BSCS FINAL PROJECT

Design and Test Specification

AI-Driven Traffic Flow Optimizer



Project Advisor

**MISHA ASIF**

Presented by:

**Group ID: F24BS175**

Student Reg# Student Name

**L1F21BSCS0987 SULMAN AHMAD**

**L1F21BSCS1208 ANEEQ-UR-REHMAN L1F21BSCS0864 INAYAT NAVEED**

**Faculty of Information Technology & Computer Science**

**University of Central Punjab**

Design and Test Specification

SDP Phase III

AI-Driven Traffic Flow Optimizer

Advisor: Misha Asif

Team F24BS175

|  |  |
| --- | --- |
| Member Name | Primary Responsibility |
| SULMAN AHMAD | Software and Hardware Integration |
| ANEEQ-UR-REHMAN | AI-Model Training |
| INAYAT NAVEED | Documentation, Algorithm design |

Table of Contents

Table of Contents i

Revision History ii

Previous Phases Feedback ii

Abstract 1

1. Introduction 2

1.1 Product 2

1.2 Background 2

1.3 Objective(s)/Aim(s)/Target(s) 3

1.4 Scope 3

1.5 Business Goals 3

1.6 Document Conventions 3

1.7 Miscellaneous 4

2. Technical Architecture 5

2.1 Application and Data Architecture 5

2.2 Component Interactions and Collaborations 8

2.3 Design Reuse and Design Patterns 11

2.4 Technology Architecture 11

2.5 Architecture Evaluation 12

3. Detailed/Component Design 13

3.1 Component-Component Interface 15

3.2 Component-External Entities Interface 15

3.3 Component-Human Interface 16

4. Screenshots/Prototype 17

4.1 Workflow 17

4.2 Screens 17

4.3 Additional Information 18

5. Other Design Details 19

6. Test Specification and Results 20

6.1 Test Case Specification 20

6.2 Summary of Test Results 23

7. Revised Project Plan 24

8. References 25

Appendix A: Glossary 26

Appendix B: IV & V Report 27

Semester wise SDP Meeting Report 28

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

Previous Phases Feedback

**Idea Defense Feedback (Screenshot)**

A black and white document

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.**Phase 2 (SDS) Feedback (Screenshot)**

# Abstract

Urban traffic congestion during peak hours and emergencies demands intelligent solutions. Our AI-Driven Traffic Flow Optimizer combines deep learning with real-time control to improve intersection management. The system uses YOLOv8 for fast vehicle and pedestrian detection, while ESP32 microcontrollers instantly adjust traffic signals.

We developed this solution using Arduino IDE (v2.3.6) with ESP32 support (v3.2.0) and Python 3.13.3 for AI processing. The system performs local edge analysis to identify regular traffic and emergency vehicles, then sends priority signals to controllers. A web dashboard enables real-time monitoring and manual overrides when needed.

This report details the design, development, and validation, demonstrating improved traffic flow and emergency vehicle priority at intersections.

# Introduction

Modern cities face complex traffic challenges as vehicle numbers grow and emergency response needs intensify. Existing traffic lights (using fixed timers or basic sensors) cannot adapt to real-time situations like approaching ambulances or sudden traffic buildups. This solution bridges this gap by combining AI vision with responsive signal control.

At its core, the system uses YOLOv8 AI to accurately detect vehicles (cars, trucks, motorcycles) and emergency vehicles in real time. The Python-powered analysis runs locally, monitoring traffic density and prioritizing emergency responders. Detection data feeds to an ESP32 microcontroller (programmed in C++ via Arduino IDE), which instantly adjusts traffic lights through GPIO outputs.

We tested multiple ESP32 boards for real-world reliability, with CH340 drivers enabling stable communication. The system offers both automatic AI control and manual override via a local Flask dashboard, giving operators live monitoring and emergency intervention capabilities.

Unlike theoretical models, this end-to-end solution actually connects video analysis to physical traffic lights, creating intersections that dynamically respond to real conditions while ensuring emergency vehicles get priority passage.

## Product

The AI-Driven Traffic Flow Optimizer is a hardware-based solution designed to address urban traffic congestion by integrating Arduino microcontrollers, YOLOv8-powered AI algorithms, and high-resolution cameras. This system processes real-time traffic data to dynamically adjust signal timings, reducing congestion and improving traffic flow. A key feature is the prioritization of emergency vehicles, such as ambulances, to clear their paths quickly. Additionally, the system includes a web-based control panel, allowing traffic operators to monitor traffic conditions and system performance. This solution combines AI optimization, advanced hardware, and an intuitive interface to provide a scalable and cost-effective approach to modern traffic management challenges.

## Background

Traditional traffic management systems that rely on static light systems are often inadequate for responding to real-time traffic conditions. While adaptive systems are available, many of these solutions are either limited in their functionality or prohibitively expensive for widespread adoption.

Existing approaches, such as the Automated Traffic Control System (ATCS), which leverages computer vision, aim to enhance traffic flow but fall short in delivering real time adaptability. Other solutions, like TrueSec (Trucking Surveillance System) and Automated License Plate Recognition systems, address specific issues such as truck monitoring or traffic violation detection without providing comprehensive traffic management optimization. Similarly, Accident Detector Systems primarily focus on emergency response, while Digital Traffic Wardens depend on human intervention to enforce regulations.

Our AI-Driven Traffic Flow Optimizer bridges these gaps by integrating YOLOv8, for real-time identification of vehicles. By incorporating cost-effective hardware, such as Arduino microcontrollers, this solution offers a scalable and affordable alternative. The system's web-based control panel further enhances its utility by enabling operators to monitor traffic, make adjustments, and access diagnostic insights, thereby providing flexibility that goes beyond fully automated systems. As a result, this solution presents a practical and dynamic replacement for outdated fixed-timing signals.

## Objective(s)/Aim(s)/Target(s)

The primary objective of this project is to develop and deploy a real-time object detection system capable of identifying vehicles such as cars, trucks, buses, motorcycles and emergency vehicles like ambulances. The system aims to enhance emergency response by implementing signal pre-emption, dynamically adjusting traffic lights to provide uninterrupted passage for emergency vehicles. Additionally, it seeks to optimize overall traffic flow by modifying signal timings based on live traffic conditions, ensuring efficient intersection management and reduced congestion.

## Scope

The project focuses on deploying a traffic optimization system that enhances real-time traffic management utilizing object detection technology for vehicle and pedestrian recognition, prioritizing emergency vehicles through signal pre-emption, and dynamically adjusting traffic signals based on live traffic conditions. Additionally, an intuitive operator interface will be provided to enable efficient real-time monitoring and management of traffic flow. The product's focus is on improving overall traffic efficiency, reducing congestion, and enhancing emergency vehicle response times.

## Business Goals

This traffic management platform significantly reduces the reliance on manual monitoring, leading to lower operational costs for municipal authorities. By prioritizing emergency vehicles, it helps to minimize response times, ultimately saving lives. Furthermore, the platform enhances the efficiency of transportation networks by decreasing delays, streamlining the movement of goods and services, and boosting overall business productivity. In alignment with the nation’s smart city objectives, it integrates advanced AI technologies into urban infrastructure, enabling more informed and data-driven decision-making processes.

## Document Conventions

In this Report, the following standards and typographical conventions have been applied:

* Bold: Used for section titles, key terms, and significant concepts within the document.
* Italicized: Used to highlight code snippets.
* Headings and Subheadings:
* A clear numbering system is used for sections and subsections (e.g., 1.6, 2.3) to maintain logical flow.
* Acronyms are spelled out in full with the acronym in parentheses
* Figures, tables, and diagrams are sequentially numbered and referenced by their specific number (e.g., Figure 2.1, Table 4.2).

## Miscellaneous

To ensure the smooth execution of the project, collaboration was maintained through communication platforms and task management tools. Key documents, including the SRS, SDS, and DTS report, were regularly reviewed and updated to reflect evolving design decisions and implementation milestones. Testing and validation phases focused on system performance, detection accuracy, and emergency response effectiveness. Feedback from traffic authorities and potential end-users was incorporated to guide iterative improvements. All essential hardware components including Arduino microcontrollers, ESP32 boards, and high-resolution cameras were procured and deployed as scheduled. Contingency measures were successfully implemented to address minor technical and logistical challenges encountered during the development process.

# Technical Architecture

The architecture is designed as a distributed, multi-layer architecture that seamlessly merges high-level computer vision tasks with low-level real-time control mechanisms. The structure involves input sensors (cameras), an AI processing engine (YOLOv8), embedded controllers (ESP32), actuator modules (traffic lights and digital displays), and a web-based dashboard interface for human supervision.

The system follows a client-server architecture with the following major subsystems:

* Data Acquisition and Processing Layer (Camera + YOLOv8 on Python)
* Decision and Communication Layer (Flask-based API server)
* Embedded Execution Layer (ESP32 running Arduino C++)
* Display Interface Layer (Web GUI for manual override)

**System Overview**

The input layer is composed of high-resolution cameras positioned at intersection corners. The captured frames are processed by a Python script running the YOLOv8 model, which uses ultralytics and opencv-python libraries for frame decoding, object detection, and bounding box extraction. Based on the results (vehicle count, emergency vehicle detection, pedestrian presence), a decision matrix is computed.

The decision is transmitted to the ESP32 Dev Module, programmed via the Arduino IDE (v2.3.6) using the Espressif 3.2.0 board package. The controller interprets incoming commands to switch the red, yellow, or green lights via GPIO pins.

