Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 4, Issue 3, 2026

SIGNAL-IQ: AN INTEGRATED, LOW-COST, HARDWARE-VALIDATED ADAPTIVE TRAFFIC SIGNAL CONTROL SYSTEM USING YOLOV8-BASED VEHICLE AND EMERGENCY-VEHICLE DETECTION

Muhammad Rehan Yousaf[*1], Sibghatullah[2], Mairaj Ahmed Khoso[3], Sumair Kashif[4]

[*1,2,3,4]*Department of Computer Science, Institute of Space Technology, KICSIT Kahuta Campus*

[*1]mrehan249249@gmail.com

DOI: https://doi.org/10.5281/zenodo.23313118

Keywords
adaptive traffic signal control, YOLOv8, emergency vehicle preemption, hardware-in-the-loop, intelligent transportation systems, embedded systems

Article History
Received: 26 January 2026
Accepted: 11 March 2026
Published: 25 March 2026

Copyright @Author
Corresponding Author: *
Muhammad Rehan Yousaf

Abstract

Fixed-time controllers run most intersections in Pakistan and in comparable settings. They hand an approach its full green whether a queue waits behind the stop line or the lane sits empty. Infrastructure-driven adaptive platforms such as SCATS and SCOOT correct this, and they depend on detector networks whose installation and upkeep cost more than many municipal budgets allow. A large share of published reinforcement-learning controllers stops at simulation. Signal-IQ is an adaptive controller assembled entirely from off-the-shelf parts: a webcam feed, two separately trained YOLOv8 detectors, a rule-based Python controller, and an Arduino Uno R3 driving twelve LEDs arranged as four three-lamp signal heads. The controller counts vehicles inside four per-approach regions of interest, sets the green interval from the count, skips approaches with no vehicles, and hands an approach an unconditional green when the second detector reports an ambulance, fire truck, or police car. Seven use cases fix the functional scope across surveillance, detection, direction-based counting, and emergency override. A documented suite of ten test cases exercises the models, the control code, the Arduino firmware, the integrated pipeline, and four traffic scenarios, plus actuation on physical LED signal heads. On a held-out test set the general vehicle model (car, bus, truck, motorcycle) reached 0.94 precision, 0.91 recall, and 0.93 mAP@0.5. The emergency-vehicle model (ambulance, fire truck, police car) reached 0.98, 0.96, and 0.97. Measured against a fixed-time baseline with equal 30-second splits, average vehicle wait time across three simulated scenarios fell by 72.9% to 81.8%. The contribution is integration and physical validation rather than a new detector or a new control algorithm. We state the evaluation work still undone directly: comparison against a published adaptive controller, ablation of the individual control features, repeated trials with reported variance, and field deployment.

I. INTRODUCTION

Most intersections in Pakistani cities still run on fixed-time controllers. A plan built from historical counts assigns each approach a green interval, and the controller serves the full interval whether twenty vehicles queue behind the stop line or none at all [1]. Three consequences follow. Green time lands in the wrong place whenever demand across the approaches is uneven, which describes the normal condition at most junctions rather than the exception. Ambulances and fire trucks wait in the same queue as everyone else, because nothing in the controller knows they are there. And the adaptive platforms built to solve both problems, SCATS [2] and SCOOT [3] among

them, rest on inductive-loop or equivalent detector networks. The cited deployments assume detector infrastructure, communication links, and central control software are installed and maintained. For an authority working under tight budget constraints, all of this is a large and recurring commitment.

We built Signal-IQ to measure how much adaptive behaviour a university laboratory recovers from equipment already on the shelf: a webcam, a laptop with a mid-range GPU, an Arduino Uno R3, and twelve LEDs. The system passes camera frames to two YOLOv8 detectors we trained ourselves, one for ordinary vehicles and one for emergency vehicles. A Python controller turns the per-approach counts into a green interval. The Arduino turns each decision into light on a physical signal head.

The contribution is narrow, and we prefer to state it plainly. Signal-IQ introduces no detection architecture, and its control policy is a finite-state machine governed by three named constants rather than a learned policy. The work delivers four things. First, one pipeline where density-based green-time allocation, empty-lane skipping, and vision-based emergency preemption run together rather than separately. Second, two YOLOv8 models trained on a dataset compiled for this project. Third, a control rule an engineer reads in full and retunes by editing three numbers. Fourth, validation ending at a physical signal head rather than at a plot, with the whole path from frame capture to digitalWrite() documented and covered by a test suite.

II. Related Work

A. Evolution of Traffic Signal Control

Intersection control has moved through a clear sequence of generations. The earliest arrangements were manual, an officer standing in the junction, and clockwork timers replaced them. Pre-timed plans [1] formalised the practice by deriving a static green split from periodic traffic-count studies. Vehicle-actuated control then placed inductive loops at the stop line, so a phase extended or ended early depending on whether a vehicle was present. The controller responds, though only to demand already arrived.

SCATS [2] and SCOOT [3] went considerably further. Both adjust cycle length, phase split, and offset across a coordinated network of junctions from continuous detector input. Both have run at city scale in several countries. Both assume a detector network, a communications path to a central system, and staff to keep the whole arrangement working.

Fuzzy-logic control [12] took a different route. It encodes a traffic engineer's judgement as membership functions over quantities such as queue length and arrival rate. The resulting rules stay legible to a human reader, and an engineer has to tune them by hand. Reinforcement learning [13], [14], resting on the framework set out by Sutton and Barto [15], instead treats the signal as an agent selecting phases inside a Markov decision process, and the results reported in simulators such as SUMO are strong. The cost is training compute, large volumes of simulated interaction, and a policy whose internal reasoning a municipal engineer cannot readily inspect or retune for one particular junction geometry.

B. Vision-Based Vehicle Detection

Vehicle detection before deep learning leaned on handcrafted descriptors, among them Haar-like cascades [4] and histograms of oriented gradients [5]. These run on modest hardware and degrade quickly under occlusion, scale change, and shifting illumination. Convolutional networks [9], [10] moved the baseline. Region-proposal detectors, R-CNN [11] and later Faster R-CNN [8], raised localisation accuracy at a computational cost awkward for real-time work on ordinary hardware. Single-stage detectors, YOLO [6] and SSD [7], predict boxes and classes in one pass over the image. A signal controller cannot act on a detection arriving two seconds after the frame it came from, so throughput weighs more heavily here than the small accuracy margin the single-stage formulation gives up.

