

Edge AI System for Traffic Monitoring: Helmet Violation Detection and Evidence Packaging

Chia-Cheng Wei¹, Yi-Da Huang¹, Jia-Siang Han¹, Tain-Lieng Kao^{2*}

¹Dept. of Intelligent Network Technology, I-Shou University, Taiwan, R.O.C.

^{2*} Assistant professor, Dept. of Information Engineering, I-Shou University, Taiwan, R.O.C. Corresponding Author

Abstract - In Taiwan, the citizen reporting workflow for motorcyclists who do not wear helmets often stalls at the "evidence collection" step: a usable report must include an identifiable license plate, a timestamp, and a location, and it must be successfully submitted on a low-cost device that may need to operate offline. We use YOLOv5s for helmet compliance detection and train a second YOLOv5 model for license plate localization; we then perform geometric rectification using a quad-first/Hough-fallback strategy and read the plate using a Tesseract 4.1.1 LSTM ensemble (three binarization variants \times three PSM modes), with character-level voting guided by priors that match Taiwan license plate formats. Finally, we use Selenium to automatically populate and submit the police agency's reporting form. On a Raspberry Pi 5 edge deployment with 480×480 input and OpenCV 4.8 DNN (FP16/NEON), YOLOv5 detection averages 278.2 ms per image, while the OCR pipeline averages 1,705.6 ms per image (the ensemble OCR accounts for $\sim 78\%$ of the latency). We also evaluate non-frontal images and obtain correct recognition results.

Key Words: Edge AI, helmet violation detection, license plate recognition, automated citizen reporting.

1. INTRODUCTION

A green light turns. A motorcyclist without a helmet crosses the intersection. The evidence is not absent. Capture is the hard part. Under Article 7-1 of the Road Traffic Management and Penalty Act [6], a citizen may file a report. A usable report needs three items in a single shot: a legible license plate, an accurate timestamp, and a precise location. The result is then uploaded from a handheld device. In practice this takes several rounds of manual editing.

In 2024, Taiwan recorded 2,950 road-traffic fatalities within a 30-day window, or 12.6 per 100,000 population. Japan recorded 2,663 fatalities, or 2.14 per 100,000. The ratio is about 5.9. Sweden sits near 2.2 per 100,000 [1][2][3]. Road conditions alone cannot explain the gap. Helmet non-use is a leading factor in motorcyclist casualties [4][5]. Citizen reporting is a legal channel for intervention. Its evidence-collection workflow remains underdeveloped.

All four stages run on a single Raspberry Pi 5: detection, evidence capture, structuring, and submission. First,

YOLOv5s detects helmetless riders. Input is 480×480 . Mean per-image latency on the Pi 5 is 278.2 ms ($n = 2$ pilot). Second, an independently trained YOLOv5 detector locates the license plate. A Tesseract 4.1.1 LSTM ensemble reads it. Mean latency is 1,705.6 ms per image ($n = 2$ pilot). The ensemble OCR accounts for about 78% of this latency. Third, a NEO-M8 GNSS receiver supplies latitude and longitude. Nominatim reverse geocoding converts the reading into a city / district / street address. Openpyxl then produces an Excel evidence package with embedded thumbnails. Fourth, Selenium populates the police agency's online reporting form. Imagery, location, and timestamp are produced on the same edge device. Photos are not offloaded to the cloud [14][15].

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the system design. Section 4 reports per-image latency and pilot recognition results at the default 480×480 input. Section 5 concludes this study.

2. RELATED WORK

2.1 Feasibility of Vision-Based Algorithms

The question in this period was straightforward: whether the algorithms themselves were sufficient. Helmet detection, license-plate recognition, and embedded ignition interlock were validated separately, not yet integrated into a single system.

Silva et al. [7] provide a pre-deep-learning baseline. Their pipeline is classical CV: background subtraction, ROI CHT, HOG, and MLP. Accuracy reaches $\approx 91.37\%$ under 10-fold cross-validation on CPU-only hardware. Anagnostopoulos [8] surveys LPR in tutorial form across three stages: localization, segmentation, and recognition. He identifies image-capture conditions as the upper-bound determinant of recognition. He also calls for separate reporting of character-level and plate-level accuracy.