Below is a high-level explanation of the technology layers used in your system:

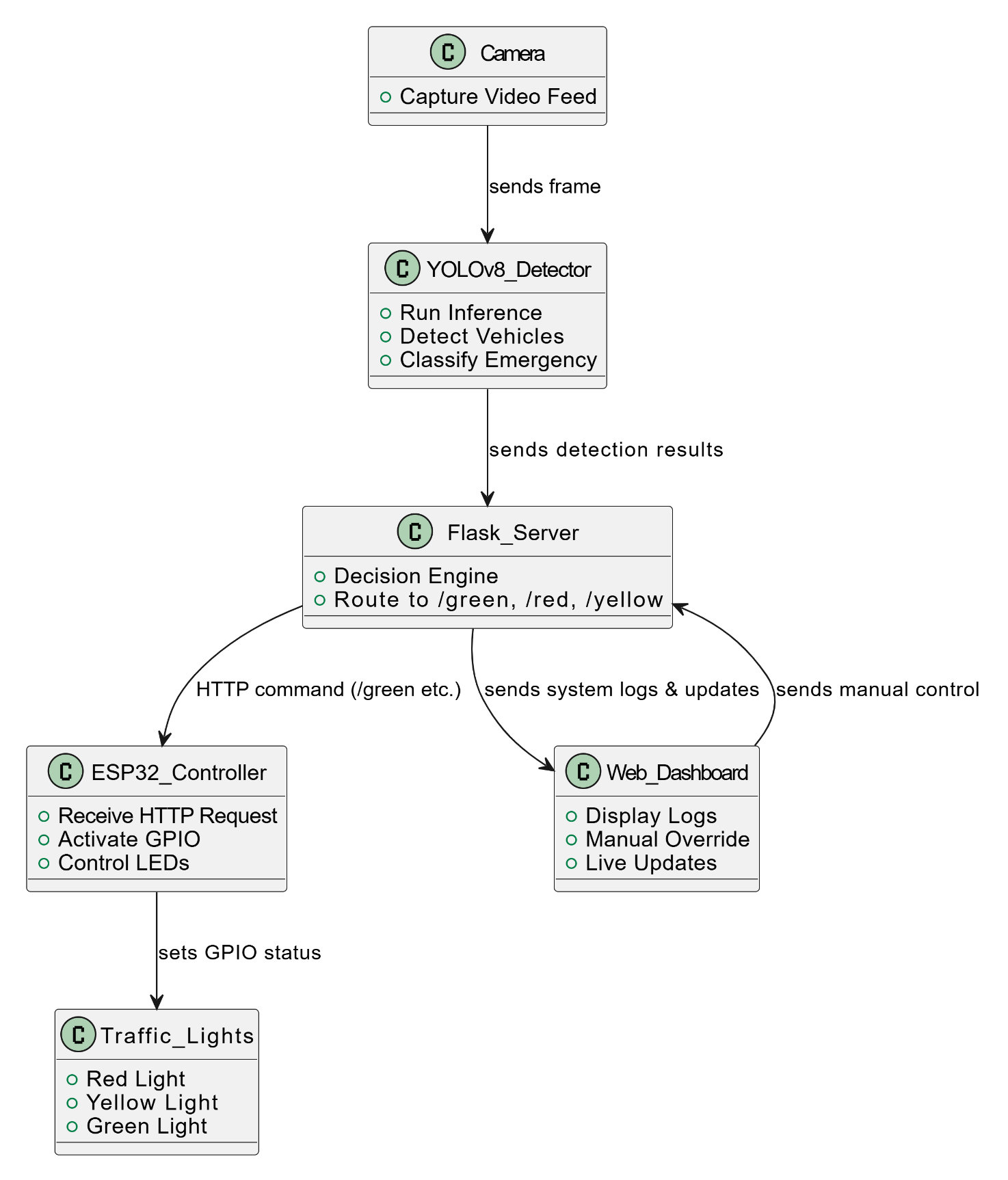
## Application and Data Architecture

The data architecture begins with a Python-based detection pipeline. The app01.py script reads video frames from a USB or IP camera using OpenCV, loads the yolov8n.pt weights, and runs object detection in real-time. The system distinguishes between common vehicle types (car, truck, motorcycle, bus), and uses trained bounding box labels to detect emergency vehicles such as ambulances.

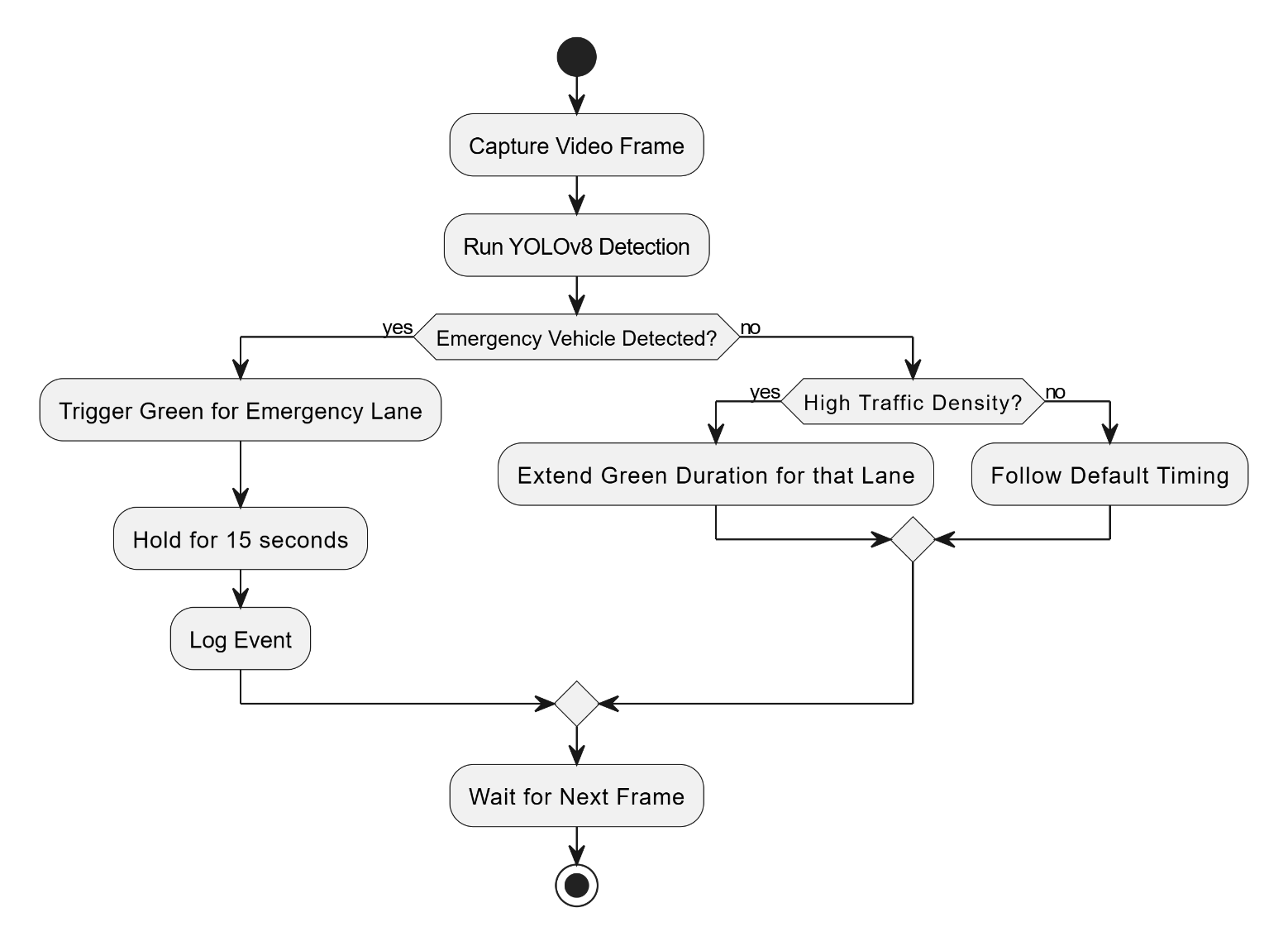
The decision logic is encapsulated in conditional branches within the Python server. If an emergency vehicle is detected with confidence above a threshold, the logic immediately changes the signal flow by dispatching a REST command to the ESP32.

The ESP32 module, connected through either Wi-Fi, processes incoming REST commands. Upon receiving a signal priority request, the embedded firmware activates the corresponding traffic light sequence, enabling the green signal while deactivating the red signal, simultaneously initiating a 15-second countdown displayed on the display.

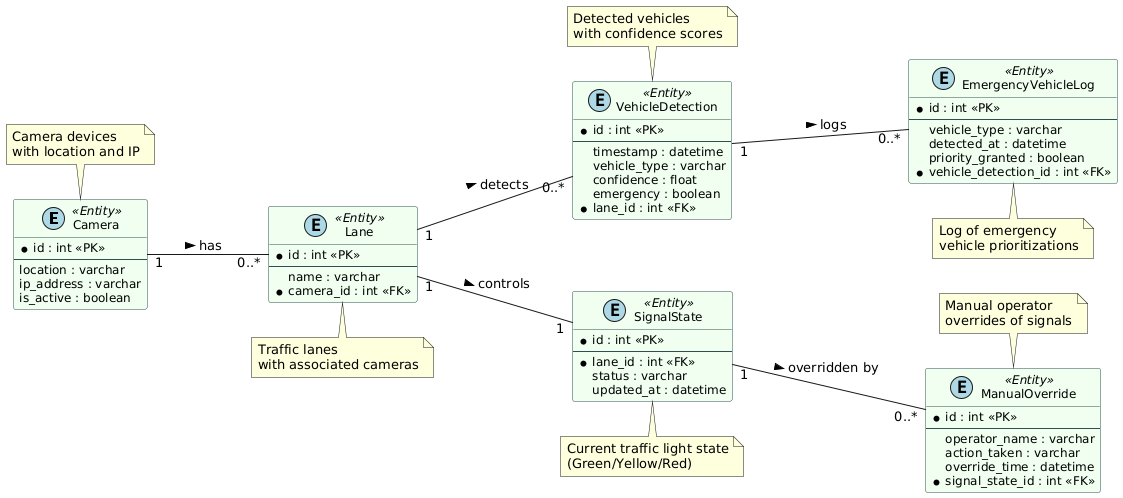
All system parameters are centrally managed within the Arduino-compatible firmware file, implemented using standard Arduino C++ syntax. This approach ensures maintainability while providing straightforward adjustment of operational parameters.