Two studies within the YOLO family bear directly on what we built. Bakirci [16] evaluated YOLOv8n, the smallest variant in the family, on a drone-collected dataset spanning several vehicle classes, several lighting conditions, and a range of camera altitudes. The study reports detection accuracy above 80% with inference at roughly 12 ms per frame, about 68 FPS, on an NVIDIA RTX 2080 Ti. The failure modes identified there were low light, high-altitude

viewpoints, and physically small classes. Per-class mAP@0.5 for cars stayed comfortably above 80%, while motorcycle and bicycle mAP fell substantially even under adequate lighting and dropped further when either the light or the altitude worked against the detector. We expect comparable class-dependent behaviour from our own general-vehicle model, although we did not measure per-class accuracy in this study (Section VI-D).

Omari Alaoui et al. [17] describe a YOLOv8-based system adjusting signal timing on detection of an emergency vehicle. The system processes several camera feeds so a preemption decision arrives quickly. Their reported evaluation spans a range of simulated emergency situations, and the authors name adverse weather and heavily congested traffic as conditions they had not resolved. Both limitations apply to Signal-IQ as well, and Section VI-D records them rather than treating them as settled.

C. Emergency-Vehicle Preemption

Published preemption schemes fall into three families. Acoustic detection listens for the frequency signature of a siren through roadside microphones. It needs no line of sight, and it has to compete with ambient noise. Transponder-based schemes place a GPS or RFID unit in the emergency vehicle and a receiver at the junction. The approach is dependable, and it commits an authority to equipping and maintaining every vehicle in the fleet. The third family classifies emergency vehicles directly from video already captured for traffic monitoring, so nothing has to travel with the vehicle.

Signal-IQ belongs to this third family, as does the system described in [17]. Our variation trains a separate detector for the emergency classes rather than folding them into the general model. A network responsible only for ambulance, fire-truck, and police livery spends its capacity on those cues instead of splitting it with the car-versus-bus-versus-truck problem. Whether the separation outperforms a single six-class model is a comparison we did not run, and it belongs with the ablation work listed in Section VI-D.

D. Observed Gaps in the Reviewed Work

Three patterns stand out across this body of work. We state them as observations about the literature reviewed here rather than as claims about everything published on the subject. Integration is uncommon. A given paper advances a detector, a timing scheme, or an emergency-classification model, rather than the three operating together. Physical actuation is rarely shown. Many adaptive-control evaluations end inside a simulator, and the step from a validated algorithm to wiring, firmware, and timing signals stays implicit. And the strongest-performing controllers, deep-RL policies in particular, are difficult to audit, which matters to an engineer who has to explain or adjust signal behaviour at one specific junction.

Signal-IQ takes aim at these three points. It runs all three control behaviours inside one loop, drives real LED heads from a real microcontroller, and states its policy as a small set of named thresholds, MIN_GREEN_TIME, MAX_GREEN_TIME, and TIME_PER_VEHICLE, rather than a learned function.

E. Comparative Positioning

Table I places the control basis, hardware-validation status, and principal reported limitation of each approach discussed in Sections II-A to II-C alongside Signal-IQ, so a reader checks the positioning argued above in one place. The limitation column reports what each set of authors states about their own work, not our assessment of it.

TABLE I. QUALITATIVE COMPARISON WITH APPROACHES DISCUSSED IN SECTION II

System	Control basis	HW in loop	Key reported limitation
SCATS [2]	Loop-detector network, city-scale coordination	Yes (deployed)	High infrastructure and maintenance cost

System	Control basis	HW in loop	Key reported limitation
SCOOT [3]	Loop-detector network, city-scale coordination	Yes (deployed)	High infrastructure and maintenance cost
Fuzzy-logic [12]	Rule-based membership functions	Not reported	Manual tuning of membership functions
Deep-RL [13], [14]	Learned MDP policy	Simulation only	Opaque policy logic, heavy training cost
Bakirci [16]	Vision (YOLOv8n), detection only	No	Accuracy drop for small and low-light classes
Omari Alaoui et al. [17]	Vision (YOLOv8) + EV preemption	Not reported	Adverse weather, congestion (per authors)
Signal-IQ (this work)	Vision (2× YOLOv8) + rule-based FSM + EV preemption	Yes (LED array)	Single-run results, no adaptive baseline (VI-D)

III. System Architecture

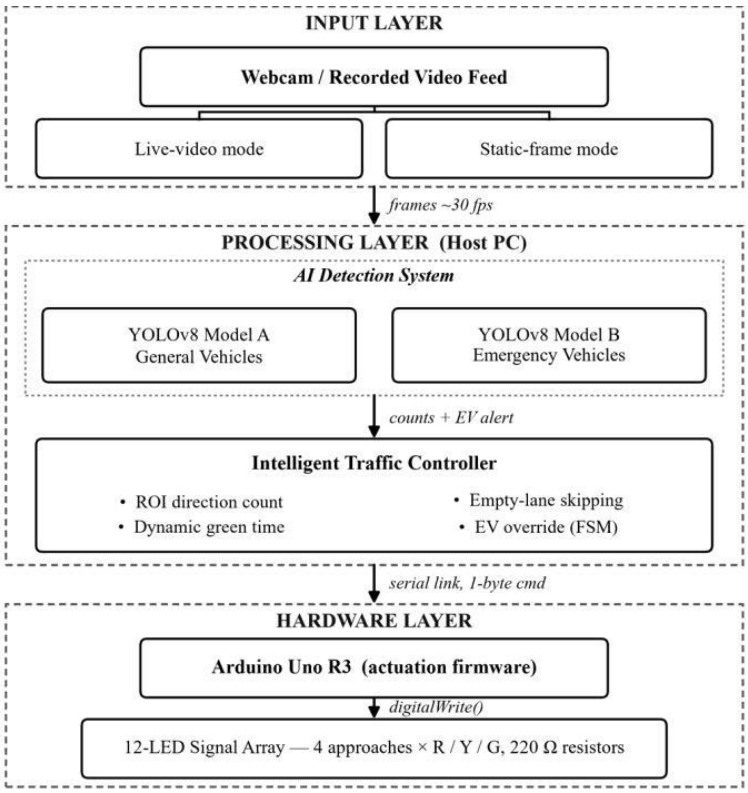


Fig. 1. Three-layer architecture of Signal-IQ: perception, decision, and physical actuation. Two independent YOLOv8 detectors consume frames from the input layer. The rule-based controller converts the per-direction counts and the emergency alert into a single-byte serial command driving the physical LED signal array.