Shi et al. propose CRNN [9]. Redmon et al. propose YOLOv1 [10]. The former trains from string-level labels alone, avoiding error propagation from character segmentation. The latter casts detection as a single regression and achieves real-time inference.

Deployment then shifts toward single-board computers. Athuljith et al. [11] extend the line to a Raspberry Pi 3.

They combine a Haar Cascade, a relay, and RTO notification into a pre-ride ignition-interlock prototype. Melchera et al. [12] propose a two-layer eCall design from an ITS perspective. The design couples BLE, helmet sensing, an Android application, and EN 15722 MSD extensions. Neither is complete. [11] is only a functional demonstration without quantitative measurement. [12] stops at conceptual design and offers no prototype measurement.

By the end of this period, the algorithmic options were in place. The reporting conventions for quantitative evaluation were not.

2.2 Traffic-Event Determination

Franklin and Mohana [13] are representative of this period. Using YOLOv3, they address red-light running, speeding, and vehicle counting jointly. They add ROI definitions (stop line and signal line), inter-frame logic, and IoU thresholds. They report accuracies of 97.67% for vehicle counting and 89.24% for speeding violations. A single detection output, combined with event-level logic, can yield multiple violation determinations.

However, deployment transparency remains insufficient. Reference [13] does not adequately disclose ground truth, data splits, or deployment conditions (hardware, FPS, latency). The conclusions are difficult to reproduce on edge platforms.

The bottleneck has moved from detector metrics to event determination and the chain of evidence. Yet on-device measurement at the edge has not been adequately reported in the works above.

2.3 Integrated Quantification on the Edge

VegaEdge, by Katariya et al. [14], reports latency, throughput, and power simultaneously. The system (YOLOv8l + ByteTrack + PishguVe) is deployed on Jetson Orin and Xavier NX. On Jetson Orin, it reaches 758 trajectories per second at approximately 18 W. Angle-based anomaly detection attains AUC-ROC of 0.94. Event-level determination is handled by trajectory-deviation heuristics.

This is quantification on a high-end platform. The low-end counterpart is Tahilramani et al. [15]. Their Raspberry Pi-based YOLOv9 helmet-compliance monitoring overlaps closely with the present work. Several items are not sufficiently disclosed: whether on-device latency is measured on the Pi itself, generalization to nighttime and occlusion, and integration with enforcement workflows. We add license-plate evidence capture, structured evidence packaging, and automated submission. We do not pursue a head-to-head detection-accuracy comparison.

From the event-logic side, Kathait et al. [16] provide an alternative design. They incorporate MOT, a no-parking polygon, and a dwell-time threshold into illegal-parking detection. Violation determination is recast from the “instantaneous frame” to the “event duration.” This can reduce false alarms from brief stops and traffic jams. The authors do not provide full latency or F1 measurements. Feasibility on Raspberry Pi-class compute remains to be verified.

Without disclosure of latency, power, and hardware, edge-AI results are difficult for others to reproduce. Reporting conventions across high-end and low-end platforms have not yet converged.

2.4 Our Approach

Viewing the three periods together, four points of consensus emerge. Real-time detectors of the YOLO family, coupled with event-level logic (ROI, tracking, and time thresholds), can produce usable violation determinations [7][10][13][16]. License-plate recognition must be evaluated as an end-to-end pipeline capture, localization, and recognition rather than at any single stage [8][9]. Edge systems must be quantified along latency, throughput, and power simultaneously [14]. Traffic safety is a cross-module integration problem [11] [12].

Yet the last mile has not been stitched together. Works [7]–[16] each complete only one segment of the chain. Some emphasize detector or classifier performance. They lack license-plate evidence capture and a submittable evidence chain. Others claim “edge AI” yet provide no end-to-end, reproducible measurement on resource-constrained single-board computers such as the Raspberry Pi 5. Adverse capture conditions (nighttime, occlusion, and plate soiling) are particularly under-reported.