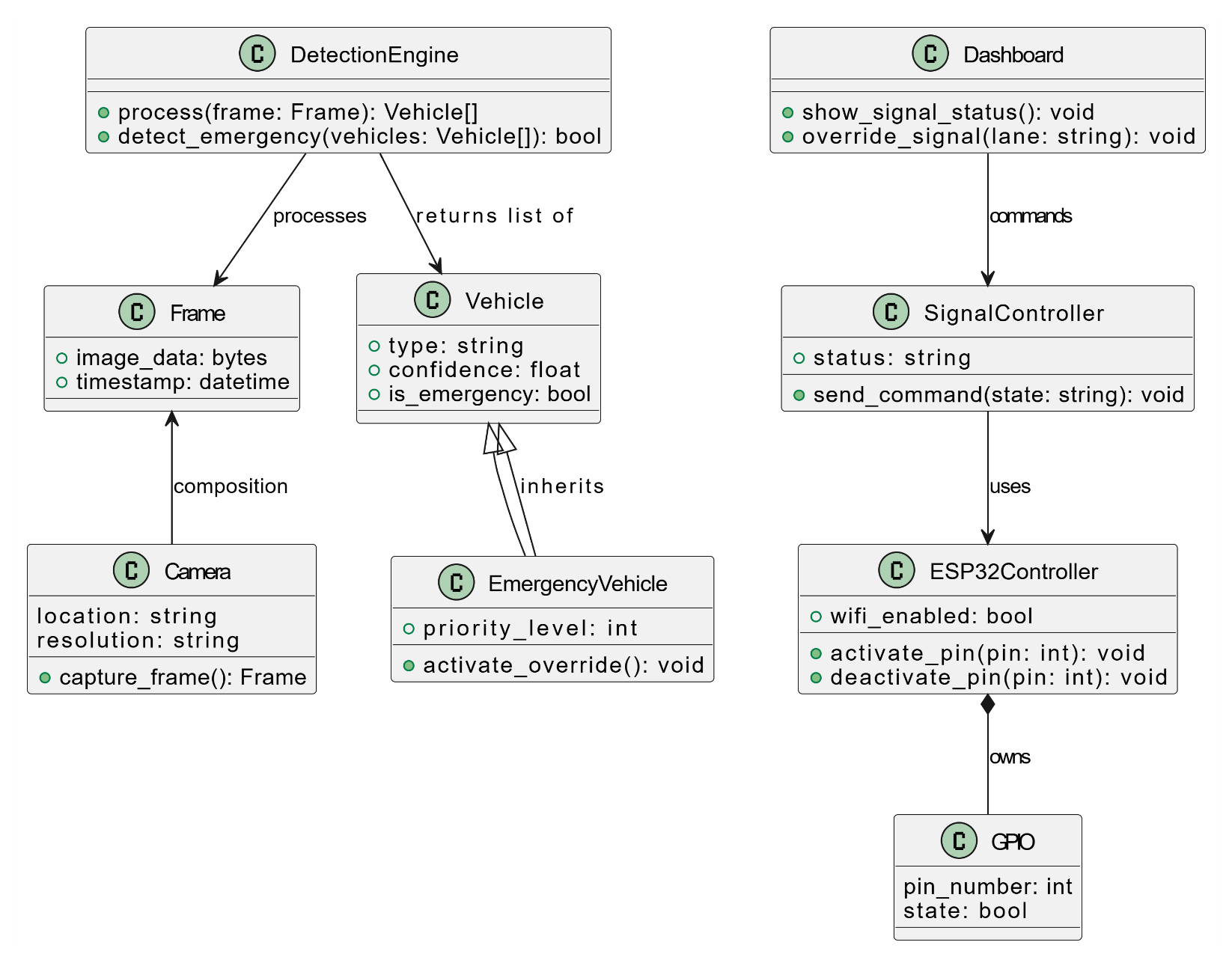
2.1.1. COMPONENT DIAGRAM



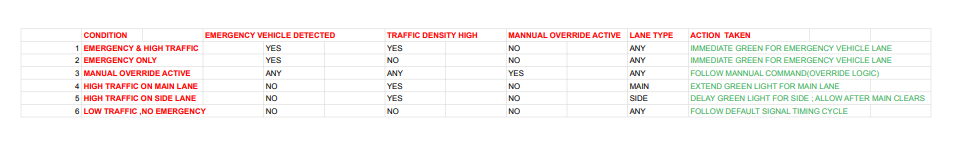
2.1.2. ACTIVITY DIAGRAM



2.1.3. ENTITY-RELATIONSHIP DIAGRAM



2.1.4. CLASS DIAGRAM

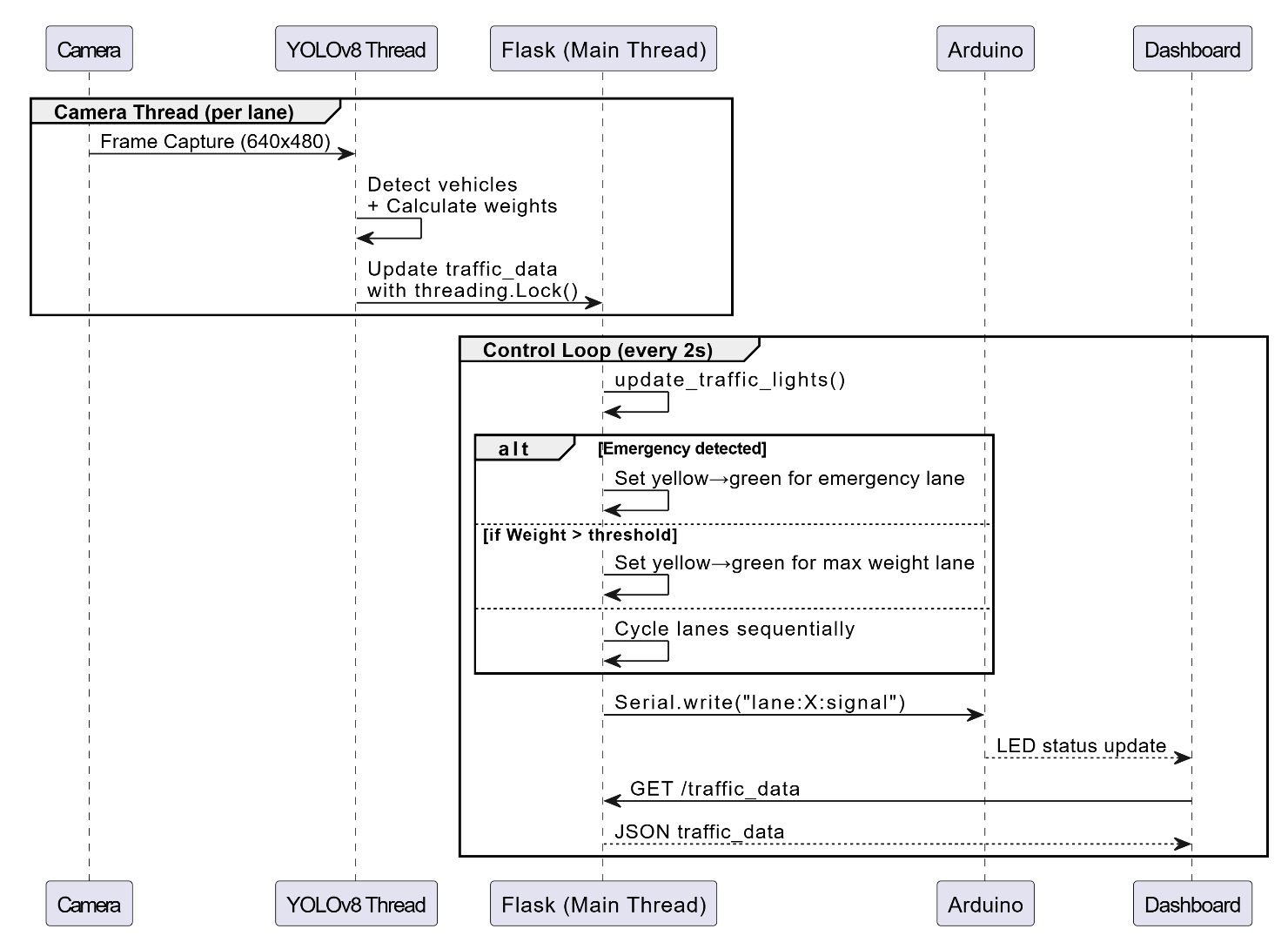


2.1.5. DECISSION TABLE

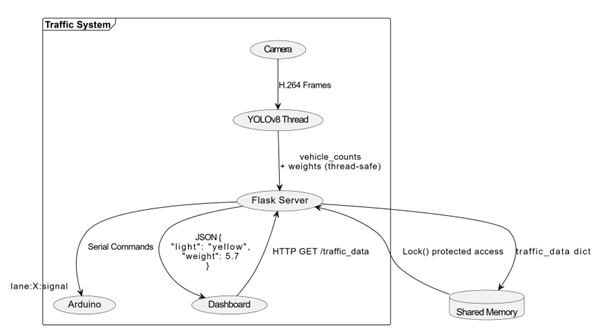
## Component Interactions and Collaborations

Each major component communicates in a linear and event-driven pattern. The flow begins with:

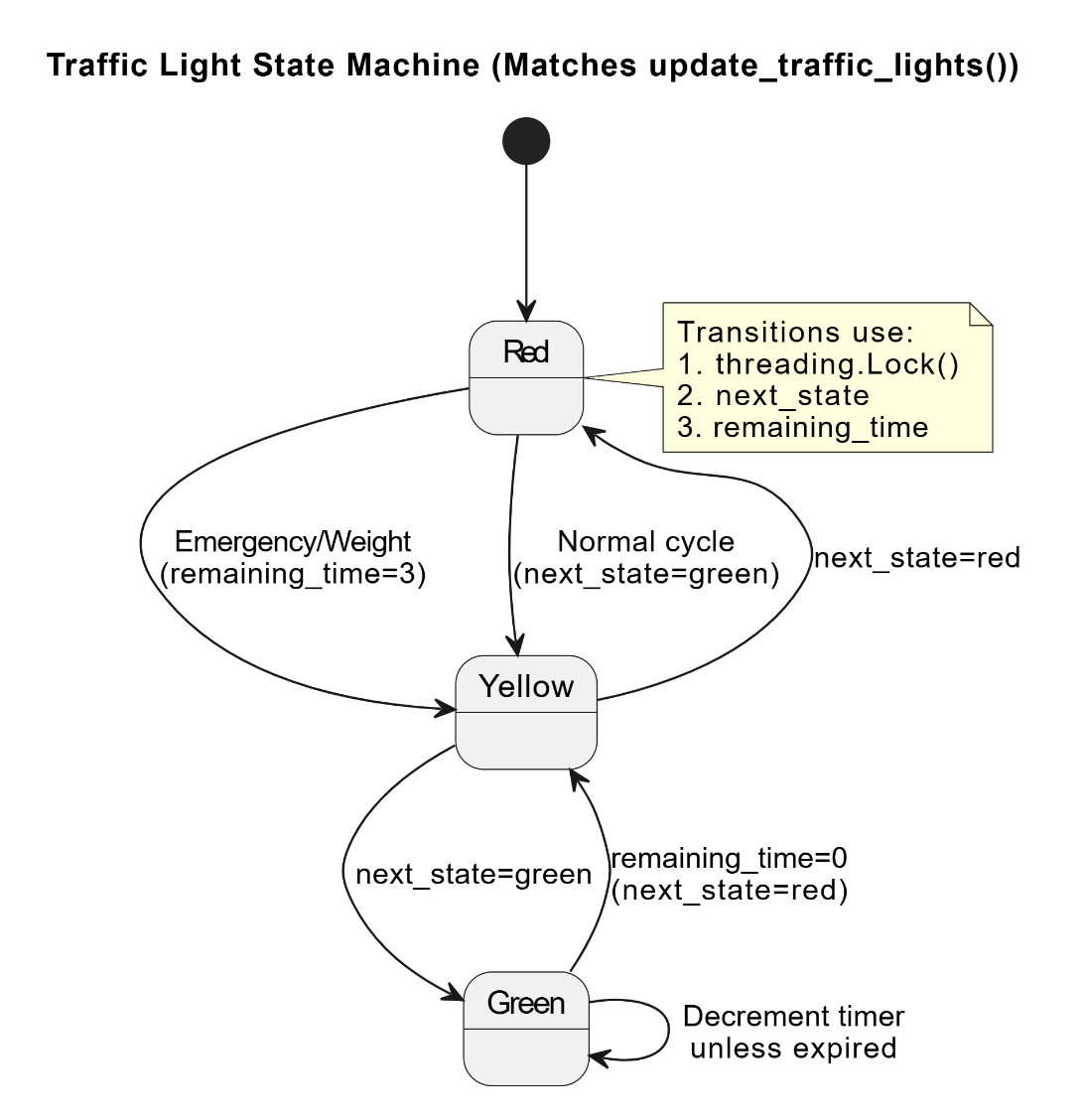
* The camera capturing frames,
* The YOLOv8 model detecting objects and determining signal strategy,
* The Flask server issuing control commands via HTTP/GET,
* The ESP32 microcontroller executing pin toggles and controlling physical LEDs,
* The dashboard updating every 2 seconds via JavaScript using fetch() or WebSocket.
* The entire interaction loop takes less than 2 seconds from detection to signal actuation to matching the required responsiveness for real-time embedded traffic systems.



