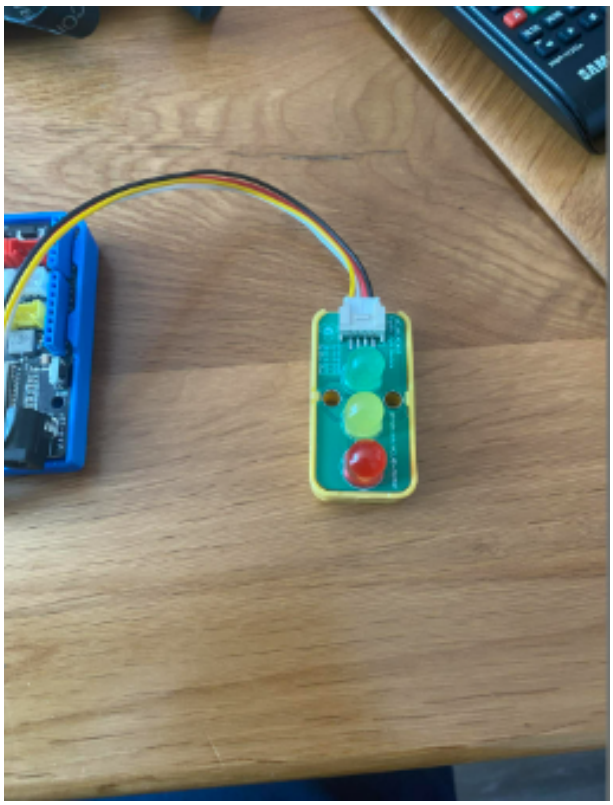

Figure 26: Traffic lights with 3 colors

AWS-IoT core

This project leverages the **Amazon Web Services (AWS) IoT Core** as the cloud platform for receiving traffic density data from deployed ESP32-S3 devices. AWS IoT Core provides a managed Message Queuing Telemetry Transport (MQTT) broker that enables secure, bidirectional communication between IoT devices and cloud applications. The integration provides real-time traffic monitoring and allows the captured data to be processed by the Arduino.

The **MQTT** protocol was chosen for this system because it is well suited to resource-constrained IoT environments, particularly for the ESP32S3 device and limited internet connection. MQTT is a messaging protocol that revolves around subscribing-publishing to topics, which helps reducereducing the system complexity while ensuring that all data is delivered and processed reliably.

In addition, MQTT provides multiple **Quality of Service (QoS)** levels (0, 1, and 2), allowing applications to balance reliability and latency according to system requirements. The protocol also supports persistent sessions, enabling messages to be queued and delivered after temporary network interruptions. Furthermore, its **topic-based hierarchical routing** mechanism allows scalable and organized data distribution across the system. In this implementation, QoS Level 1 (at-least-once delivery) is used to ensure reliable transmission of traffic density data to the cloud while maintaining sufficiently low latency for real-time monitoring.

Security Architecture

TLS Mutual Authentication

Secure communication between the ESP32-S3 devices and AWS IoT Core is achieved through **Transport Layer Security (TLS)** mutual authentication. This approach requires both the client device and the server to authenticate each other using X.509 certificates. On the device side, the AWS IoT broker is verified using a trusted root Certificate Authority (CA) certificate embedded in the firmware. Conversely, AWS IoT authenticates each device using a unique client certificate and its associated private key.

This mutual authentication mechanism significantly reduces the risk of unauthorized access and man-in-the-middle attacks, ensuring that only trusted devices are allowed to publish data to the IoT platform.

Certificate Management

Device credentials are embedded directly into the firmware binary using ESP-IDF's binary embedding mechanism (target_add_binary_data). The certificate chain includes the Amazon Root CA certificate for server verification, a device-specific client certificate issued by AWS IoT, and the corresponding private key.

While this approach is suitable for development and small-scale deployments, production systems should store cryptographic credentials in secure elements or hardware security modules (HSMs).

Network Security

All MQTT communication is conducted over TLS-encrypted connections using port 8883, which is the standard port for secure MQTT traffic. The system also supports Application-Layer Protocol Negotiation (ALPN), allowing MQTT communication over port 443 when required by restrictive network environments such as firewalls or captive networks.

Connection Establishment

The MQTT connection process follows a defined sequence. First, the device initializes its network interface and establishes a Wi-Fi connection. Once connected, a TLS handshake is initiated with the AWS IoT endpoint using the embedded root CA certificate. After successful TLS negotiation, the device sends an MQTT CONNECT packet containing a unique client identifier, a keep-alive interval, and session parameters. Upon successful authentication, AWS IoT responds with a CONNACK packet, completing the MQTT session setup.