Signal-IQ is organised in three layers, shown in Fig. 1. The input layer is a webcam or a recorded clip covering a four-way junction, and it runs in two modes: a live-video mode for normal operation, and a static-frame mode pushing individual preselected images through the pipeline so a test repeats exactly.

The processing layer runs on a host PC and holds the two parts doing the thinking. The AI Detection System produces per-approach vehicle counts and an emergency-vehicle alert from each frame. The Intelligent Traffic Controller takes the small data packet and decides which approach is served next and for how long.

The hardware layer is an Arduino Uno R3. It receives timing commands over a USB serial link and drives a twelve-LED array, four red, four yellow, and four green, arranged as one three-lamp head per approach. Every decision the controller makes appears as light on the board rather than as a line in a log file.

A. Actors and Use Cases

Two external actors appear in the design. The Normal Vehicle actor covers any non-emergency road user, private cars, motorcycles, buses, and freight trucks, whose goal is to cross the junction safely and without unnecessary delay. The system treats this actor passively. It observes, and nothing is exchanged in the other direction. Density-based timing and empty-lane skipping exist to cut this actor's waiting time.

The Emergency Vehicle actor covers ambulances, fire trucks, and police cars. Here the objective changes from optimisation to immediate and unconditional priority, and detection of this actor reconfigures the controller's behaviour through the override logic in Section IV-D.

The design specifies seven use cases around these two actors, covering the full pipeline. We wrote each one during design with its preconditions, postconditions, and execution flow, and each became a module in the implementation described in Sections IV and V. Table II summarises their functional scope.

TABLE II. SIGNAL-IQ USE CASE SUMMARY

Use case (ID)	Trigger / actor	Functional definition
Traffic Surveillance (UC-01)	Camera / operator	Captures intersection frames continuously at ~30 fps from a live feed, or from a recorded clip or static image in the alternate flow used for controlled testing, then forwards them to detection. A camera failure goes to the log and the controller falls back to a safe default state.
Detect Vehicles (UC-02)	System-internal (from UC-01)	Runs the general-vehicle YOLOv8 model on each frame, drops detections below the confidence threshold, and passes the survivors, including the valid empty-frame case, to direction counting and emergency detection.
Emergency Vehicle Detection (UC-03)	System-internal (from UC-02)	Runs the dedicated emergency-vehicle model on the same frame to find ambulances, fire trucks, and police cars. Returns a found or not-found alert with vehicle type and location, which conditionally triggers the override.
Count by Direction, N/E/S/W (UC-04)	System-internal (from UC-02)	Reduces detected bounding boxes to four per-approach counts through a point-in-polygon test against the predefined ROIs. Always returns a complete four-direction count object, even when an approach is empty.
Control Traffic Signals (UC-05)	System-internal (from UC-02)	The central decision-maker. Steps through the four approaches in order, calls dynamic-timing calculation for each, issues GREEN, YELLOW, and RED commands to the Arduino, and branches into the skip-empty-roads or emergency-override extensions when their conditions hold.

Use case (ID)	Trigger / actor	Functional definition
Calculate Dynamic Timing (UC-06)	System-internal (included by UC-05)	Applies green_time = base_time + (vehicle_count × time_per_vehicle) to an approach's count and clamps the result to the 7-second minimum and 90-second maximum.
Emergency Override (UC-07)	System-internal (extends UC-05)	On positive emergency detection, suspends the normal cycle, forces all non-priority approaches to RED and the priority approach to GREEN, holds the state until the vehicle clears, then resumes cycling. A defined priority rule applies when emergency vehicles appear on more than one approach.

B. Control-Loop Behaviour

At runtime the system works through a four-phase operational cycle. On start-up the models load into GPU memory and the serial connection to the Arduino opens, after which the controller drives every approach to red and waits there. The main loop captures a frame, runs both YOLOv8 models on it, updates the four per-approach counts, evaluates the finite-state machine of Section IV-C to choose the next phase and its duration, and sends the matching command to the Arduino. Transitions follow the computed green interval, then a yellow interval, then advancement to the next approach in sequence. The emergency-override branch of Section IV-D interrupts this cycle at any point. On shutdown every output returns to red and the serial port closes.

Counting per approach uses a point-in-polygon test. Four regions of interest, one per compass direction, are defined over the camera frame. The centre point of each detected bounding box is tested against those polygons, and four counters update on every frame. Algorithm 1 states the procedure.

Algorithm 1: Direction-wise vehicle counting

Input: detections D = {(bbox_i, class_i)}; ROI polygons R_N, R_E, R_S, R_W
Output: counts[N,E,S,W]
1: counts <- {N:0, E:0, S:0, W:0}
2: for each (bbox_i, class_i) in D do
3: if class_i is a vehicle class then
4: c <- center(bbox_i)
5: for each d in {N,E,S,W} do
6: if point_in_polygon(c, R_d) then
7: counts[d] <- counts[d] + 1
8: break
9: end if
10: end for
11: end if
12: end for
13: return counts

IV. Methodology

A. Dataset

We compiled training data from three sources so the models would not be tied to one viewpoint: the COCO dataset for broad object diversity, publicly available footage from stationary traffic cameras, and video clips we recorded ourselves under varying light (daylight, dusk, overcast) and from two camera placements (high-angle and side-view). The resulting corpus holds over 5,000 annotated images across six classes, car, bus, truck, and motorcycle, plus a separate emergency-vehicle category covering ambulance, fire truck, and police car. We annotated bounding boxes by hand in YOLO format and

split the corpus 80% training, 10% validation, and 10% test.

B. Model Training

We trained the two models independently with the Ultralytics YOLOv8 framework, 100 epochs each, using stochastic gradient descent on an NVIDIA GPU. Each run took several hours end to end. The first model handles the general vehicle classes, car, bus, truck, and motorcycle. The second sees only the emergency-vehicle subset, which lets it specialise on the features separating those vehicles from ordinary traffic, livery, light bars, and markings, instead of dividing its capacity with the general task. We computed precision, recall, and mAP on the validation split after every epoch to track convergence for both models.