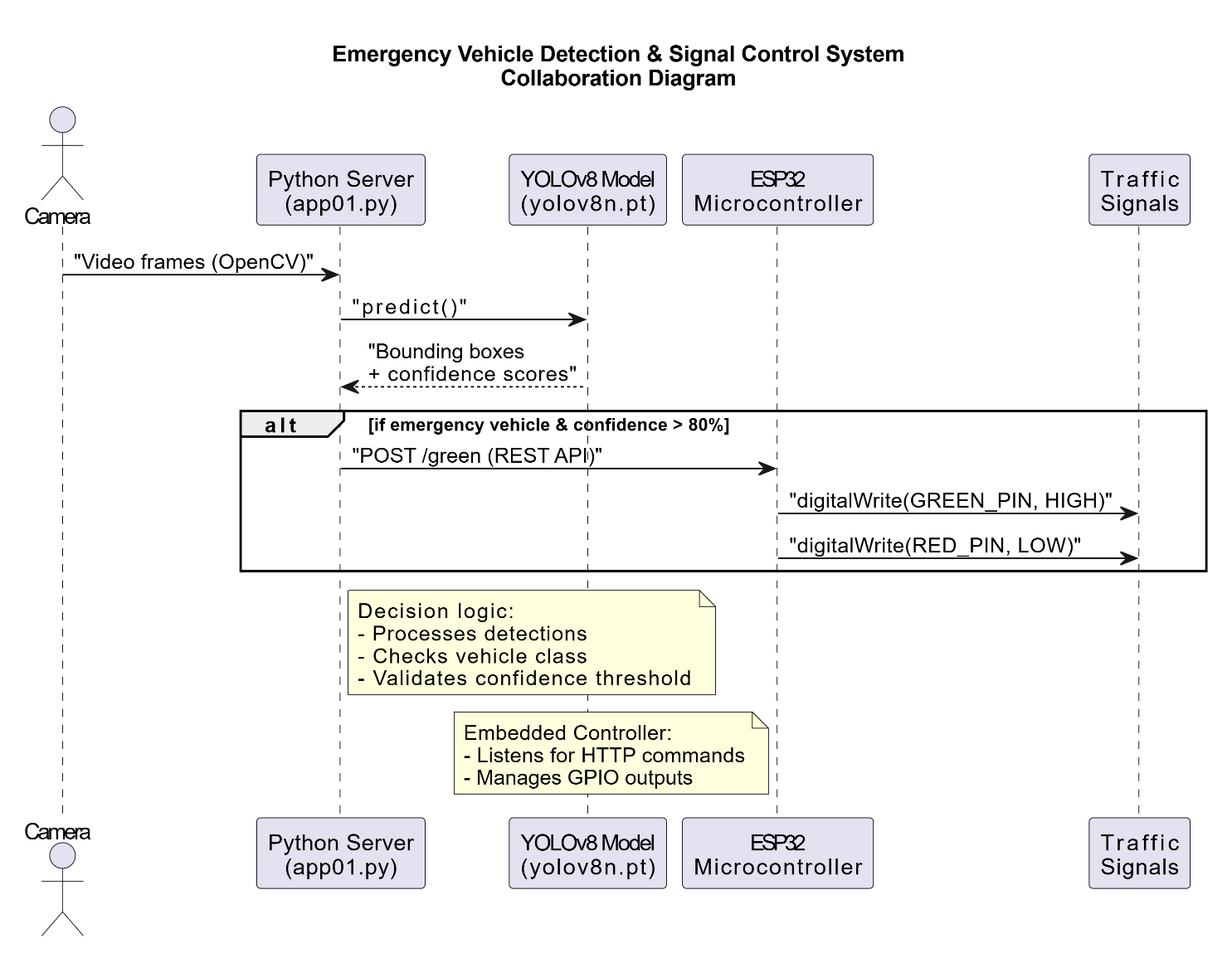
2.2.1. DESIGN LEVEL SEQUENCE DIAGRAM

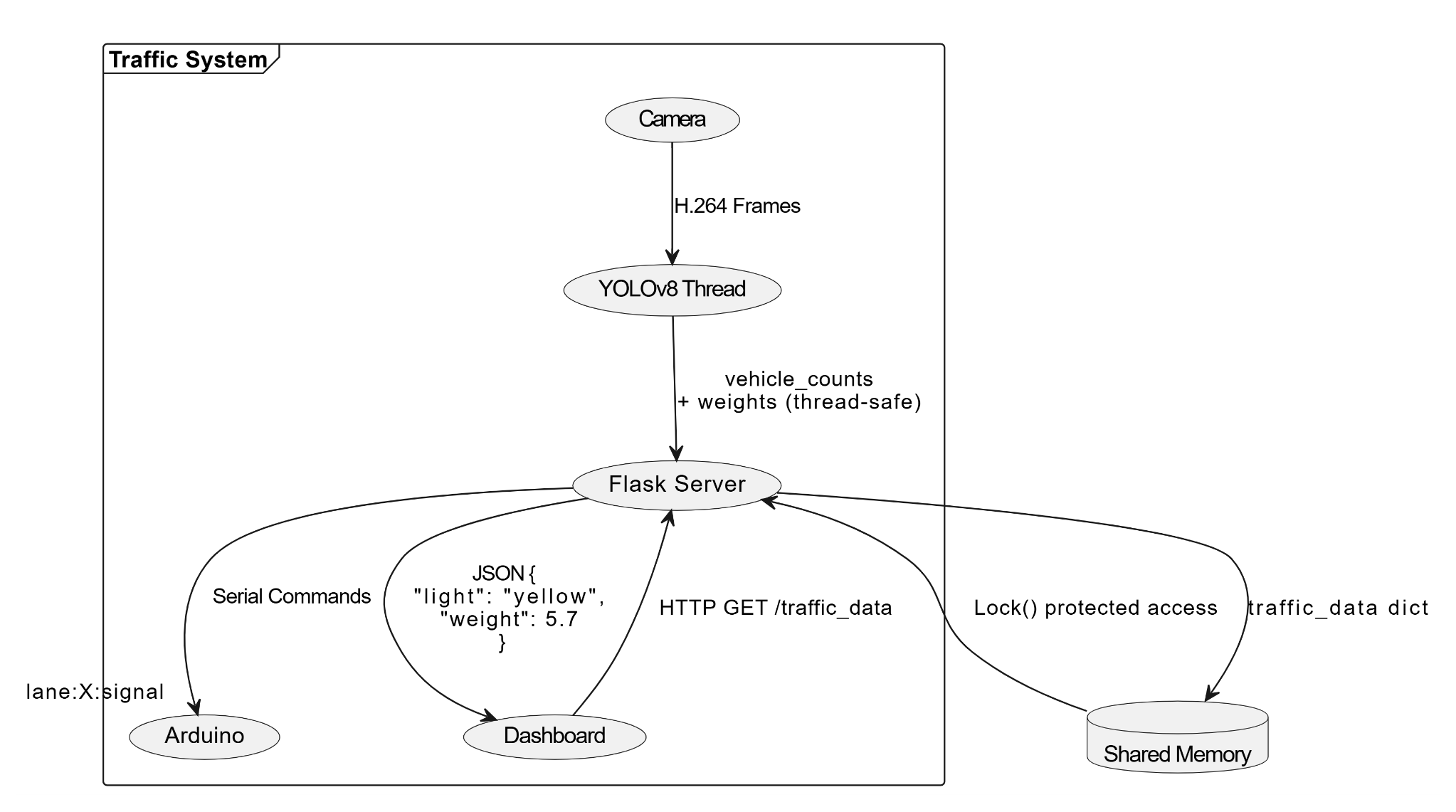


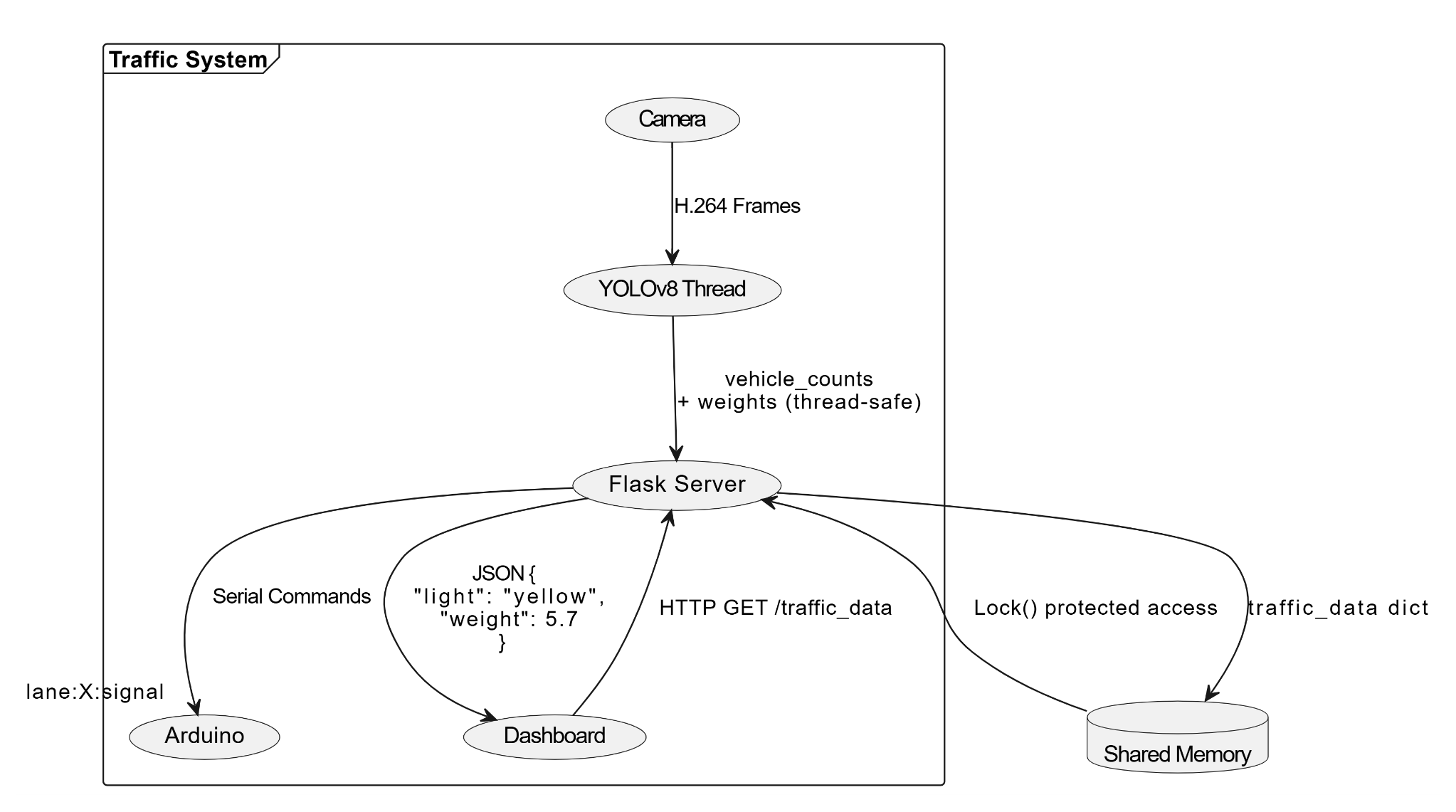
2.2.2. DETAILED DFD



2.2.3. State Transition Diagram

2.2.4. Collaboration Diagram

**

**

## Design Reuse and Design Patterns

The project uses proven architecture and design patterns such as:

* RESTful Communication Pattern: For client-server message exchange between Python back-end and microcontroller.
* Observer Pattern: Utilized within the web dashboard for updating real-time signal changes.
* Modular Reuse: The Arduino handlers (handleGreen(), handleRed(), etc.) are separated and reusable on any GPIO-driven system.

In addition, transfer learning is applied for the YOLOv8 model, making it possible for fast fine-tuning on localized traffic datasets with pretrained weights (yolov8n.pt).

## Technology Architecture

The following platforms, boards, and libraries were used:

**2.4.1. Host System:**

* Python 3.13.3 (64-bit)
* OpenCV-Python
* Ultralytics
* Flask (for backend API)
* PySerial (for testing USB/Serial fallback)

**2.4.2. Embedded Environment:**

* Arduino IDE 2.3.6 (Windows 64-bit)
* Board: ESP32 Dev Module
* Core: esp32 by Espressif Systems v3.2.0
* Libraries: WiFi.h, TM1637Display.h, WebServer.h

**2.4.3. Display Module:**

* Traffic light states (Red, Yellow, Green) represented using physical LEDs connected to ESP32's GPIO pins.