To handle transient network failures, the system implements an exponential backoff retry strategy. Connection attempts are retried with progressively increasing delays, starting at 500 ms and capped at 5000 ms, up to a configurable maximum number of retries. This strategy helps avoid excessive network traffic during outages while still allowing the device to recover connectivity automatically.

Message Publishing

Traffic density data is published to a hierarchicalMQTT topic with the following structure:

esp32/traffic/density_now

Each MQTT message contains a JSON-formatted payload, for example:

```
{
  "density_now_dir1": 0.694,
  "density_pct": 69.4
}
```

or

```
{
  "density_now_dir2": 0.694,
  "density_pct": 69.4
}
```

The `density_now_dirX` field represents a normalized traffic density value ranging from 0.0 to 1.0, while `density_pct` provides a human-readable percentage representation of the same metric. The numerical suffix (X) identifies the ESP32 device or traffic direction associated with the measurement. For instance, one edge device publishes data with `density_now_dir1` in the message, whereas additional devices publish with `density_now_dir2`.

Traffic density updates are published at configurable intervals, with a default value of 90 seconds. This interval represents a compromise between timely data updates, power efficiency, and network bandwidth usage. During operation, the MQTT client periodically invokes `MQTT_ProcessLoop()` to handle keep-alive messages, acknowledgments for QoS 1 transmissions, and any incoming packets.

Implementation Details

The system uses the FreeRTOS CoreMQTT library, which is designed for embedded systems with limited memory. The library offers MQTT 3.1.1 compliance, thread-safe operation under FreeRTOS, and configurable buffer sizes to accommodate constrained environments such as the ESP32-S3.

A custom network transport layer is implemented using ESP-IDF's TLS socket APIs. This layer abstracts platform-specific networking details and handles TLS session management, socket communication, and error handling.

Connection parameters such as the AWS IoT endpoint, client identifier, and port configuration are manually managed through ESP-IDF's menuconfig system.

Several improvements can be built on top of the current implementation, including the use of AWS IoT Device Shadow for state synchronization, AWS IoT Rules for automated event handling, and AWS IoT Jobs for over-the-air firmware updates. Additional enhancements may also include support for multiple traffic directions and advanced data analytics.

Code snippet

Horizontal fill algorithm

```
1  static float compute_hfill( std::vector<BoxF> & dets, const RoiPx & roi, int
frame_w, int frame_h, int * out_box_count )
2  {
3      if( out_box_count ≠ nullptr )
4      {
5          *out_box_count = 0;
6      }
7
8      // Convert detections to x-intervals after ROI clip, then merge.
9      std::sort( dets.begin(), dets.end(), []( const BoxF & a, const BoxF & b )
{ return a.score > b.score; } );
10
11      std::vector<IntervalF> intervals;
12      intervals.reserve( CONFIG_MBDET_TOPK );
13
14      const float fx1 = static_cast<float>( roi.x1 );
15      const float fx2 = static_cast<float>( roi.x2 );
16      const float fy1 = static_cast<float>( roi.y1 );
17      const float fy2 = static_cast<float>( roi.y2 );
18
19      for( const auto & d : dets )
20      {
21          if( d.score < score_thr() )
22          {
23              break;
24          }
25          if( static_cast<int>( intervals.size() ) ≥ CONFIG_MBDET_TOPK )
26          {
27              break;
28          }
29
30          // d is in frame pixels already (we will store as pixels below).
31          float x1 = d.x1;
32          float y1 = d.y1;
33          float x2 = d.x2;
34          float y2 = d.y2;
35
36          // Clip to ROI.
37          x1 = std::max( x1, fx1 );
38          x2 = std::min( x2, fx2 );
39          y1 = std::max( y1, fy1 );
40          y2 = std::min( y2, fy2 );
41
42          if( x2 ≤ x1 || y2 ≤ y1 )
43          {
44              continue;
45          }
46
47          intervals.push_back( { x1, x2 } );
48      }
49
50      if( out_box_count ≠ nullptr )
51      {
```

```
52          *out_box_count = ( int ) intervals.size();
53      }
54
55      if( intervals.empty() )
56      {
57          return 0.f;
58      }
59
60      std::sort( intervals.begin(), intervals.end(), []( const IntervalF & a,
const IntervalF & b ) { return a.x1 < b.x1; } );
61
62      float merged_len = 0.f;
63      IntervalF cur = intervals[ 0 ];
64      for( size_t i = 1; i < intervals.size(); ++i )
65      {
66          const IntervalF nxt = intervals[ i ];
67          if( nxt.x1 ≤ cur.x2 )
68          {
69              cur.x2 = std::max( cur.x2, nxt.x2 );
70          }
71          else
72          {
73              merged_len += ( cur.x2 - cur.x1 );
74              cur = nxt;
75          }
76      }
77      merged_len += ( cur.x2 - cur.x1 );
78
79      const float roi_w = static_cast<float>( roi.x2 - roi.x1 );
80      if( roi_w ≤ 1.f )
81      {
82          return 0.f;
83      }
84
85      return clamp01( merged_len / roi_w );
```

Listing 1: Horizontal fill computation (`compute_hfill`) for density estimation.