C. Dynamic Timing and Control Algorithm

The controller is a finite-state machine over signal phases, for example GREEN_NORTH followed by YELLOW_NORTH, then GREEN_EAST, and so on. On entry to a green phase the controller computes the interval as

$$\text{green_time} = \text{MIN_GREEN_TIME} + (\text{vehicle_count} \times \text{TIME_PER_VEHICLE}) \quad (1)$$

MAX_GREEN_TIME sets a ceiling of 90 s and MIN_GREEN_TIME stays fixed at 7 s, long enough for one observed vehicle to clear the junction and for a pedestrian crossing. TIME_PER_VEHICLE is configurable. When an approach has no detected vehicles, the controller drops its green and yellow intervals altogether, the behaviour we refer to as empty-lane skipping, so no cycle time goes to an approach nobody is waiting on. Algorithm 2 restates the rule together with the skip.

Algorithm 2: Dynamic green-time calculation

Input: vehicle_count for current approach
Output: green_time (seconds)
raw <- MIN_GREEN_TIME + vehicle_count * TIME_PER_VEHICLE
green_time <- clamp(raw, MIN_GREEN_TIME, MAX_GREEN_TIME)
if vehicle_count = 0 then
green_time <- 0 // empty-lane skip
end if
return green_time

IEER
Institute for Excellence in Education & Research

D. Emergency-Vehicle Override Logic

A global emergency-mode flag is set as soon as the emergency-vehicle model reports a priority class on any approach. While the flag is set, the controller sends a green command to the approach and red to the other three, overriding the normal phase order. The flag clears once the detector stops reporting an emergency vehicle in frame, and density-based cycling resumes from its next scheduled approach. Algorithm 3 summarises the behaviour, including the fixed priority order applied when more than one approach reports an emergency vehicle at the same time.

Algorithm 3: Emergency-vehicle override

Input: ev_detected[N,E,S,W] from UC-03
if any ev_detected[d] = true then
d* <- approach reporting the vehicle
(priority order N>E>S>W if >1)
set all approaches to RED
set d* to GREEN
hold while ev_detected[d*] = true
on clear: resume normal cycle at
next scheduled approach
end if

V. Implementation

Inference and control run on a laptop with an Intel Core i7-9750H, an NVIDIA GTX 1660 Ti, and 16 GB of RAM, under Python 3.9. Detection runs on Ultralytics YOLOv8, capture and visualisation run on OpenCV, and PySerial carries the link to the microcontroller. The actuation unit is an Arduino Uno R3 programmed through the Arduino IDE, connected to the host over USB, and driving the twelve-LED array through 220 Ω current-limiting resistors on a solderless breadboard. Each LED anode runs through its resistor to a dedicated digital pin, and every cathode returns to a common ground rail tied to the Arduino's ground pin, so setting a pin HIGH completes the circuit and lights the lamp. Splitting the system this way mirrors a realistic deployment where a compute-capable roadside unit pairs with a cheap, field-replaceable signal driver.

A. Diagnostic and Control Interface

We built the graphical interface for ourselves as developers and for an operator, not as a public-facing dashboard, and it was the main instrument we used to check system behaviour while building the system. It has a Control Panel, a Status Bar, and a Display Area split into four panels, one per cardinal direction. The Load Model button performs a one-time load of both trained YOLOv8 files into GPU memory, with progress written to the Status Bar. Four per-direction frame-load buttons feed a single static image into one directional region of interest, which let us unit-test the counting logic on its own, away from live video. The Process Frames button starts the live loop and with it the full adaptive behaviour.

Each of the four display panels shows the annotated output of the detection system rather than the raw feed: bounding boxes, class labels, and confidence scores, next to a sub-panel giving that direction's current vehicle count, its countdown during a green phase, and its present signal state, colour-coded as red, green, or yellow. We confirmed counting, timing, and state-machine transitions visually through this interface before running the quantitative evaluation of Section VI.

B. Arduino Sketch: Pin Mapping and Serial Command Parsing

The actuation firmware follows the ordinary three-part structure of an Arduino sketch: preprocessor pin definitions, a setup() routine running once, and a loop() repeating.

Pin mapping. Each of the twelve LEDs binds to a digital I/O pin through a named #define constant, NORTH_RED_PIN, NORTH_YELLOW_PIN, NORTH_GREEN_PIN, and so on, rather than a bare pin number. Rewiring a head then means editing one line instead of hunting through the sketch. The twelve LEDs map in order by approach across the Uno's digital range, pins 2 to 13, as listed in Table III.

Table III. Arduino Digital Pin Mapping for the 12-LED Signal Array

Approach	Red pin	Yellow pin	Green pin
North	2	3	4
East	5	6	7
South	8	9	10
West	11	12	13

Setup and safe state. setup() calls pinMode() with OUTPUT on all twelve pins, opens the serial port with Serial.begin(9600), a baud rate matching the one configured in the host-side Python script, and drives every approach to red before the main loop starts. The junction therefore never shows an undefined lamp pattern between power-on and the first command from the host.

Command listening and parsing. loop() polls Serial.available() on every iteration. A value above zero means at least one unread byte is waiting, and Serial.read() consumes the byte as a single character. The character goes to an executeCommand() handler written as a switch over a small fixed alphabet. The characters 'N', 'E', 'S', and 'W' each call the matching setGreen<Direction>() function.

setGreenNorth(), for instance, calls digitalWrite(NORTH_RED_PIN, LOW), digitalWrite(NORTH_YELLOW_PIN, LOW), and digitalWrite(NORTH_GREEN_PIN, HIGH) while forcing the red pin of every other approach HIGH. The character 'X' calls setAllRed(), which serves as the controller-issued full stop and as the sketch's fallback safe state.

PySerial integration. On the host side the controller opens the link once at start-up with ser = serial.Serial('COM3', 9600), followed by a two-second time.sleep(2). The delay exists because opening a serial port toggles the board's DTR line and resets it, so the bootloader needs a moment before the input buffer becomes usable. Every timing or override decision then goes out as one byte, for example ser.write(b'N'). Keeping the protocol to a single ASCII character per command, with no framing or checksum, keeps the transmission path short enough for an override to reach the LED array in well under a second. This matches the sub-second response we observed during scenario testing (Section VI-B, TC_EMRG_001).