**2.4.4. Additional Drivers:**

* The ESP32 uses the CH340SER.EXE USB-to-serial driver for initial setup. In case of disconnection from the AI backend or dashboard, it defaults to a preset signal cycle to maintain safe and continuous traffic operation.
* CH340SER.EXE used for serial driver communication on Windows for boards that use the CH340 USB-to-Serial chip

## Architecture Evaluation

**2.5.1. Model Selection: YOLOv8**

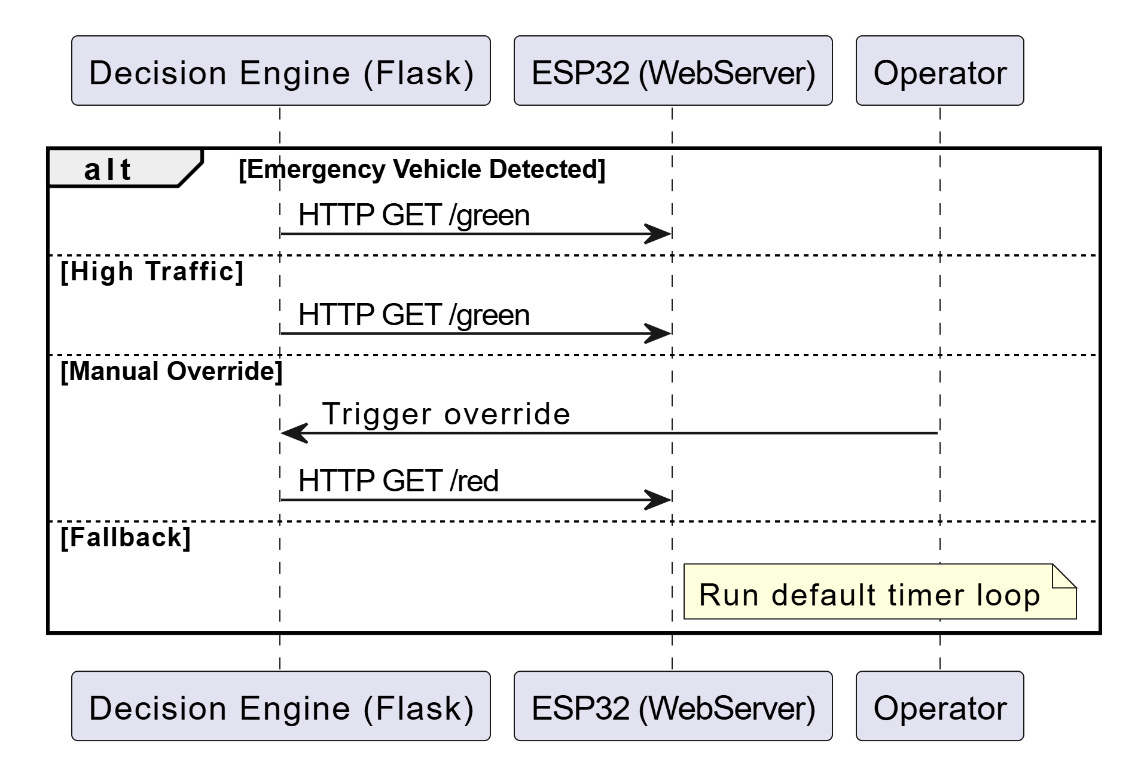
The YOLOv8 architecture was selected due to its optimal balance between detection accuracy and inference speed. Specifically, the **yolov8n.pt** variant was chosen for its lightweight design, making it well-suited for real-time edge deployments. While larger models such as **yolov8x** demonstrate superior accuracy, their increased computational requirements introduce latency, rendering them impractical for time-sensitive applications.

**2.5.2. Hardware Selection: ESP32 Dev Module**

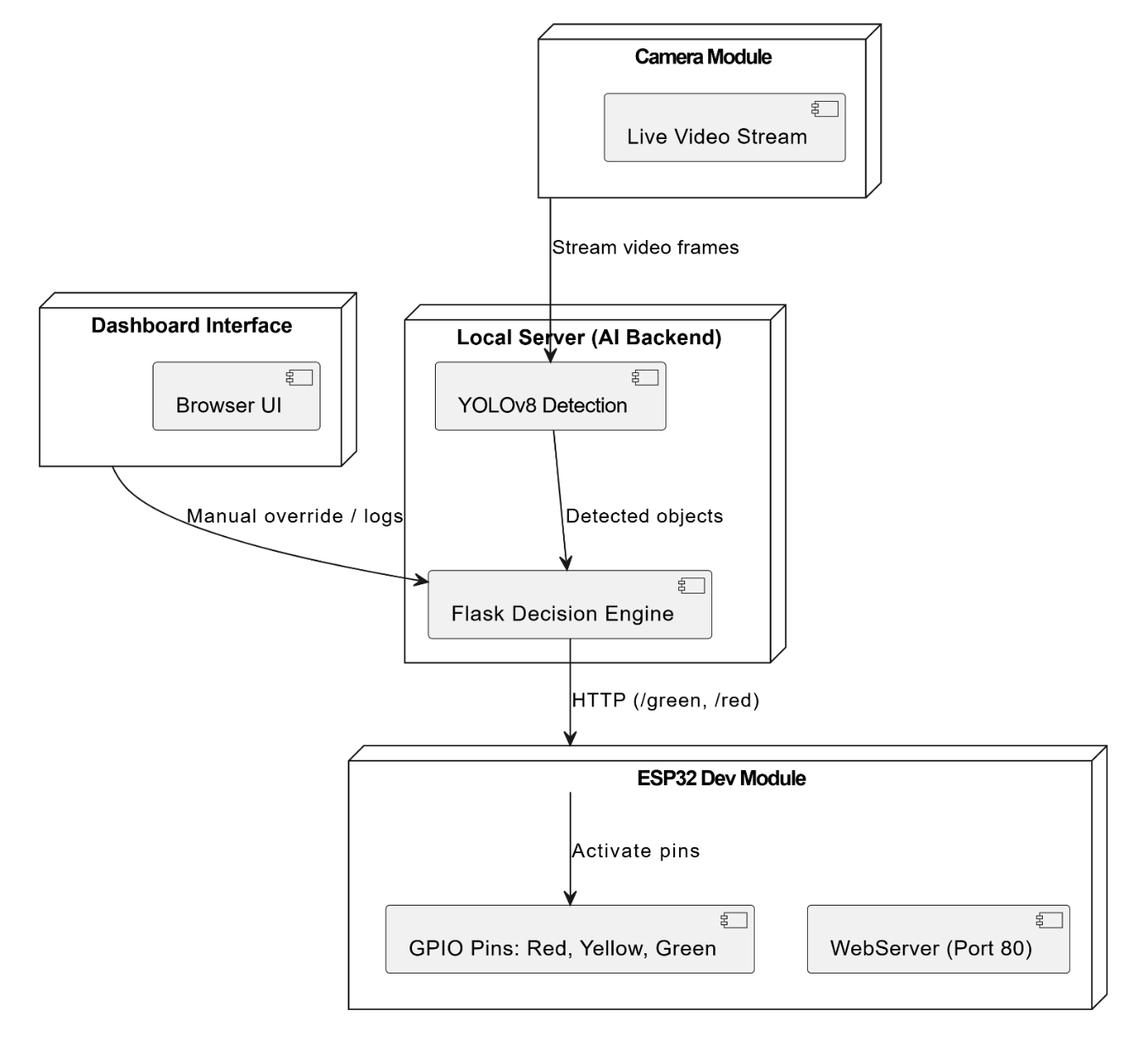
The **ESP32 Dev Module** was selected over conventional microcontrollers (e.g., Arduino Uno) due to its integrated Wi-Fi functionality and enhanced processing capabilities. This enables efficient execution of HTTP-based API communications with the AI backend in real time. Alternatives such as the Uno would necessitate additional modules for network connectivity and lack the necessary computational resources for concurrent operations.