Task Scheduler

In this project, our group can not directly adjust real roadside traffic lights on the road because of violating regulations and exceeding our authorization. Therefore, we implemented a simple prototype using ESP32 microcontroller combining two leds to simulate a traffic light. For working on real time and parallelly listening messages from MQTT Amazon Console, we utilized real time knowledge on tick-based scheduler. This scheduler run periodically three tasks: a timer run per 1 ticks a state machine run every 10 ticks for evaluating transitions and AWS loop tasks every 50 ticks for recording new messages from consoles.

```
1  #include "Arduino.h"
2  #include "scheduler.h"
3
4  #ifdef __cplusplus
5  extern "C" {
6  #endif
7
```

```
8  #define INTERRUPT_CYCLE   10 // 10 milliseconds
9  #define PRESCALER         64
10  #define COUNTER_START     65536 - INTERRUPT_CYCLE * 1000 * 16 / 64 // 16 MHz
core clock
11
12  typedef struct {
13    void (*pTask)(void);
14    uint32_t Delay;
15    uint32_t Period;
16    uint8_t RunMe;
17    uint32_t TaskID;
18  } sTask;
19
20  // The array of tasks
21  static sTask SCH_tasks_G[SCH_MAX_TASKS];
22  static uint8_t array_Of_Task_ID[SCH_MAX_TASKS];
23  static uint32_t newTaskID = 0;
24  static uint32_t rearQueue = 0;
25  static uint32_t count_SCH_Update = 0;
26
27  static uint32_t Get_New_Task_ID(void);
28  static void TIMER_Init(void);
29
30  // Hardware timer handler
31  hw_timer_t *Timer0_Cfg = NULL;
32
33  void SCH_Init(void){
34    // TIMER_Init();
35  }
36
37  void SCH_Update(void){
38    count_SCH_Update++;
39    if (SCH_tasks_G[0].pTask && SCH_tasks_G[0].RunMe == 0) {
40      if (SCH_tasks_G[0].Delay > 0) {
41        SCH_tasks_G[0].Delay--;
42      }
43      if (SCH_tasks_G[0].Delay == 0) {
44        SCH_tasks_G[0].RunMe = 1;
45      }
46    }
47  }
48
49  uint32_t SCH_Add_Task(void (*pFunction)(), uint32_t DELAY, uint32_t PERIOD){
50    uint8_t newTaskIndex = 0;
51    uint32_t sumDelay = 0;
52    uint32_t newDelay = 0;
53
54    for (newTaskIndex = 0; newTaskIndex < SCH_MAX_TASKS; newTaskIndex++){
55      sumDelay += SCH_tasks_G[newTaskIndex].Delay;
56
57      if (sumDelay > DELAY){
58        newDelay = DELAY - (sumDelay - SCH_tasks_G[newTaskIndex].Delay);
59        SCH_tasks_G[newTaskIndex].Delay = sumDelay - DELAY;
60
61        for (uint8_t i = SCH_MAX_TASKS - 1; i > newTaskIndex; i--){
62          SCH_tasks_G[i].pTask  = SCH_tasks_G[i - 1].pTask;
63          SCH_tasks_G[i].Period = SCH_tasks_G[i - 1].Period;
```

```
64          SCH_tasks_G[i].Delay  = SCH_tasks_G[i - 1].Delay;
65          SCH_tasks_G[i].TaskID = SCH_tasks_G[i - 1].TaskID;
66        }
67