C. Physical Testbed and Signal-Head Assembly

The testbed is a four-way junction laid out on a black display board, with white tape for lane markings, stop bars, and pedestrian crossings, sized so one overhead webcam sees all four approaches in a single frame. Four signal heads, each carrying a red, a yellow, and a green LED, stand at the corners of the junction facing their approach. Each head's three anodes run through 220 Ω resistors to the pins listed in Table III, and all cathodes return to the common ground rail on the breadboard.

Fig. 2 shows the assembled board. Panels (a) and (b) were taken in room light, with the Arduino and breadboard visible at the edge of the board and the wiring loom running to the four heads. Panels (c) and (d) were taken with the room lights off so the lit lamps read without ambiguity. Panel (c) is the all-red state the firmware enters at power-on and returns to on the 'X' command. Panel (d) is a served phase, one approach green while the other three hold at red. Fig. 3 redraws the same layout as a labelled schematic, showing the four counting regions used by Algorithm 1 and the pin group assigned to each head.

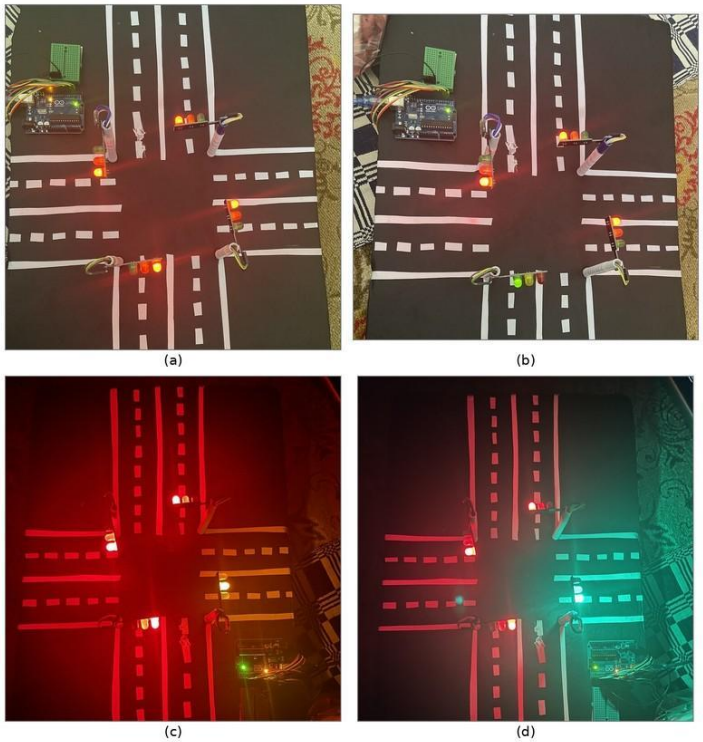


Fig. 2. The Signal-IQ bench testbed. (a) and (b) show the taped four-way junction in room light, with the Arduino Uno R3 and breadboard at the edge of the board and the wiring loom running to the four three-lamp signal heads. (c) shows the

all-red state entered at power-on and on the 'X' command. (d) shows a served phase, with one approach green while the remaining three hold at red.



Fig. 3. Schematic of the same testbed: the four counting regions used by Algorithm 1, the signal head serving each approach, and the Arduino digital pins assigned to each head (Table III). Vehicles are drawn for illustration.

D. Animated Phase Visualiser

Reading lamp states off a photograph is awkward when the point being made is a sequence rather than a single instant, so we also drew the testbed as a browser-based animation replaying the controller's phase order, the countdown on the approach being served, the empty-lane skip, and the emergency override, using the same geometry as the physical board. The animation is a presentation aid. It redraws decisions produced by the control logic of Section IV. It is not a simulator, not a second implementation of the controller, and not the source of any number reported in Section VI. Fig. 4 shows four frames from it: the all-red start-up state, a density-based green on the North approach, a skipped phase on an empty approach, and an override serving the approach carrying an ambulance.

IEER
Institute for Excellence in Education & Research

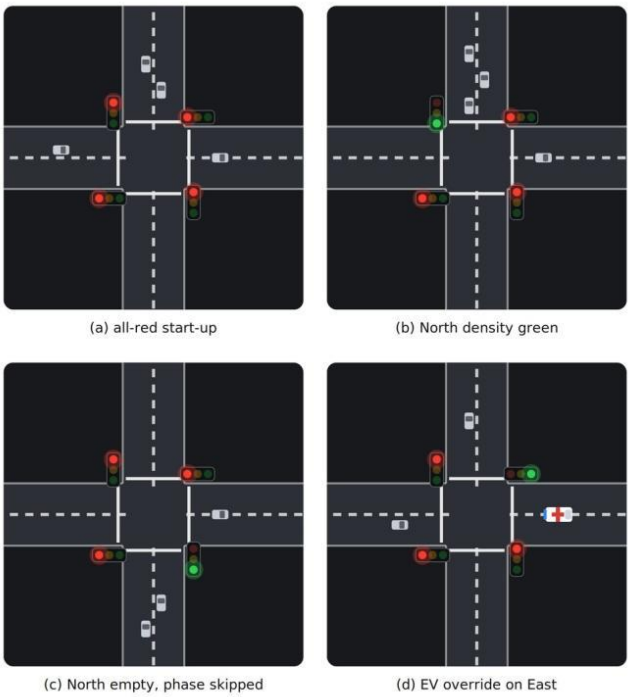


Fig. 4. Four frames from the animated phase visualiser, redrawing decisions taken by the controller of Section IV: (a) the all-red state held at start-up, (b) a density-based green interval served to the North approach, (c) an empty North approach whose green and yellow phases are skipped, with South served instead, (d) the emergency override, holding East green for an approaching ambulance while the other three approaches stay red.

E. Testing Methodology

Validation ran at three levels and appears as the test-case suite in Section VI-B. Unit testing covered the AI models, evaluating both trained networks independently on the held-out test set of Section IV-A, and the software logic, exercising the dynamic-timing calculation and the per-direction counting functions in isolation across a range of vehicle counts, including the boundary cases at MIN_GREEN_TIME and MAX_GREEN_TIME. Hardware unit testing exercised the Arduino sketch on its own through the IDE's Serial Monitor. We typed direction commands directly and checked the right LED lit before connecting any software to it.