We constrain the scope to a single Raspberry Pi 5. We integrate the four stages of detection, evidence capture, structuring, and submission into one end-to-end pipeline. The measurement is quantifiable and reproducible.

3. SYSTEM DESIGN

3.1 System Overview

Two machines split the workload. Figure 1 shows the six stages across them. We build the dataset and train the weights on the workstation. The Raspberry Pi 5 handles model conversion, real-time inference, license-plate recognition, and structured evidence generation in a single pipeline.

The edge device handles detection and evidence capture without network access [14][15]. The handoff

between stages uses standardized artifacts. Training and augmentation stay on the workstation; the Pi only runs inference and writes output. The rest of this section walks through the stages in that order.

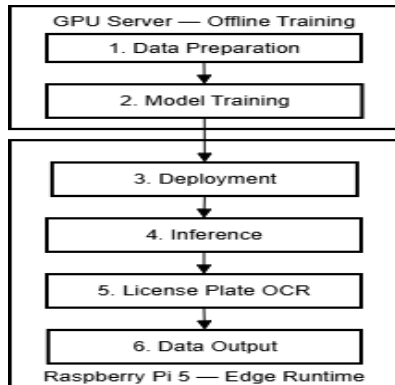


Fig -1: Six stages of system

3.2 Data Preparation

This stage builds the image sets we use for training and evaluation. Figure 2 traces the procedure from collection through partitioning. Two sources feed in. Public datasets give us 2,000 samples. We capture another 900 ourselves in Taiwan. That leaves 2,900 images going into screening. We then strip out anything severely blurred, duplicates of an earlier scene, and badly over-exposed frames. The survivors go to Roboflow Annotate, where we label them under three classes — motorcycle, head, and helmet.

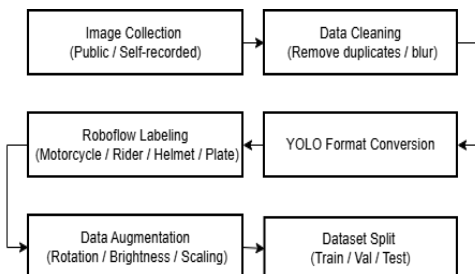


Fig -2: Procedures of data preparation

We export annotations in YOLO label format, one coordinate file per image. Each file lists the class label and the normalized bounding-box coordinates. Augmentation runs offline, before training begins. We apply three. A horizontal flip, random rotation up to $\pm 15^\circ$, and a brightness shift within $\pm 20\%$. We split the set 8:1:1. That gives us 2,320 images for training, 290 images for validation, and 290 images for testing. For license-plate detection we build a second dataset: 600 Taiwanese plate images, with only the plate box labeled. The 8:1:1 split carries over.

3.3 Model Training

YOLOv5s was chosen for three reasons. First, deployment limits. The system runs on a Raspberry Pi 5 (4-

core Arm CPU, 4 GB RAM). This budget must also cover detection, OCR, GNSS parsing, and the Selenium pipeline.

Second, real-time response. The pipeline must finish one violation frame before the next event arrives. At 480×480 , YOLOv5s takes 278.2 ms per image on the Pi 5. Larger variants would push latency into seconds and reduce margin under the 5-second cooldown.

Third, tool chain maturity. YOLOv5 has stable training recipes, many pretrained checkpoints, and a reliable ONNX export path. Export is direct (no custom plugin layers), and OpenCV DNN can run the model without an extra runtime.

We also evaluated YOLOv8 and YOLOv9. They offer refinements and slightly higher COCO mAP. We prioritized a reproducible pipeline on a single-board computer over peak accuracy. On the Pi 5, added toolchain risk (drivers, exporter regressions, fewer reference deployments) outweighed the likely gain [10][14].

YOLOv5s was trained on an NVIDIA RTX 5070 Ti (8 GB VRAM) GPU. We follow the standard YOLOv5 layout in Fig. 3.