68        SCH_tasks_G[newTaskIndex].pTask  = pFunction;
69        SCH_tasks_G[newTaskIndex].Delay  = newDelay;
70        SCH_tasks_G[newTaskIndex].Period = PERIOD;
71        SCH_tasks_G[newTaskIndex].RunMe  = (newDelay == 0) ? 1 : 0;
72        SCH_tasks_G[newTaskIndex].TaskID = Get_New_Task_ID();
73        return SCH_tasks_G[newTaskIndex].TaskID;
74
75      } else {
76        if (SCH_tasks_G[newTaskIndex].pTask == 0x0000){
77          SCH_tasks_G[newTaskIndex].pTask  = pFunction;
78          SCH_tasks_G[newTaskIndex].Delay  = DELAY - sumDelay;
79          SCH_tasks_G[newTaskIndex].Period = PERIOD;
80          SCH_tasks_G[newTaskIndex].RunMe  = (SCH_tasks_G[newTaskIndex].Delay ==
0) ? 1 : 0;
81          SCH_tasks_G[newTaskIndex].TaskID = Get_New_Task_ID();
82          return SCH_tasks_G[newTaskIndex].TaskID;
83        }
84      }
85    }
86    return SCH_tasks_G[newTaskIndex].TaskID;
87  }
88
89  uint8_t SCH_Delete_Task(uint32_t taskID){
90    uint8_t Return_code = 0;
91
92    if (taskID != NO_TASK_ID){
93      for (uint8_t taskIndex = 0; taskIndex < SCH_MAX_TASKS; taskIndex++){
94        if (SCH_tasks_G[taskIndex].TaskID == taskID){
95          Return_code = 1;
96
97          if (taskIndex != 0 && taskIndex < SCH_MAX_TASKS - 1){
98            if (SCH_tasks_G[taskIndex + 1].pTask != 0x0000){
99              SCH_tasks_G[taskIndex + 1].Delay += SCH_tasks_G[taskIndex].Delay;
100           }
101         }
102
103         for (uint8_t j = taskIndex; j < SCH_MAX_TASKS - 1; j++){
104           SCH_tasks_G[j] = SCH_tasks_G[j + 1];
105         }
106
107         SCH_tasks_G[SCH_MAX_TASKS - 1].pTask  = 0;
108         SCH_tasks_G[SCH_MAX_TASKS - 1].Period = 0;
109         SCH_tasks_G[SCH_MAX_TASKS - 1].Delay  = 0;
110         SCH_tasks_G[SCH_MAX_TASKS - 1].RunMe  = 0;
111         SCH_tasks_G[SCH_MAX_TASKS - 1].TaskID = 0;
112
113         return Return_code;
114       }
115     }
116   }
117   return Return_code;
118 }
119
```

```
120 void SCH_Dispatch_Tasks(void){
121   if (SCH_tasks_G[0].RunMe > 0){
122     (*SCH_tasks_G[0].pTask)();
123     SCH_tasks_G[0].RunMe = 0;
124 
125     sTask temtask = SCH_tasks_G[0];
126     SCH_Delete_Task(temtask.TaskID);
127 
128     if (temtask.Period != 0){
129       SCH_Add_Task(temtask.pTask, temtask.Period, temtask.Period);
130     }
131   }
132 }
133 
134 static uint32_t Get_New_Task_ID(void){
135   newTaskID++;
136   if (newTaskID == NO_TASK_ID){
137     newTaskID++;
138   }
139   return newTaskID;
140 }
141 
142 void IRAM_ATTR Timer0_ISR(){
143   SCH_Update();
144 }
145 
146 static void TIMER_Init(void){
147   // configure timer here
148 }
149 
150 #ifdef __cplusplus
151 }
152 #endif
```