# Detailed/Component Design

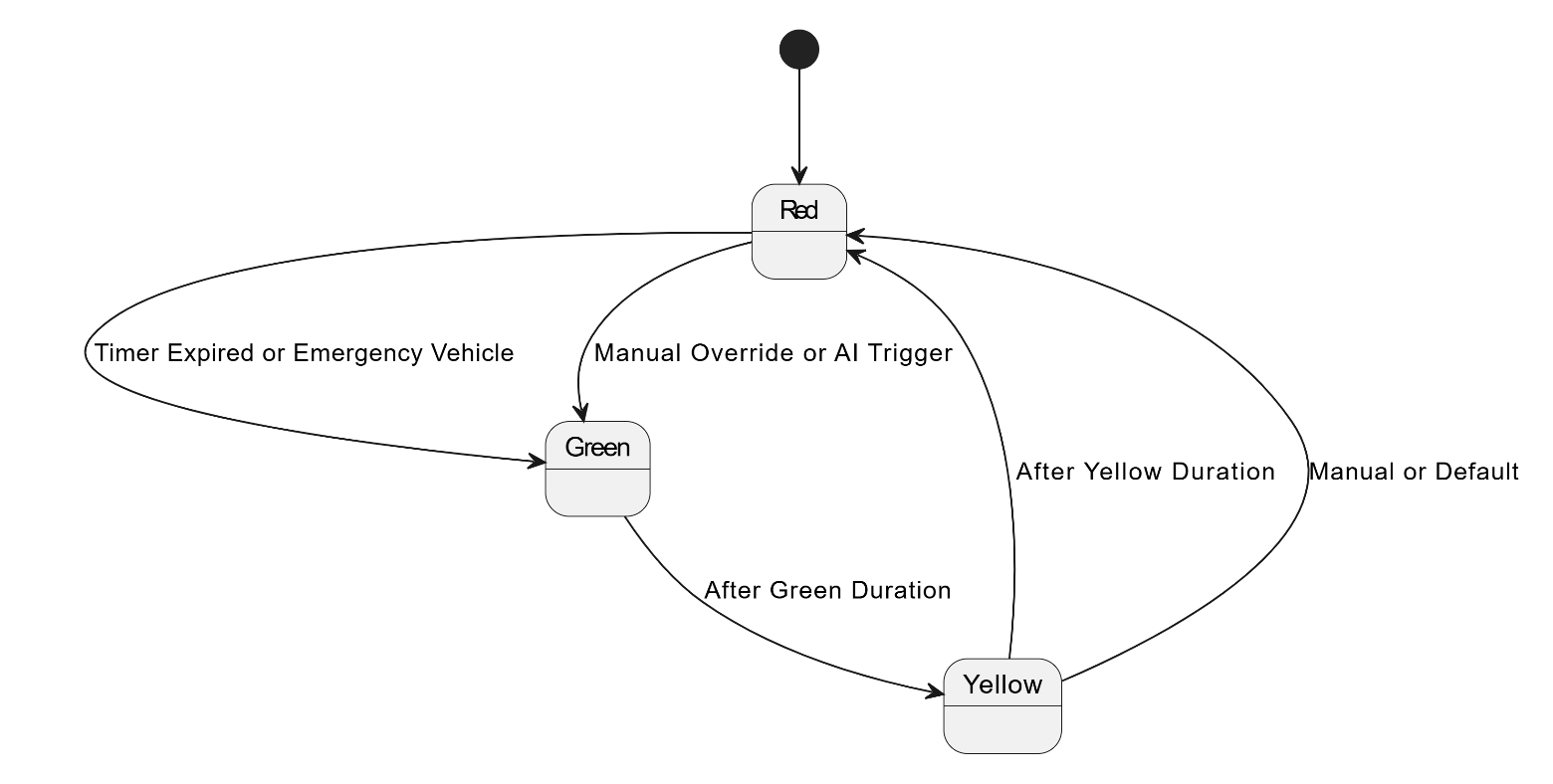
This section presents the internal interaction model of the AI-Based Traffic Optimizer. It expands upon the component-level logic described above by detailing how the system’s software and hardware modules collaborate to process video input, classify vehicles, make decisions, and actuate physical signals. The architecture is organized in a modular fashion, with clear separation between detection logic, decision-making, actuation, and human control.



3.1. Design-Level Sequence Diagram (Real-time Flow)



3.2. Deployment Diagram (Infrastructure Layout)



* 1. State Diagram (Traffic Light Control)

## Component-Component Interface

The internal interaction of the system is driven by modular communication between the AI backend, decision-making logic, and the embedded controller. Camera continuously captures video frames, which are processed in real time by the YOLOv8 model. Detected objects are passed to a Flask-based decision engine that determines the appropriate traffic light action based on lane density or emergency vehicle presence.

Once a decision is made, the Flask server issues request to the ESP32 microcontroller, which operates as a lightweight web server. For example, if a green light needs to be activated:

**3.1.1. Python Backend (Decision Engine):**

*requests.get("http://192.168.1.50/green")*

On the embedded side, the ESP32 handles this request by activating the relevant GPIO pins controlling signal lights.

**3.1.2. ESP32 Web Server Handler:**

*server.on("/green", handleGreen);*

*void handleGreen() {*

*digitalWrite(greenPin, HIGH);*

*digitalWrite(redPin, LOW);*

*digitalWrite(yellowPin, LOW); }*

The web dashboard enables operators to trigger the same endpoints manually using JavaScript-based controls. The combined architecture supports both autonomous and operator-driven traffic control. A design-level sequence diagram in the following subsection illustrates this end-to-end interaction flow.

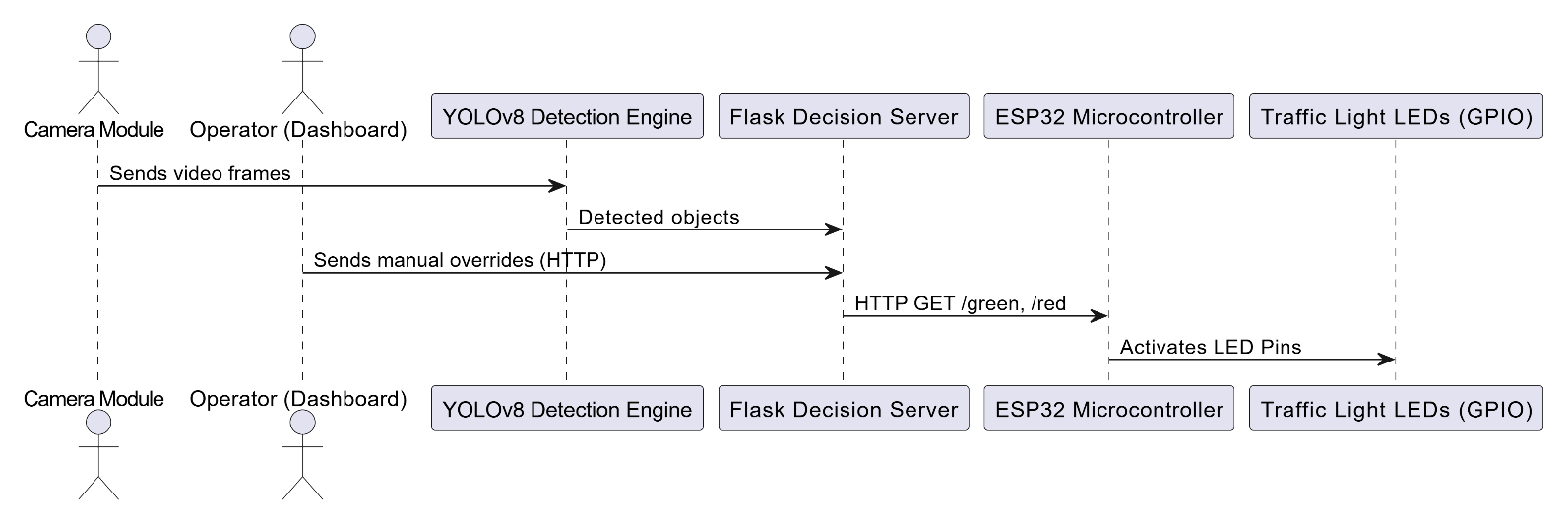
## Component-External Entities Interface

This system interfaces with several physically or logically external components essential to its operation but distinct from its core AI and logic modules. These include the image acquisition unit (camera), the hardware controller (ESP32), and the human-facing dashboard interface.

The camera operates independently of the server, functioning as an external sensor that supplies raw visual data to the detection engine. Once detection and classification are performed internally, control decisions are dispatched to the ESP32, which operates on a separate microcontroller platform and interprets these instructions via stateless HTTP requests.

In addition, the dashboard exists outside the automated detection pipeline, providing authorized personnel a point of control without direct access to the core detection model. This ensures that manual inputs remain logically decoupled from the AI decision layer while still having execution priority.

The system operates entirely within a local environment and does not rely on external APIs or national data services such as NADRA or banking systems. A diagram is included to illustrate these external interfaces and their communication paths.