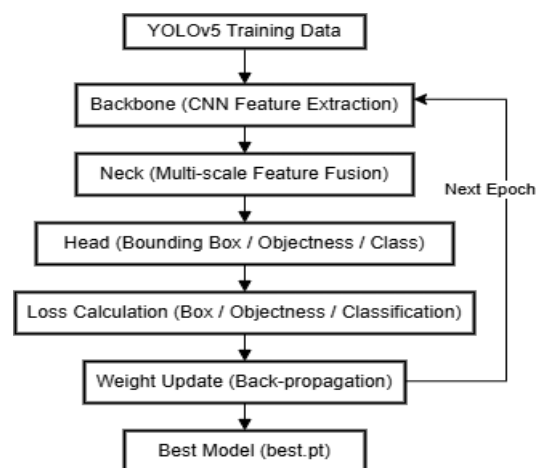


Fig -3: Model training

Training hyper-parameters are kept standard. Inputs of 640×640 are fed in batches of 16 for 100 epochs. Transfer learning is the lever for both fast convergence and overfitting control on a dataset this small. During training we monitor validation mAP@0.5 and keep the best checkpoint, which feeds the next stage. The plate detector is trained separately with the same recipe but only the plate class, which keeps it from interfering with the helmet model in class space.

3.4 Model Conversion and Deployment

The weight file is converted to ONNX for edge deployment. ONNX is a cross-framework format. During conversion we fix the input size at 640×640 but leave the batch dimension dynamic. On the Raspberry Pi 5 we run inference through OpenCV DNN. Input resolution we pick empirically among 320, 480, and 512.

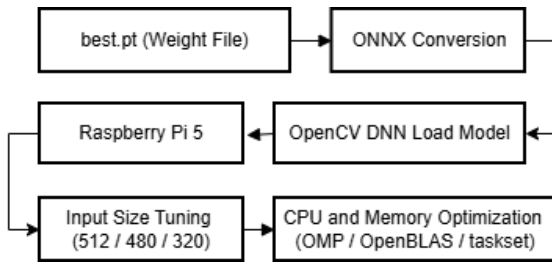


Fig -4: Model Conversion and Deployment

3.5 Real-time Inference

On the Raspberry Pi 5, we run frame-level violation checks on the live camera stream. Each frame is processed in four steps: preprocessing, YOLOv5 inference, post-processing, and event determination. We implement the core computer-vision steps with three reusable utilities: letterbox, parse_yolov5_output, and nms.

letterbox resizes the input with gray padding and returns the geometric parameters needed to map detections back to the original image. parse_yolov5_output converts the ONNX outputs into boxes and scores, and nms removes duplicate detections.

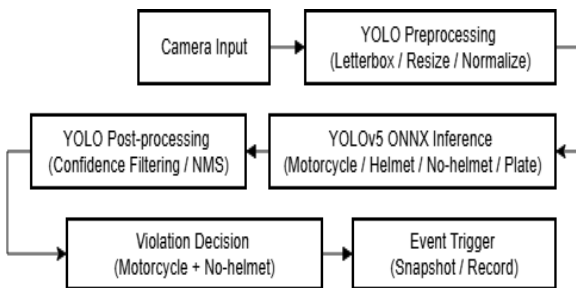


Fig -5: Real-time Inference

Violation logic is a lightweight spatial rule over three classes (motorcycle, head, helmet). A rider is flagged when a head overlaps a motorcycle but no corresponding helmet is detected. Both the raw and annotated frames are stored as downstream inputs.

To prevent repeated triggers across consecutive frames, events are rate-limited with a 5-second cooldown. This heuristic assumes sufficient illumination and does not target multi-rider or heavily occluded cases.

3.6 License Plate OCR

This stage converts a violation screenshot into a license-plate string. The plate is first localized with a dedicated YOLOv5 detector, a padded ROI is cropped, and lightweight preprocessing is applied. Text is then recognized using a Tesseract (LSTM) ensemble. Three binarization variants \times three PSM modes produce nine candidates, and a Taiwan plate-format prior guides character-level voting so that the final plate string can be obtained. Algorithm 1 summarizes the ensemble procedure. Latency impact is reported in 4.3.