Integration testing then checked the serial link between the Python controller and the Arduino with a dedicated script walking through each directional command with a fixed pause, which exercised the PySerial transmission path and the Arduino's parsing under realistic message timing. System-level testing ran the complete pipeline against a defined baseline traffic condition and against edge cases built to stress the adaptive logic: a high-density scenario to confirm MAX_GREEN_TIME caps correctly, an empty-lane scenario to confirm the skip, and an emergency-vehicle scenario to confirm the override activates and releases. A long-duration run watched for memory growth, dropped serial links, and state-machine deadlock over extended operation.

F. Bill of Materials

Table IV gathers the physical components introduced in Sections V-A to V-C into a single bill of materials. Every item is a generic part available from a hobby electronics supplier. None of it is traffic-control hardware.

TABLE IV. SIGNAL-IQ HARDWARE BILL OF MATERIALS

Component	Qty	Role
Arduino Uno R3	1	Actuation controller and firmware host
LED (red / yellow / green)	12	Signal heads, 4 approaches × 3 colours
220 Ω resistor	12	Current limiting, one per LED
Solderless breadboard	1	LED array mounting
USB cable	1	Host–Arduino serial link at 9600 baud
Webcam / video source	1	Vision input (Section III)
Host PC (i7-9750H, GTX 1660 Ti, 16 GB)	1	AI inference and control (Section V)

VI. RESULTS AND EVALUATION

A. Perception Performance

Table V reports test-set performance for the general vehicle-detection model. Precision of 0.94 means false positives are rare, which matters here in a specific way. A phantom detection buys green time for an approach with no vehicle on it. Recall of 0.91 means roughly nine vehicles in a hundred present in a frame go missed, and mAP@0.5 of 0.93 reflects both correct classification and reasonable box placement.

TABLE V. GENERAL VEHICLE DETECTION MODEL: TEST SET PERFORMANCE

Metric	Result
Precision	0.94
Recall	0.91
mAP@0.5	0.93

Table VI reports the same three figures for the emergency-vehicle model. Precision (0.98) and recall (0.96) both sit above the general model's, which is what training a smaller and visually distinctive class set on its own should produce. For preemption the recall figure carries the safety weight, since a missed ambulance produces no override at all.

TABLE VI. EMERGENCY VEHICLE DETECTION MODEL: TEST SET PERFORMANCE

Metric	Result
Precision	0.98
Recall	0.96
mAP@0.5	0.97

B. Documented Test Case Suite

Beyond the accuracy figures above, validation appears as a documented suite of test cases spanning five levels: model accuracy, software-logic correctness, hardware-level command processing, full-system integration, and scenario and stability behaviour. Table VII lists them. The suite puts the unit, integration, and system-level methodology of Section V-E on paper. It gives every functional claim made elsewhere in this paper, dynamic timing, empty-lane skipping, emergency override, and long-run stability, an explicit pass or fail criterion a reader re-runs.

TABLE VII. DOCUMENTED TEST CASE SUITE

Test ID	Level	Module / focus	Pass criterion
TC_AI_001	Unit (AI)	Both trained YOLOv8 models on the held-out test set	Accuracy meets the targets reported in Tables V and VI
TC_SW_001	Unit (software)	calculate_dynamic_timing() across a range of counts, boundaries included	Returned green time always falls within 7–90 s
TC_HW_001	Unit (hardware)	Arduino direction-command processing through the IDE Serial Monitor	The correct directional LED lights for each of N/E/S/W
TC_SYS_001	System	Complete integrated pipeline against all functional requirements	All functional requirements satisfied
TC_ENV_001	Environment	Physical testbed: webcam, toy-car rig, monitor for video playback	All test equipment works correctly
TC_BASE_001	Scenario	Moderate, balanced traffic on all four approaches	System operates within expected normal parameters
TC_DENSE_001	Scenario	High density on one approach, low on the others	Extended green goes to the dense approach
TC_SKIP_001	Scenario	Heavy traffic on two approaches, none on the other two	Empty approaches receive no green or yellow phase
TC_EMRG_001	Scenario	Ambulance-approach footage, detection-to-green response timed	Override activates immediately, response time recorded

IEER
Institute for Excellence in Education & Research

Test ID	Level	Module / focus	Pass criterion
TC_STAB_001	Stability	Looped traffic video over a multi-hour run	No memory growth, serial drops, or performance loss

C. Comparative Controller Evaluation

We compared Signal-IQ against a fixed-time controller granting equal 30-second splits, across three simulated traffic conditions: tidal flow, meaning heavy demand in one direction during a simulated rush hour, an unbalanced intersection, a major arterial crossing a minor side street, and off-peak random arrivals, sparse and sporadic traffic on all four approaches. We logged average vehicle wait time as the comparison metric, reported in Table VIII.

TABLE VIII. AVERAGE VEHICLE WAIT TIME: SIGNAL-IQ VS. FIXED-TIME CONTROLLER

Scenario	Fixed-time (s)	Signal-IQ (s)	Reduction
Tidal flow	85.4	22.1	74.1%
Unbalanced intersection	68.2	18.5	72.9%
Off-peak random	45.7	8.3	81.8%

The per-scenario behaviour is worth spelling out, because the mechanism differs in each case. Under tidal flow the fixed-time baseline gave the heavily loaded approach the same 30 seconds it gave its three quiet neighbours, and the queue on the dominant direction grew accordingly. Signal-IQ pushed the approach toward the 90-second ceiling while holding the others at the 7-second floor. On the unbalanced intersection the baseline under-served the arterial and over-served the side street. Signal-IQ gave the arterial long continuous greens and served the side street only when the detector found vehicles on it. In the off-peak case the baseline kept cycling through all four 30-second phases even where an approach sat empty, so a single arriving vehicle waited up to 90 seconds for its turn. Empty-lane skipping instead held the junction at all-red until something appeared and then served it at the minimum green. This is why the largest relative improvement, 81.8%, shows up in the sparsest scenario rather than the busiest one.