3.2.1 figure

## Component-Human Interface

The human interface consists of a web-based dashboard, hosted locally. It includes: A real-time traffic status panel, showing current signal phase and vehicle count. Manual override buttons to change light status instantly (helpful for traffic officers or admins). Error log viewer to identify failed communications or camera dropouts. The dashboard HTML (index.html) uses JavaScript fetch() calls to trigger signal change endpoints:

*function triggerGreen()*

*{*

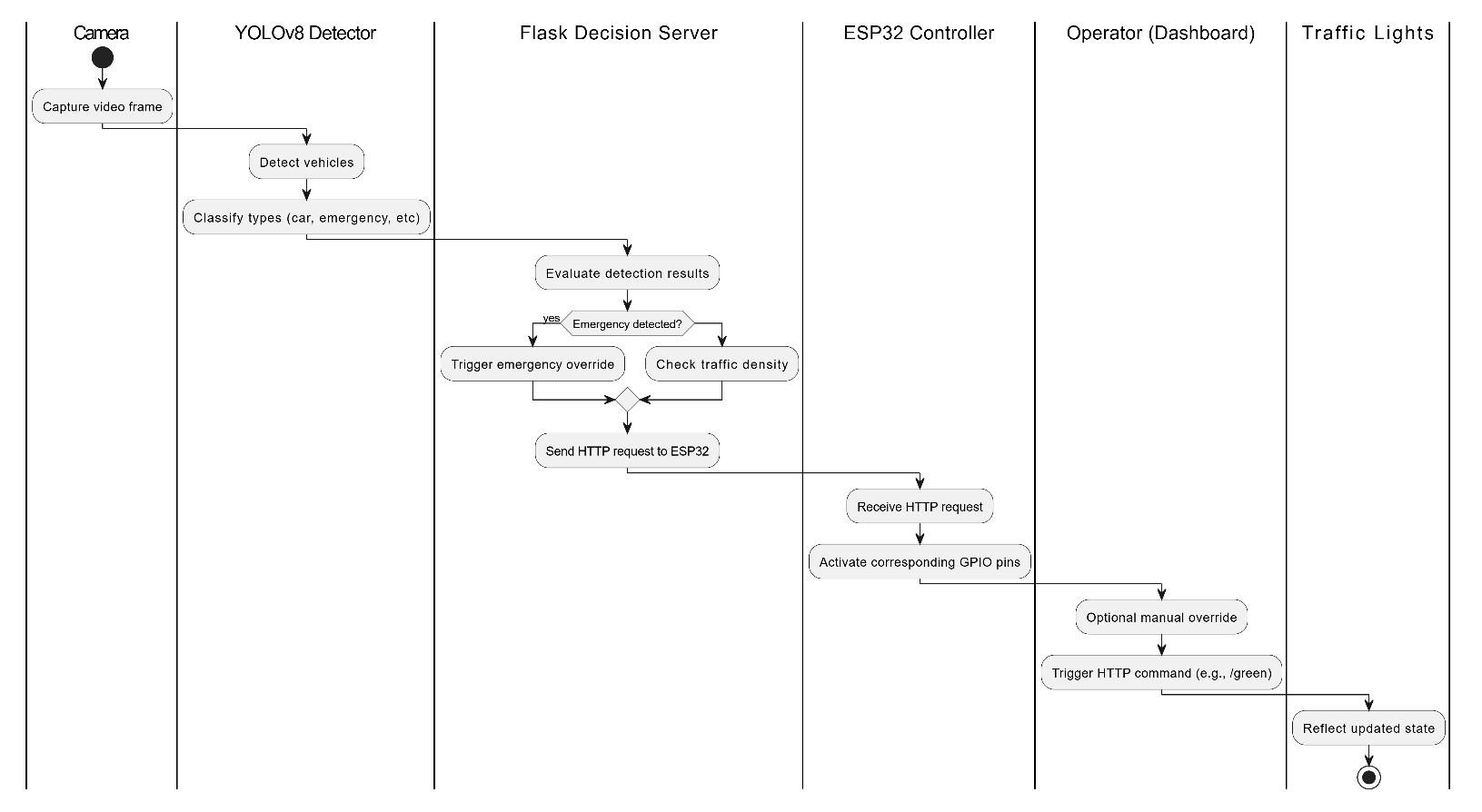
*fetch('http://192.168.1.50/green');*

*}*

This allows any user on the LAN to access and interact with the system securely and instantly. Additionally, HCI principles such as color consistency (Red = Stop, Green = Go), minimal clicks, and alert-based feedback (for emergency override events) were implemented to support usability in high-pressure traffic control scenarios*.*

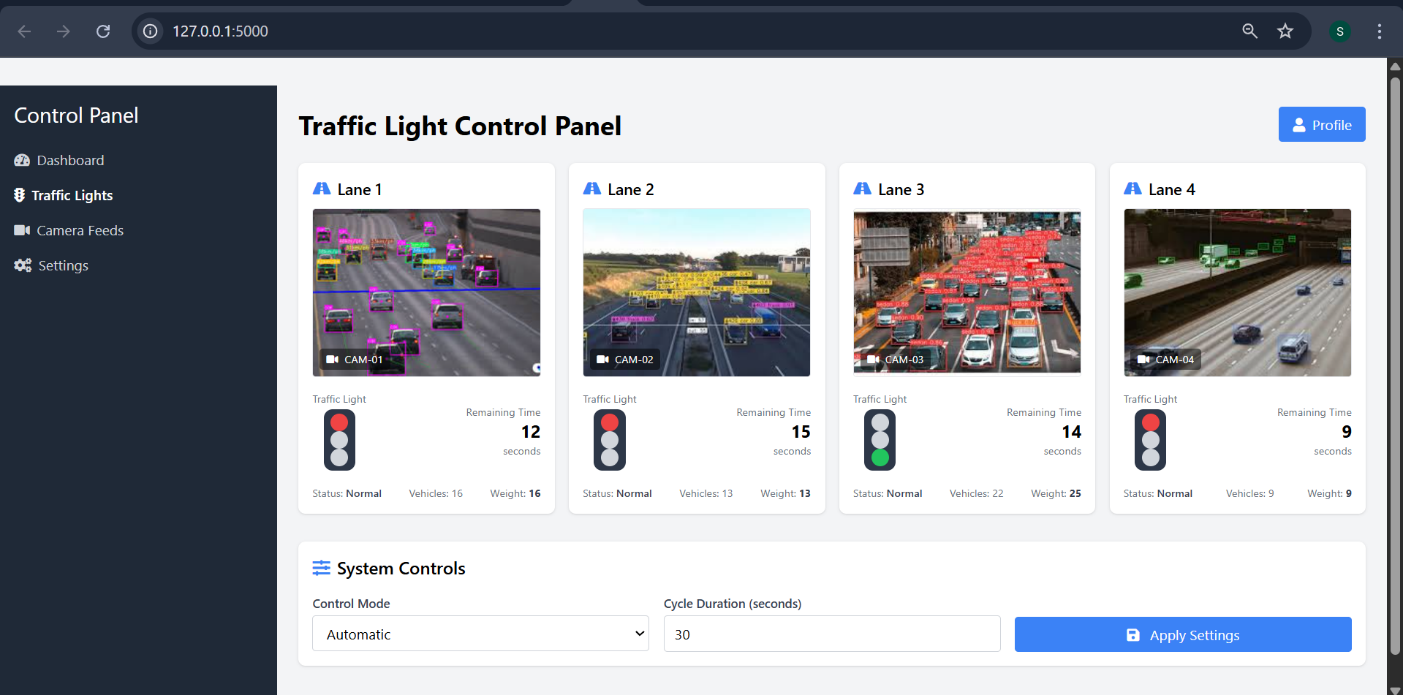
# Screenshots/Prototype

## Workflow

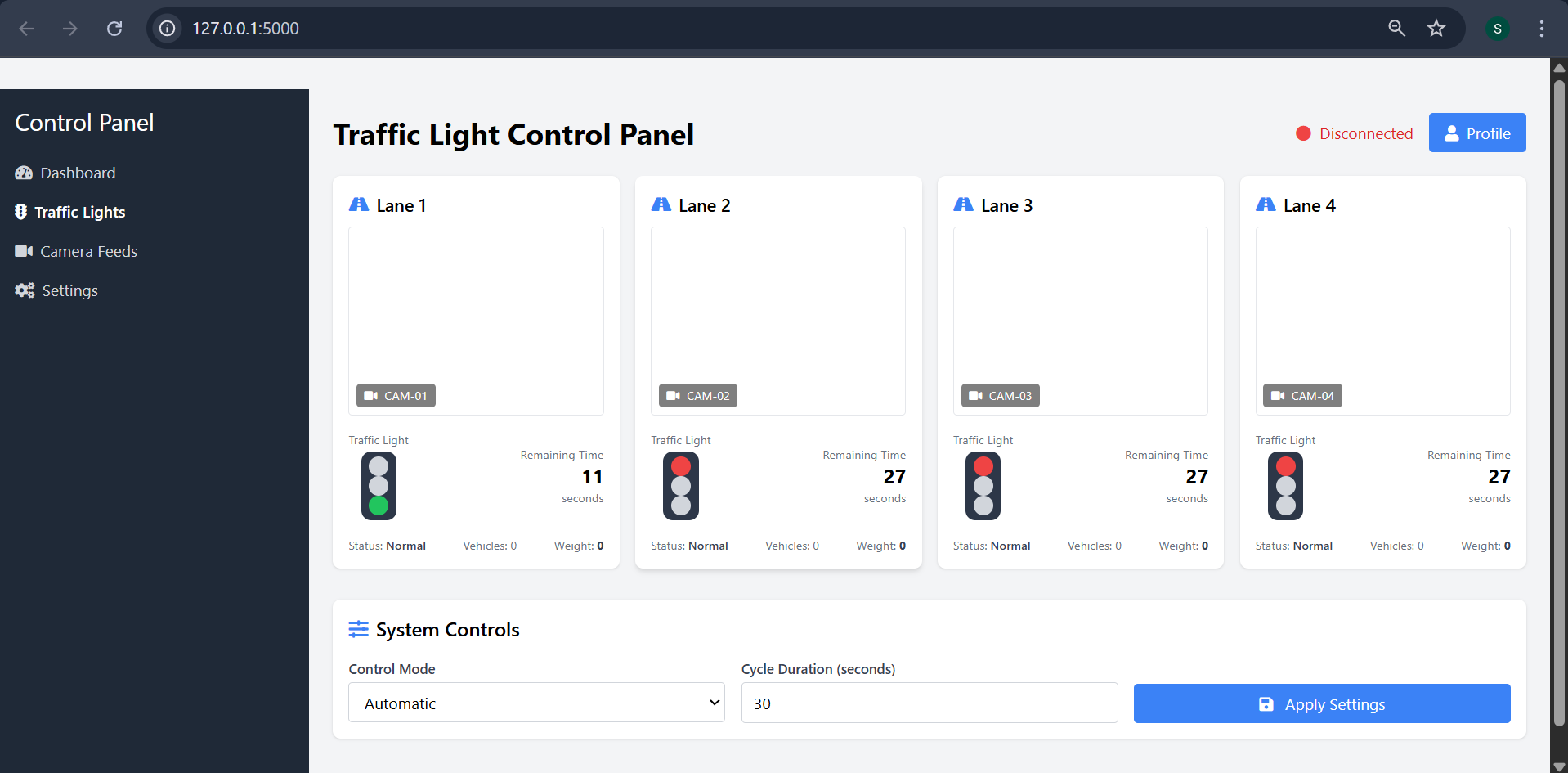


## Screens

Screenshots of software application’s graphical user interface:



4.2.1 Screen (Connected Display)



4.2.2 Screen (Disconnected Display)

## Additional Information

* The dashboard was built using HTML, CSS, and JavaScript, served via Flask’s backend.
* Signal state commands are triggered using asynchronous JavaScript (fetch() API), reducing reload time and improving responsiveness.
* The layout is mobile-responsive, allowing access from tablets or smartphones on the same network.

