Got it ✅ now we’re on the **correct SIH problem statement**:

**Problem Statement ID – 25050**  
**Title – Smart Traffic Management System for Urban Congestion**  
**Theme – Transportation & Logistics**  
Your solution = **SanchalanAI** → An AI-based traffic management system with:

* Reinforcement learning for adaptive traffic signals
* Integration of IoT sensors & cameras for real-time data
* Emergency green corridor feature
* Cloud-based scalable deployment

Since you’re the **AI/ML lead** and your team has:

* **You (AI/ML)**
* **2 members on Data collection (YOLO/OpenCV + IoT devices)**
* **3 members on AWS + IoT connectivity**

👉 Here’s the **Step-by-Step “Shame” (System Architecture & Execution Guide)** tailored to this problem:

**🔹 Step 1 — Team Task Assignment**

**You (AI/ML)**

* Design **traffic signal RL model** (simulation-based, e.g., SUMO + Stable-Baselines3).
* Train/test in a simulated city network with different traffic densities.
* Provide API for “signal timing decision” → green/red duration per signal.

**Members 1 & 2 (CV + IoT data)**

* Use **YOLOv8 + OpenCV** to detect vehicle count, type (car, bus, bike, ambulance).
* Camera setup: **Raspberry Pi 4 + Pi Camera HQ + Arducam 6mm lens** (clear vehicle capture at intersections).
* Send counts (per lane) + emergency vehicle flag → IoT pipeline.

**Members 3, 4 & 5 (AWS/IoT)**

* **AWS IoT Core**: receive live data from cameras.
* Store historical traffic data in **AWS RDS (Postgres)**.
* Deploy AI model API on **AWS EC2/ECS**.
* **S3** for video/image storage (optional).
* **CloudWatch** for monitoring.

**🔹 Step 2 — Data Flow (End-to-End)**

1. Camera → YOLO (vehicle count, emergency detection).
2. IoT device → AWS IoT Core (publish vehicle counts).
3. AWS Lambda → push structured data to RDS.
4. RL Model API (FastAPI) → pulls traffic state → computes optimal signal time.
5. IoT device (Raspberry Pi) → updates traffic light controller.
6. Special case: If emergency vehicle detected → **Green Corridor** override.

**🔹 Step 3 — Your AI/ML Implementation**

1. **Simulation Setup**
   * Use **SUMO (Simulation of Urban Mobility)** with traci Python API.
   * Define a **4-way intersection** with variable traffic inflow.
   * Input = vehicle counts from each lane.
   * Output = next phase duration (green/yellow/red).
2. **Model**
   * Use **Reinforcement Learning (RL)** → DQN or PPO from stable-baselines3.
   * State = [vehicle counts per lane, waiting times].
   * Action = [adjust green time for NS/EW direction].
   * Reward = - (average waiting time + queue length).
3. **Training**
4. import gym
5. from stable\_baselines3 import PPO
6. # custom SUMO gym env
7. env = SumoTrafficEnv()
8. model = PPO("MlpPolicy", env, verbose=1)
9. model.learn(total\_timesteps=50000)
10. model.save("traffic\_rl\_model")
11. **Emergency Vehicle Override**
    * If YOLO detects ambulance → override RL and force **Green for corridor**.
12. **FastAPI Deployment**
    * /get\_signal\_timing → input = JSON {vehicle\_counts}, output = {NS: 30s, EW: 40s}.

**🔹 Step 4 — IoT Device Setup**

* Device: **Raspberry Pi 4 + Pi Camera HQ + Arducam lens**.
* Accelerator (optional): **Google Coral USB TPU** for faster YOLO inference.
* Microcontroller: **ESP32** for lightweight control of traffic signal lights.
* MQTT protocol to send JSON:
* {
* "intersection\_id": "INT\_001",
* "timestamp": "2025-09-15T12:30:00",
* "vehicle\_counts": {"NS": 20, "EW": 35},
* "emergency\_detected": true
* }

**🔹 Step 5 — AWS Cloud Setup**

1. **AWS IoT Core** → Device → Rule → Lambda.
2. **AWS Lambda** → store traffic data to RDS (Postgres).
3. **RDS Postgres schema**:
4. intersections(id, location, total\_lanes)
5. traffic\_data(id, intersection\_id, ts, ns\_count, ew\_count, emergency\_flag)
6. **EC2/ECS** → host FastAPI RL model service.
7. **Frontend Dashboard (Optional)** → Streamlit / React (traffic visualization).