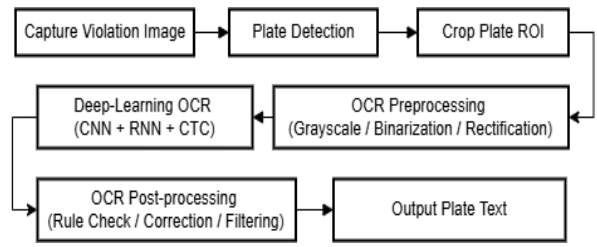


Fig -6: License Plate OCR

Algorithm 1: License Plate OCR Ensemble

Input: plate ROI image P

binarization configuration set V =

- { (scale=2.0, otsu),
- (scale=2.8, otsu),
- (clahe, otsu) }

page-segmentation mode set M = {7, 8, 13}

Output: final plate string s*

- 1: candidates $\leftarrow \emptyset$
- 2: for each v $\in V$ do
- 3: $I_v \leftarrow \text{Binarize}(P, v)$
- 4: for each m $\in M$ do
- 5: $t \leftarrow \text{Tesseract_OCR}(I_v, \text{psm} = m, \text{lang} = \text{"eng"})$
- 6: candidates \leftarrow candidates $\cup \{t\}$
- 7: end for
- 8: end for
- 9: $s^* \leftarrow \text{CharacterVote}(\text{candidates}, \text{TW_PlatePattern})$
- 10: return s*

3.7 Data Output

Fig-7 shows the violation evidence into a structured record. Location comes from a NEO-M8 multi-constellation GNSS module. A background thread keeps reading latitude and longitude, and the moment an event fires we attach the latest valid fix to the event metadata. Packaging follows two tracks: one for local archiving and one for submission. The first output is an Excel worksheet. We embed a downscaled violation thumbnail next to the license-plate string, timestamp, and address fields. The second output is automated submission. The same structured fields are passed to Selenium WebDriver, which targets the police agency's online form. The script clicks through the pre-submission confirmation page and automatically fills in the reporter information that was saved in advance.

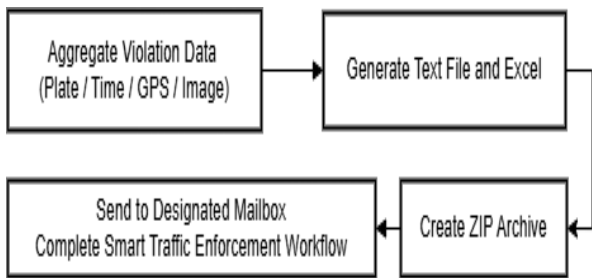


Fig -7: Data Output

4. RESULTS AND ANALYSIS

This section presents the system test results and analyzes them in detail. 4.1 sizes up input resolution and picks the default working point. 4.2 through 4.4 trace the evidence chain across violation detection, OCR, and packaging. 4.5 closes on the back-end form and personal-data handling. Table 1 lists the setup.

Table -1: Setup configuration.

| Item | Specification |
|-------------------------|-------------------------------|
| Edge computing platform | Raspberry Pi 5 (4 GB RAM) |
| Camera | IMX219 (8 MP, 1080p @ 30 fps) |
| GNSS module | NEO-M8 |
| Deep-learning framework | OpenCV 4.8 + ONNX Runtime |
| OCR engine | Tesseract 4.1.1 |

4.1 Input Resolution and System Load

Input resolution trades YOLO inference latency against small-target recognizability. Three square sizes were tested offline: 320×320, 480×480, and 512×512. At each setting, we logged CPU usage, thread count, load average, and resident memory. The system-side numbers came close: 1-minute load averages of 2.22, 2.37, 2.37 (a spread of about 7%), driver-process RSS in 187–233 MB, and total resident memory between 1.236 GB and 1.256 GB out of 7.876 GB. The image side told a different story. At 480×480 and 512×512 the plate-character edges stayed readable; at 320×320 they fell apart. 480×480 was selected as the default, the lower of the two readable resolutions.