Across all three patterns the adaptive controller produced lower average wait times than the fixed-time baseline, by margins varying with the traffic pattern as Table VIII shows. These figures are simulation results obtained with the controller in the loop. They describe behaviour under the modelled arrival patterns, and a reader should not read them as a prediction of the same percentages at a real junction, where driver behaviour, pedestrian demand, turning movements, and detector error all enter.

D. Threats to Validity

We prefer to report the boundaries of this evaluation rather than have a reviewer find them. (1) Baseline. The comparison runs against a fixed-time controller we implemented ourselves in simulation, not against a published adaptive method such as a fuzzy-logic or RL controller from Section II-A. How Signal-IQ compares with other adaptive approaches therefore stays untested. (2) Statistical rigour. The suite in Table VII establishes functional correctness, and the wait-time numbers in Table VIII come from a single run per scenario. We report no repeated trials, variance, or confidence intervals. (3) Ablation. Table VIII evaluates the complete system. We have not separated how much of the improvement comes from density-based timing, how much from empty-lane skipping, and how much from preemption. (4) Deployment. Evaluation is confined to simulated traffic scenarios and to LED actuation on a bench testbed. No live or semi-live intersection trial has run. (5) Dataset. Class balance, the licensing terms of the scraped video

sources, and performance under adverse weather are not fully characterised. Closing these five gaps is the future-work agenda in Section VII, and we treat it as the main outstanding work rather than an afterthought.

E. Discussion

The precision and recall figures of Section VI-A carry different weight for the two models, because the two models fail in different directions. For the general-vehicle model the costly error is a false positive. It grants green time to an approach with nothing on it, and does so at the expense of an approach with vehicles waiting. Precision of 0.94 keeps the error infrequent. For the emergency-vehicle model the costly error is a false negative, since a missed ambulance receives no priority at all. This is why 0.96 recall rather than 0.98 precision is the figure mattering for the model's role.

This asymmetry connects to the class-dependent error pattern Bakirci [16] reported and we discussed in Section II-B. Large, geometrically distinct classes, cars, buses, and ambulances with light bars and livery, are detected more reliably than small or visually ambiguous ones such as motorcycles and bicycles. We did not evaluate our general-vehicle model per class, so whether the model shows the same weakness on motorcycles is an open question rather than a finding. We flag it as a specific and testable extension of the per-class analysis Bakirci carried out, not as something transferring by assumption.

Covering daylight, dusk, and overcast conditions from both high-angle and side-view placements (Section IV-A) was intended to give the general model some tolerance of the viewpoint and illumination variation a fixed roadside camera meets in practice. We did not stress-test the coverage against the adverse weather and heavy congestion Bakirci [16] and Omari Alaoui et al. [17] separately identify as unresolved for YOLOv8-based traffic vision. This limitation is shared across that line of work rather than peculiar to Signal-IQ, and Section VI-D records it as inherited rather than solved.

Everything reported here concerns one four-approach junction. How empty-lane skipping and density-based timing behave at junctions with a different number of approaches, or under the coordination effects multi-intersection green-wave control introduces, is not established by this evaluation. The finite-state controller of Section IV-C generalises structurally to any number of approaches, but that is an architectural property, not a demonstrated result.

VII. Conclusion and Future Work

Signal-IQ combines YOLOv8-based vehicle and emergency-vehicle detection with a transparent rule-based controller and an Arduino-driven actuation layer, at a component cost a university department or a small municipality absorbs. Seven use cases fixed the functional scope (Section III-A), and a documented suite of ten test cases checked it, covering model accuracy, software logic, hardware commands, full-system integration, and scenario and stability behaviour (Section VI-B), alongside actuation on the physical LED signal heads described in Sections IV and V. On the held-out test set the general and emergency-vehicle models reached mAP@0.5 of 0.93 and 0.97 respectively. Across three simulated scenarios the controller cut average wait time by 72.9% to 81.8% relative to a fixed-time baseline.

Read the contribution as system integration and physical validation rather than algorithmic novelty. Density-based timing, empty-lane skipping, and vision-based emergency preemption have each been studied before. What this work adds is a single inspectable pipeline running all three and ending at a physical signal head, documented from the use-case specification down to the pin mapping and the serial command alphabet, so another group rebuilds it.

With integration and hardware-in-the-loop actuation now in place, what remains concerns the breadth and rigour of the evaluation rather than missing functionality. In order of priority, future work should benchmark Signal-IQ against a published adaptive baseline, fuzzy-logic or RL, rather than against fixed-time control alone, run an ablation isolating the wait-time contribution of each control feature, repeat the scenario trials and report variance, run a limited pilot at a live or semi-live junction, and extend the controller to multi-intersection coordination and to a dataset covering adverse weather and a wider range of lighting.

APPENDIX: DETAILED USE-CASE SPECIFICATIONS

Table II summarised the functional scope of the seven use cases at the level the main text needs. This appendix reproduces the precondition, postcondition, and flow format used during design for the three use cases most central to the adaptive behaviour, so a reader inspects the design-time documentation directly rather than seeing it cited alone.

TABLE IX. USE-CASE SPECIFICATION: UC-05, CONTROL TRAFFIC SIGNALS

Element	Specification
Precondition	The AI Detection System (UC-02) has produced a valid four-direction vehicle-count object for the current frame, and the serial connection to the Arduino is open.
Postcondition	Exactly one approach is GREEN, or every approach is RED during the all-red default or transition state, and the Arduino has acted on the matching command.
Main flow	1. Select the next approach in the fixed N→E→S→W order. 2. Call Calculate Dynamic Timing (UC-06) for the approach's count. 3. Issue GREEN for the computed duration, then YELLOW for the fixed inter-phase interval, then RED. 4. Advance to the next approach and repeat.
Alternate flow	If the selected approach has no vehicles, branch to skip-empty-roads (Section IV-C) instead of step 2. If an emergency vehicle appears at any point, branch immediately to Emergency Override (UC-07), suspending the current phase.

TABLE X. USE-CASE SPECIFICATION: UC-06, CALCULATE DYNAMIC TIMING

Element	Specification
Precondition	UC-05 has selected an approach and supplied its current vehicle count.
Postcondition	A green-time value in the closed interval [MIN_GREEN_TIME, MAX_GREEN_TIME] = [7, 90] seconds returns to UC-05.
Main flow	1. Compute raw = MIN_GREEN_TIME + (vehicle_count × TIME_PER_VEHICLE). 2. Clamp raw to the [7, 90] second range. 3. Return the clamped value (Algorithm 2, Section IV-C).
Alternate flow	If vehicle_count = 0, return a green_time of 0 seconds instead, which tells UC-05 to skip this approach's green and yellow phases entirely (empty-lane skipping).