**🔹 Step 6 — Demo Workflow (For Judges)**

1. Show live camera feed detecting cars + emergency (YOLO).
2. Data pushed to AWS (IoT → DB).
3. RL model API outputs optimal signal timing.
4. Visualize simulation in SUMO (queue lengths reduced).
5. Show Green Corridor activation when ambulance detected.

**🔹 Step 7 — Deliverables**

* **Prototype Demo** (YOLO detection + SUMO RL + AWS pipeline).
* **Presentation Slides** (Problem → Architecture → Demo → Benefits).
* **Demo Video** (backup for judges).
* **Code repo** (structured with ML, IoT, AWS, docs).

⚡ With this **shame (system architecture)**, you can now:

* Start **your RL model training in SUMO + FastAPI wrapper** immediately.
* CV team starts **YOLO + Pi Camera setup**.
* AWS team starts **IoT pipeline setup**.

👉 Do you want me to now make a **clear architecture diagram** (AI + IoT + AWS + CV flow) that you can directly use in your presentation?

SanchalanAI/

│── README.md # Project overview, setup instructions

│── requirements.txt # Python dependencies (root-level)

│── docker-compose.yml # For local testing (optional)

│── .gitignore

│

├── backend/ # FastAPI service + APIs

│ ├── main.py # FastAPI entrypoint

│ ├── routers/ # API routes

│ │ ├── traffic.py # /get\_signal\_timing, /emergency

│ │ └── health.py

│ ├── models/ # Pydantic models (request/response schemas)

│ ├── services/ # Business logic

│ │ ├── rl\_engine.py # Call trained RL model

│ │ └── db\_service.py # Fetch/store traffic data

│ ├── database/ # DB connection utils (Postgres/AWS RDS)

│ ├── tests/ # Unit tests

│ └── Dockerfile

│

├── ml/ # AI/ML models (your domain)

│ ├── rl\_model/ # Reinforcement learning training

│ │ ├── train.py # Train PPO/DQN with SUMO

│ │ ├── env.py # Custom SUMO gym environment

│ │ └── config.yaml

│ ├── emergency\_override.py# Ambulance/green corridor override logic

│ └── notebooks/ # Jupyter notebooks for experiments

│

├── cv\_iot/ # Computer Vision + IoT device code

│ ├── yolo/ # YOLO vehicle detection

│ │ ├── train.py

│ │ ├── detect.py

│ │ └── data.yaml

│ ├── ocr/ # OCR (if extracting plate text/extra info)

│ │ └── tesseract\_demo.py

│ ├── raspberry\_pi/ # Pi + camera code

│ │ ├── cam\_capture.py # Capture & preprocess frames

│ │ ├── send\_mqtt.py # Publish counts to AWS IoT Core

│ │ └── requirements.txt

│ └── esp32/ # ESP32 code for traffic light control

│ └── traffic\_light.ino

│

├── infra/ # AWS & deployment configs

│ ├── terraform/ # If using IaC (Terraform/CDK)

│ ├── iot\_rules/ # AWS IoT Core rules JSON

│ ├── lambda/ # AWS Lambda functions

│ │ └── store\_data.py

│ └── scripts/ # Helper scripts (DB migrations, setup)

│

├── frontend/ # Dashboard (Streamlit/React)

│ ├── app.py # Streamlit demo

│ └── components/ # UI components

│

├── simulation/ # SUMO simulation configs

│ ├── network.net.xml # SUMO road network

│ ├── routes.rou.xml # Vehicle routes

│ ├── config.sumocfg # Main simulation config

│ └── run\_sumo.py # Run SUMO simulation with RL

│

└── docs/ # Reports, presentation, SIH deliverables

├── architecture.png # System architecture diagram

├── sih\_submission.pdf

└── sprint\_plan.md