4.2 Violation Event Detection and Evidence Capture

The pipeline turns a frame into evidence in one shot. When all three classes are detected in the same frame and the violation condition is satisfied, the raw BGR image is captured for downstream OCR. An annotated copy is saved for human review, and the latest valid GNSS fix is written to a file together with the Nominatim reverse-geocoded

address. Figure 8 is a nighttime street violation. The boxes mark the motorcycle in green and the rider's head in red. There is no helmet box. The frame meets the violation condition. The system logs the timestamp and location. Figure 9 is a harder case. The plate sits off-axis to the camera. It is used to check whether quad-first rectification keeps the character boundaries readable when the plate is rotated away from the lens.



Fig -8: Test scenario 1



Fig -9: Test scenario 2

4.3 License Plate OCR Results

The OCR pipeline runs end to end. It starts at plate localization, walks the ROI through quad-first rectification and character-edge cleanup, then hands the rectified ROI to the Tesseract ensemble for the final character vote. On the n = 2 pilot, the plate detector locked onto both images with high confidence even at different incidence angles (det = 0.93 and 0.95). After preprocessing, the ensemble vote in Algorithm 1 settled on the same string for both images: "NJN-9327". The agreement is only as strong as the pilot allows. Table 2 breaks down the per-unit mean latency by stage. Ensemble OCR eats about 78% of the per-image budget. The cost falls in two places: running preprocessing multiple times, and voting across PSM modes.

Table -2: Per-image / per-plate mean latency of the license-plate OCR pipeline (n = 2 images / 2 plates).

| Stage | Unit | Mean latency |
|-----------------------------------|-----------|--------------|
| Plate localization (detect) | per image | 278.2 ms |
| Geometric rectification (rectify) | per plate | 3.3 ms |
| Normalization + ROI | per plate | 78.0 ms |
| Ensemble OCR | per plate | 1,330.3 ms |
| Per-image total latency | per image | 1,705.6 ms |

4.4 Evidence Aggregation and Dispatch

The dual-track design wraps the violation evidence into a structured package. Packaging kicks in automatically once OCR finishes. The first output is an Excel report with violation thumbnails embedded inline. We then bundle the report with the same-name image files into a ZIP archive and SMTP-dispatch the result to a preset recipient mailbox. The ZIP holds the image and the Excel file. No human keystroke is needed end to end.

4.5 Form and Personal-Data Management

Selenium WebDriver carries the last mile. The submission target is the police agency's online reporting mailbox (policemail.kcg.gov.tw/Mail.aspx). Two helper interfaces are implemented on the front end. The personal-data interface stores the reporter's identity on the local machine. It holds the basics, name, national ID, and residential address, plus contact details (email, phone, gender), and feeds them straight into the form filler. The violation-list interface pulls pending events in from an Excel spreadsheet and lets us tick rows off to delete them. It also blocks the same violation from being sent twice after multiple triggers or re-packaging.

The system touches personally identifiable information (PII) and administrative reporting at the same time. Under the Personal Data Protection Act, the auto-filled national ID and residential address count as sensitive data. They should be kept on the local machine and encrypted, and a way to delete them should be exposed by the system. Before submission, a human should still look the case over. That step keeps a misclassified report from going straight to the agency and turning into a dispute. Three items remain unmeasured. End-to-end submission success rate is one. Selector robustness against form revisions on the agency's page is another. CAPTCHA handling is the third. The observations in this chapter cover the pilot sample and one violation type. What 4.5 actually shows is narrower. On a single Pi 5, the system carries one violation from OCR

output to a populated submission form with no manual keystroke in between. The personal-data and violation-list interfaces hold the line for human review. Anything past that boundary waits on the items above.

5. CONCLUSIONS

We build an edge-AI citizen-reporting system. The implementation runs on a single Raspberry Pi 5. The system chains four stages in one pipeline. The order is detection, evidence capture, structuring, and submission.

Evidence packaging includes the Excel report and the ZIP archive. SMTP dispatches the package to a preset mailbox. Selenium fills the form automatically. The submission target is the police agency's online reporting mailbox. We add two auxiliary front-end interfaces. The "personal-data interface" manages reporter fields. The "violation-list interface" suppresses duplicate submissions. Both interfaces also support manual review before submission.