# Other Design Details

The AI-Based Traffic Optimizer is designed for hardware-embedded deployment. Key hardware-specific elements include:

* Signal Control Circuitry: ESP32 GPIOs are connected to a relay circuit simulating standard traffic lights. Timed transitions are managed within the firmware.
* Driver Dependencies: Use of CH340SER USB-to-serial driver ensures compatibility with systems using CH340 chips.
* Failsafe Mode: In the event of network failure, the ESP32 automatically enters a fallback timed loop, ensuring signal continuity.
* Edge Compatibility: Python and YOLOv8 run on a PC allowing for future on-site deployment without cloud dependency.

# Test Specification and Results

## Test Case Specification

Table 6.1: TC-1

|  |  |
| --- | --- |
| **Identifier** | TC-1 |
| **Related requirements(s)** | Real-time detection and signal automation  (SRS 3.1.1) |
| **Short description** | System should change traffic light to green if AI detects emergency vehicle |
| **Pre-condition(s)** | YOLOv8 running; ESP32 connected to LEDs; Flask server active |
| **Input data** | Frame with emergency vehicle |
| **Detailed steps** | * Start video feed * Inject emergency vehicle image * Observe ESP32 LEDs |
| **Expected result(s)** | Green LED turns ON, red/yellow OFF |
| **Post-condition(s)** | Lane remains green for 15 seconds or until next decision cycle |
| **Actual result(s)** | Green light activated; others off |
| **Test Case Result** | Pass |

Table 6.2: TC-2

|  |  |
| --- | --- |
| **Identifier** | TC-2 |
| **Related requirements(s)** | Dashboard override functionality (SRS 3.1.3) |
| **Short description** | Operator should be able to override signals using dashboard buttons |
| **Pre-condition(s)** | Dashboard loaded; ESP32 accessible over local network |
| **Input data** | Click on "Red" button from dashboard |
| **Detailed steps** | * Open dashboard * Click “Red” override * Watch LEDs and signal icon |
| **Expected result(s)** | Red LED turns ON; red icon displayed; timer resets |
| **Post-condition(s)** | Manual override remains active until reset or timeout |
| **Actual result(s)** | Red signal executed via dashboard |
| **Test Case Result** | Pass |

Table 6.3: TC-3

|  |  |
| --- | --- |
| **Identifier** | TC-3 |
| **Related requirements(s)** | Real-time signal feedback in UI (SRS 3.1.4) |
| **Short description** | |  | | --- | |  |  |  | | --- | | Signal icon and countdown should match GPIO state in real-time | |
| **Pre-condition(s)** | Flask server running; dashboard connected; ESP32 active |
| **Input data** | Send /green or /red command from Flask |
| **Detailed steps** | * Trigger /green via Flask * Observe UI change * Confirm LED state |
| **Expected result(s)** | UI shows green/red icon; timer starts from 15 seconds |
| **Post-condition(s)** | |  | | --- | |  |  |  | | --- | | UI and GPIO remain synced until next event | |
| **Actual result(s)** | Icon and countdown synced with ESP32 |
| **Test Case Result** | Pass |

Table 6.4: TC-4

|  |  |
| --- | --- |
| **Identifier** | TC-4 |
| **Related requirements(s)** | Vehicle and emergency type detection (SRS 3.1.2) |
| **Short description** | |  | | --- | |  |  |  | | --- | | YOLOv8 must detect vehicles and classify emergency vehicles with ≥90% confidence | |
| **Pre-condition(s)** | YOLOv8 loaded with trained weights; test images prepared |
| **Input data** | Image set: cars, bikes, ambulances, fire trucks |
| **Detailed steps** | * Run detection on image * Check labels and confidence levels Observe UI change |
| **Expected result(s)** | Bounding boxes around objects; emergency classes ≥0.9 confidence |
| **Post-condition(s)** | Detection result forwarded to decision engine |
| **Actual result(s)** | All vehicle types classified correctly |
| **Test Case Result** | Pass |

Table 6.5: TC-5

|  |  |
| --- | --- |
| **Identifier** | TC-5 |
| **Related requirements(s)** | Inter-device communication reliability (SRS 3.1.5) |
| **Short description** | |  | | --- | |  |  |  | | --- | | Flask should reliably transmit signal commands to ESP32 | |
| **Pre-condition(s)** | Flask server and ESP32 connected to same network |
| **Input data** | command: requests.get("http://192.168.1.50/green") |
| **Detailed steps** | * Execute command * Monitor ESP32 LED behavior |
| **Expected result(s)** | LED responds within a second; proper state change occurs |
| **Post-condition(s)** | Response acknowledged via HTTP 200 |
| **Actual result(s)** | Reliable response |
| **Test Case Result** | Pass |

Table 6.6: TC-6

|  |  |
| --- | --- |
| **Identifier** | TC-6 |
| **Related requirements(s)** | Fallback to timed cycle on AI disconnection  (SRS 3.2.1) |
| **Short description** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | ESP32 should auto-switch to default timer if AI/backend is unresponsive | | |
| **Pre-condition(s)** | Disconnect server or simulate timeout |
| **Input data** | No HTTP request received for 30+ seconds |
| **Detailed steps** | * Disconnect Flask * Wait for 30s * Observe signal behavior |
| **Expected result(s)** | ESP32 begins 30s red → green → yellow default cycle |
| **Post-condition(s)** | Loop continues until connection resumes |
| **Actual result(s)** | ESP32 fallback loop engaged |
| **Test Case Result** | Pass |

## Summary of Test Results

**Table 6.2: Summary of Test Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Module Name** | Test cases run | Number of defects found | Number of defects corrected so far | Number of defects still need to be corrected |
| **Vehicle Detection (YOLOv8)** | TC3, TC4 | 1 | 1 | 0 |
| **Traffic Signal Controller (ESP32)** | TC2, TC6 | 2 | 1 | 1 |
| **Web-Based Dashboard Interface** | TC1, TC5 | 1 | 1 | 0 |
| **Emergency Vehicle Detection** | TC4 | 1 | 1 | 0 |
| **Manual Override & Signal Sync** | TC5, TC6 | 1 | 0 | 1 |
| **Camera Feed Processing** | TC1, TC3 | 1 | 1 | 0 |
| **Complete System** | TC1 – TC6 | 7 | 5 | 2 |

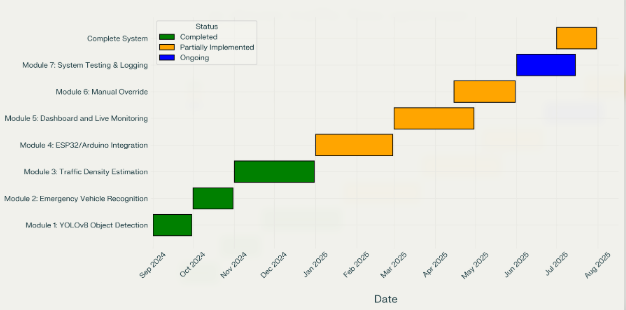
# Revised Project Plan

Table 7.1: Project Completion Status

|  |  |
| --- | --- |
| **Module Name** | **Status** |
| **Object Detection (YOLOv8)** | Completed |
| **Emergency vehicle detection** | Completed |
| **Traffic Density Estimation** | Completed |
| **Dynamic Signal Controller (ESP32 + Arduino Logic)** | Partially Implemented |
| **Web-based Dashboard & Live Feed** | Partially Implemented |
| **Manual Override Functionality** | Partially Implemented |
| **System Diagnostics & Logging** | Not Implemented |
| **Complete System** | Partially Implemented |

|  |  |
| --- | --- |
|  |  |

**7.2. Gantt Chart**



# References

* Ultralytics YOLOv8 Documentation. <https://docs.ultralytics.com>
* Arduino IDE 2.3.6 Documentation. <https://docs.arduino.cc>
* Flask Microframework. <https://flask.palletsprojects.com/>
* Choudhury, K., & Nandi, D. (2021). Detection and prioritization of emergency vehicles in intelligent traffic management system. IEEE Bombay Section Signature Conference, IBSSC 2021.
* Salekin, S. U., Ullah, M. H., Khan, A. A. A., Jalal, M. S., Nguyen, H. H., & Farid, D. M. (2024). Bangladeshi native vehicle classification employing YOLOv8. Communications in Computer and Information Science.
* IEEE 829-1998 Software Test Documentation Standard.

Appendix A: Glossary

|  |  |
| --- | --- |
| YOLOv8 | Real-time object detection model used for traffic analysis |
| ESP32 | Wi-Fi enabled microcontroller used for traffic light control |
| GPIO | General Purpose Input/Output pins for signal interfacing |
| Flask | Python micro web framework for building the API backend |
| Manual override | The capability for operators to manually control traffic signals in case of system errors. |
| CH340SER | USB-to-serial driver used to program ESP32 boards |

Appendix B: IV & V Report

**(Independent verification & validation)**

**IV & V Resource**

Name Signature

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S#** | **Defect Description** | **Origin Stage** | **Status** | **Fix Time** | |
| **Hours** | **Minutes** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| … |  |  |  |  |  |

**Table 1: List of non-trivial defects**

This document has been adapted from the following:

* Previous project templates at UCP
* High-level Technical Design, Centers for Medicare & Medicaid Services. (www.cms.gov)

# Semester wise SDP Meeting Report

**Project Title:** AI-DRIVEN TRAFFIC FLOW OPTIMIZER

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Semester:** | | | | | | **F24** | |
| **Group ID** | | **Student Roll Number** | | | **Student Name and Signatures** | **Advisor** | |
| **F24BS175** | | **L1F21BSCS0987** | | | **SULMAN AHMAD** | **Misha Asif** | |
| **L1F21BSCS1208** | | | **ANEEQ-UR-REHMAN** |
| **L1F1BSCS0864** | | | **INAYAT NAVEED** |
| Sr. | Date | | Status (P/A/L) | | | Agenda Items | Notes | |
|  | | | SULMAN AHMAD  (L1F21BSCS0987) | ANEEQ-UR-REHMAN  (L1F21BSCS1208) | INAYAT NAVEED  (L1F21BSCS0864) |  | | |
| 1 | 24-03-2025 | | Present | Present | Present | Discussion on Application |  | |
| 2 | 31-03-2025 | | Present | Present | Present | Working on architecture |  | |
| 3 | 07-04-2025 | | Present | Present | Present | Finalizing the architecture |  | |
| 4 | 14-04-2025 | | Present | Present | Present | Discussion on Diagrams |  | |
| 5 | 21-04-2025 | | Present | Present | Present | Documentation review and remaining document’s part discussion |  | |
| 6 | 28-04-2025 | | Present | Present | Present | Finalized Prototype |  | |
| 7 | 5-05-2025 | | Present | Present | Present | Review of work done |  | |
| 8 | 12-05-2025 | | Present | Present | Present | Further discussion and overview of Phase 3 submission |  | |

Date: **12th** **May**,**2025** Advisor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_