Listing 2: Real-time scheduler implementation

```
1  init_traffic_light();
2 
3  // Add tasks to scheduler:
4  // timerRun: period 1 (runs every tick to decrement timers)
5  SCH_Add_Task(timerRun, 0, 1);
6 
7  // state_machine: period 10 (runs every 10 ticks to check state transitions)
8  SCH_Add_Task(state_machine, 0, 10);
9 
10  // awsLoopTask: period 50 (runs every 50 ticks for MQTT loop)
11  SCH_Add_Task(awsLoopTask, 0, 50);
```

Listing 3: Task assignment for three periodic tasks

```
1    // helper: read float from JSON and convert to percentage (0-100)
2  // Handles both integer percentages (0-100) and decimal values (0.0-1.0)
3  static int readJsonPercentage(const JsonDocument& doc, const char* key, int
defVal) {
4    if (!doc.containsKey(key)) return defVal;
5 
```

```
  float floatVal = 0.0;

  // Try to read as float (handles both int and float JSON numbers)
  if (doc[key].is<float>() || doc[key].is<double>() || doc[key].is<int>()) {
    floatVal = doc[key].as<float>();
  } else if (doc[key].is<const char*>()) {
    const char* s = doc[key].as<const char*>();
    if (!s) return defVal;
    floatVal = atof(s);
  } else {
    return defVal;
  }

  // If value is between 0.0 and 1.0, it's a decimal percentage (convert to 0-100)
  if (floatVal ≥ 0.0 && floatVal ≤ 1.0) {
    return (int)(floatVal * 100.0 + 0.5); // Round to nearest integer
  }
  // Otherwise, assume it's already a percentage (0-100)
  else if (floatVal ≥ 0.0 && floatVal ≤ 100.0) {
    return (int)(floatVal + 0.5); // Round to nearest integer
  }

  return defVal; // Invalid range
}

static void messageHandler(String &topic, String &payload) {
  Serial.println("═══ MQTT MESSAGE ═══");
  Serial.print("Topic: "); Serial.println(topic);
  Serial.print("Raw payload: "); Serial.println(payload);

  StaticJsonDocument<256> doc;
  DeserializationError err = deserializeJson(doc, payload);
  if (err) {
    Serial.print("JSON parse failed: ");
    Serial.println(err.c_str());
    Serial.println("═══════════════════");
    return;
  }

  // optional sender field (if you include deviceId)
  const char* sender = doc["deviceId"] | "";
  Serial.print("Sender: ");
  Serial.println(sender[0] ? sender : "(none)");

  // ——— Accept density percentage values (0-100 or 0.0-1.0):
  // 1) {"density_now":80} or {"density_now":0.80} → applies to both directions (80% of max vehicles)
  // 2) {"density_now_dir1":60,"density_now_dir2":80} → separate directions
  // Handles both integer percentages (0-100) and decimal values (0.0-1.0 converted to 0-100)
  int density_percentage  = readJsonPercentage(doc, "density_now", -1);
  int d1_percentage = readJsonPercentage(doc, "density_now_dir1", -1);
  int d2_percentage = readJsonPercentage(doc, "density_now_dir2", -1);

  // Configuration: 100% density = 50 vehicles (max capacity)
  const int MAX_VEHICLES = 219;
```

```
60    const int MIN_PERCENTAGE_THRESHOLD = 20; // Minimum percentage to update (0
= accept all valid values)
61
62    // Convert percentage (0-100) to real vehicle count
63    // Formula: real_density = (percentage * MAX_VEHICLES) / 100
64
65    // If "density_now" exists: apply to both directions
66    if (density_percentage ≥ MIN_PERCENTAGE_THRESHOLD && density_percentage ≤
100) {
67      int d = (density_percentage * MAX_VEHICLES) / 100;
68      setDensityDir1(d);
69      setDensityDir2(d);
70
71      int g1 = compute_green_seconds(d);
72      int g2 = compute_green_seconds(d);
73
74      Serial.print("Parsed density_percentage = ");
Serial.println(density_percentage);
75      Serial.print("Converted to vehicles = "); Serial.println(d);
76      Serial.print("Computed GREEN1 seconds = "); Serial.println(g1);
77      Serial.print("Computed GREEN2 seconds = "); Serial.println(g2);
78    }
79    // Else if separate densities exist
80    else if ((d1_percentage ≥ MIN_PERCENTAGE_THRESHOLD && d1_percentage ≤ 100)
||
81             (d2_percentage ≥ MIN_PERCENTAGE_THRESHOLD && d2_percentage ≤
100)) {
82      int d1 = -1, d2 = -1;
83
84      if (d1_percentage ≥ MIN_PERCENTAGE_THRESHOLD && d1_percentage ≤ 100) {
85        d1 = (d1_percentage * MAX_VEHICLES) / 100;
86        setDensityDir1(d1);
87        int g1 = compute_green_seconds(d1);
88        Serial.print("Parsed density_now_dir1 = "); Serial.print(d1_percentage);
89        Serial.print("% → vehicles = "); Serial.print(d1);
90        Serial.print(" → GREEN1 seconds = "); Serial.println(g1);
91      }
92
93      if (d2_percentage ≥ MIN_PERCENTAGE_THRESHOLD && d2_percentage ≤ 100) {
94        d2 = (d2_percentage * MAX_VEHICLES) / 100;
95        setDensityDir2(d2);
96        int g2 = compute_green_seconds(d2);
97        Serial.print("Parsed density_now_dir2 = "); Serial.print(d2_percentage);
98        Serial.print("% → vehicles = "); Serial.print(d2);
99        Serial.print(" → GREEN2 seconds = "); Serial.println(g2);
100      }
101    }
102    else {
103      Serial.println("No valid density fields found in payload.");
104      Serial.println("Expected: 'density_now' (0-100) or
'density_now_dir1'/'density_now_dir2' (0-100)");
105    }
106
107    Serial.println("═══════════════════");
108  }
```

Listing 4: Code for handling message from MQTT and compute cycle