At the default 480 × 480 input, YOLOv5 violation detection averages 278.2 ms per image (n = 2 pilot). The OCR pipeline averages 1,705.6 ms per image (n = 2 pilot). The Ensemble OCR stage accounts for about 78% of the per-image latency. Two pilot images use different incidence angles. The system converges to the same plate string on both: NJN-9327. This consistency is limited to the present pilot sample.

REFERENCES

- [1] Ministry of Transportation and Communications, "Road traffic accident statistics, 2024," Road Safety Portal, Taipei, Taiwan, 2025.
- [2] Traffic Bureau, National Police Agency of Japan, "Occurrence of traffic accidents in 2024," Tokyo, Japan, 2025.
- [3] International Transport Forum, Road Safety Annual Report 2024. Paris, France: OECD/ITF, 2024.
- [4] National Police Agency, Ministry of the Interior, "National Police Statistical Yearbook, 2024," Taipei, Taiwan, 2025.
- [5] World Health Organization, Helmets: A Road Safety Manual for Decision-makers and Practitioners, 2nd ed. Geneva, Switzerland: WHO, 2024.
- [6] "Road Traffic Management and Penalty Act, Article 7-1 Citizen Reporting Clause," Laws & Regulations Database of the Republic of China (Taiwan), amended 2024.
- [7] R. R. V. e. Silva, K. R. T. Aires, and R. d. M. S. Veras, "Helmet detection on motorcyclists using image descriptors and classifiers," in Proc. 27th SIBGRAPI Conf. Graphics, Patterns and Images, Rio de Janeiro, Brazil, Aug. 2014, pp. 141-148, doi:10.1109/SIBGRAPI.2014.28.

- [8] C.-N. E. Anagnostopoulos, "License plate recognition: A brief tutorial," *IEEE Intell. Transp. Syst. Mag.*, vol. 6, no. 1, Spring 2014, pp. 59–67, doi:10.1109/MITS.2013.2292652.
- [9] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 11, Nov. 2017, pp. 2298–2304, doi:10.1109/TPAMI.2016.2646371.
- [10] 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 (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 779–788, doi:10.1109/CVPR.2016.91.
- [11] A. MK, B. Patel, S. Pardeshi, V. Rajguru, and N. More, "Intelligent system for helmet detection using Raspberry Pi," *IJSART*, vol. 3, no. 6, Jun. 2017, pp. 30–33.
- [12] V. Melchera, F. Diederichs, R. Maestre, C. Hofmann, J.-M. Nacenta, J. van Gent, D. Kusić, and B. Žagar, "Smart vital signs and accident monitoring system for motorcyclists embedded in helmets and garments for advanced eCall emergency assistance and health analysis monitoring," *Procedia Manuf.*, vol. 3, 2015, pp. 3208–3213, doi:10.1016/j.promfg.2015.07.871.
- [13] R. J. Franklin and Mohana, "Traffic signal violation detection using artificial intelligence and deep learning," in *Proc. 5th Int. Conf. Communication and Electronics Systems (ICCES)*, Coimbatore, India, Jun. 2020, pp. 839–844, doi:10.1109/ICCES48766.2020.9137873.
- [14] V. Katariya, F.-E. Jannat, A. D. Pazho, G. A. Noghre, and H. Tabkhi, "VegaEdge: Edge AI confluence for real-time IoT-applications in highway safety," *Internet of Things*, vol. 27, Jun. 2024, doi:10.1016/j.iot.2024.101268.
- [15] N. Tahilramani, P. Ahir, S. Saxena, V. P. Talreja, and P. Charanarur, "Edge-based AI solution for enhancing urban safety: Helmet compliance monitoring with YOLOv9 on Raspberry Pi," *Discov. Internet Things*, vol. 5, no. 25, Mar. 2025, doi:10.1007/s43926-025-00113-9.
- [16] S. S. Kathait, A. Kumar, S. Sawal, R. Patidar, and K. Agrawal, "Computer vision and deep learning based approach for violations due to illegal parking detection," *Int. J. Comput. Appl.*, vol. 186, no. 70, Mar. 2025, doi:10.5120/ijca2025924506.