TABLE XI. USE-CASE SPECIFICATION: UC-07, EMERGENCY OVERRIDE

Element	Specification
Precondition	Emergency Vehicle Detection (UC-03) has returned a positive alert identifying the approach carrying the vehicle.
Postcondition	The identified approach has been served an unbroken GREEN interval for the whole time the emergency vehicle remained in frame, and normal cycling has resumed from the next scheduled approach.
Main flow	1. Set the global emergency-mode flag. 2. Force RED on every approach except the priority one. 3. Set the priority approach to GREEN. 4. Hold while UC-03 keeps reporting the vehicle in frame. 5. On clear, unset the flag and resume UC-05 at its next scheduled approach (Algorithm 3, Section IV-D).

Element	Specification
Alternate flow	If UC-03 reports emergency vehicles on more than one approach at once, the fixed priority order N>E>S>W selects which one is served first. The remaining approaches are served in the same order once the first clears.

Declarations

Ethics approval and consent to participate: Not applicable.

Consent for publication: Not applicable.

Availability of data and materials: The datasets generated and analysed during the current study are available from the corresponding author on reasonable request.

Competing interests: The authors declare that they have no competing interests.

Funding: This research received no external funding.

Authors' contributions: MRY served as technical lead, integrated the vision, control, and actuation layers, and designed and ran the system-level test programme. S developed the Arduino firmware, the twelve-LED signal array and its driver circuitry, and the serial actuation link. MAK led the core research, including the literature survey, the compilation and annotation of the training dataset, and the design of the density-based timing and emergency-preemption policies. SK developed the Python control application, the YOLOv8 inference pipeline, and the diagnostic interface. All authors read and approved the final manuscript.

Acknowledgements: Not applicable.

References

P. Koonce et al., Traffic Signal Timing Manual. Federal Highway Administration, Office of Operations, Pub. No. FHWA-HOP-08-024, 2008.

P. R. Lowrie, SCATS: A Traffic Responsive Method of Controlling Urban Traffic. Roads and Traffic Authority, NSW, Australia, 1990.

P. B. Hunt, D. I. Robertson, R. D. Bretherton, and R. I. Winton, SCOOT—a traffic responsive method of coordinating signals. TRRL Lab. Rep. LR 1014, Transport and Road Research Laboratory, 1981.

P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Proc. IEEE CVPR, vol. 1, 2001, pp. I-511–I-518.

N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in Proc. IEEE CVPR, vol. 1, 2005, pp. 886–893.

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE CVPR, 2016, pp. 779–788.

W. Liu et al., "SSD: Single shot multibox detector," in Proc. ECCV, Cham: Springer, 2016, pp. 21–37.

S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in Adv. Neural Inf. Process. Syst. (NIPS), vol. 28, 2015.

A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Adv. Neural Inf. Process. Syst. (NIPS), vol. 25, 2012.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proc. IEEE CVPR, 2014, pp. 580–587.

IEER Institute for Excellence in Education & Research

C. P. Pappis and E. H. Mamdani, "A fuzzy logic controller for a traffic junction," IEEE Trans. Syst., Man, Cybern., vol. SMC-7, no. 10, pp. 707–717, Oct. 1977.

L. Wei, W. Zhang, H. Wang, and H. Liu, "Research on urban traffic signal control with deep reinforcement learning," J. Adv. Transp., vol. 2018, Art. ID 9851534, 2018.

B. Abdulhai, R. Pringle, and G. J. Karakoulas, "Reinforcement learning for true adaptive traffic signal control," J. Transp. Eng., vol. 129, no. 3, pp. 278–285, May 2003.

R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

M. Bakirci, "Real-time vehicle detection using YOLOv8-Nano for intelligent transportation systems," Traitement du Signal, vol. 41, no. 4, pp. 1727–1740, Aug. 2024.

A. Omari Alaoui, M. Oumoulylte, H. Oubalahcen, M. K. Boutahir, A. Hessane, and A. El Allaoui, "Real-time traffic signal adjustment using YOLOv8 for improved integration of emergency vehicles in smart traffic systems," Signal, Image and Video Processing, vol. 19, art. 576, 2025.

M. R. Yousaf, M. A. Abbas, and Sibghatullah, "Signal-IQ: An Adaptive, AI-Driven Traffic Signal Control System with Hardware-in-the-Loop Validation," B.S. thesis, Dept. Comput. Sci., Institute of Space Technology, KICSIT Kahuta Campus, 2025.



Muhammad Rehan Yousaf is pursuing the B.S. degree in computer science at the Institute of Space Technology, KICSIT Campus, Islamabad, Pakistan. He served as technical lead for Signal-IQ, responsible for integrating the vision, control, and actuation layers into one working pipeline and for the system-level test programme reported in this paper. His research interests include computer vision, intelligent transportation systems, and real-time embedded software.



Sibghatullah is pursuing the B.S. degree in computer science at the Institute of Space Technology, KICSIT Campus, Islamabad, Pakistan. He handled the hardware layer of Signal-IQ, covering the Arduino Uno R3 firmware, the twelve-LED signal array and its driver circuitry, and the serial actuation link between the host controller and the physical signal heads. His research interests include embedded systems, microcontroller programming, and hardware-in-the-loop validation.



Mairaj Ahmed Khoso is pursuing the B.S. degree in computer science at the Institute of Space Technology, KICSIT Campus, Islamabad, Pakistan. He led the core research for Signal-IQ, including the literature survey, the compilation and annotation of the training dataset, and the design of the density-based timing and emergency-preemption policies evaluated in this paper. His research interests include machine learning, object detection, and adaptive traffic signal control.



Sumair Kashif is pursuing the B.S. degree in computer science at the Institute of Space Technology, KICSIT Campus, Islamabad, Pakistan. He handled software development on Signal-IQ, implementing the Python control application, the YOLOv8 inference pipeline, and the diagnostic interface used to validate the counting, timing, and state-machine behaviour of the controller. His research interests include software engineering, deep learning frameworks, and real-time systems.