Intelligent Traffic Analysis & Violation Detection using Video Analytics

NETRIK -IIITDM Hackathon Project Report

Intelligent Traffic Analysis & Violation Detection using Video Analytics

Team/Participant: TEAM ORBIT

PROBLEM STATEMENT: 2

Hackathon Track: Computer Vision • Video Analytics • Smart Cities

Date (IST): 2026-01-31

Submission Format: PDF, Github Repo :-

```
https://github.com/harsha-cpp/traq.git
```

Project Codename: TRAQ

---

1. Abstract

Urban intersections in India frequently suffer from heavy congestion, weak lane discipline, and unsafe driving behavior. Traditional traffic operations often rely on fixed-time signal plans and manual monitoring, which do not scale well and provide limited real-time insight.

This project proposes an **AI-powered video analytics system** that processes **pre-recorded CCTV footage** from signalized intersections to generate actionable traffic intelligence. The system detects and tracks multiple vehicle categories, estimates **queue length** and **queue density/occupancy** near stop lines, and identifies two key violation types: **red-light jumping** and **rash driving (risk flags)** based on explainable motion/trajectory heuristics. Results are

presented through a lightweight **NextJS dashboard** and **annotated video output**, focusing on accuracy, tracking consistency, and interpretability rather than production deployment.

---

2. Problem Understanding and Motivation

2.1 Real-world challenge

Indian intersections are complex due to:

- Mixed traffic (two-wheelers, autos, cars, buses, trucks)
- Frequent occlusions and dense clustering
- Unstructured queuing (vehicles do not align neatly in lanes)
- High violation rates (red-light jumping, dangerous weaving, sudden acceleration)

These factors make manual monitoring difficult and fixed-time signaling insufficient for understanding *what is happening in the scene*.

2.2 Opportunity with CCTV + vision analytics

With widespread CCTV availability, computer vision can automate:

- **Traffic state estimation:** queue formation, density, congestion peaks
- **Event detection:** stop-line crossings during red phase
- **Behavioral flags:** trajectories suggesting risky driving

This project focuses on **analytics and perception**, not on signal control or hardware deployment.

---

3. Objectives and Scope

3.1 Core objectives (Mandatory)

1. Process pre-recorded intersection videos (Indian roads preferred)
2. Detect multiple vehicle types (cars, bikes/scooters, autos, buses, trucks, etc.)
3. Track vehicles over time with consistent IDs (multi-object tracking)
4. Compute:

- **Queue length** (vehicle count basis)
- **Queue density / occupancy** (vehicles per unit area or area coverage)

5. Detect violations:
   - **Red-light jump detection**
   - **Rash driving detection** via explainable heuristics
6. Present outputs via:
   - **NextJS dashboard**
   - **Annotated video output**

3.2 Out of scope (As per problem statement)

- No real-time camera feed requirement
- No hardware integration / IoT deployment
- No signal control logic
- No production-scale deployment requirements

---

4. Assumptions and Constraints

4.1 Assumptions

- Camera is mostly static (typical CCTV mounting).
- Stop line can be approximated manually if not clearly marked.
- Signal phase can be provided manually if traffic light visibility is poor.
- Pixel-space measurements are acceptable for trends when real-world calibration is unavailable.

4.2 Constraints

- Multi-object tracking is required (frame-wise detection alone is insufficient).
- System must be modular and explainable.
- Must include a web interface

4.3Tech Stack

Dashboard

- **Next.js** (App Router) + **Tailwind CSS**
- Charts: **Recharts** (or Chart.js)
- Media: HTML5 video player + evidence gallery

Backend

- API: **FastAPI (Python)** *(or Go: Gin/Fiber)*
- Jobs: simple queue/worker pattern

Vision Engine

- **Python**: YOLO (vehicle detection) + MOT (tracking)
- Outputs: `annotated.mp4`, `metrics_queue.csv`, `violations.json`, evidence frames/clips

Storage

- AWS/LOCAL

---

5. Proposed System Overview

5.1 High-level pipeline

1. **Video Ingestion**
2. **Preprocessing** (resize / frame sampling)
3. **Vehicle Detection** (per frame)
4. **Multi-Object Tracking (MOT)** (IDs + trajectories)
5. **Queue Analytics** (queue length + density/occupancy)
6. **Violation Analytics**
    - Red-light jump detection
    - Rash driving risk flags
7. **Visualization**
    - Annotated video
    - Evidence frames/clips for events

8. **Dashboard** (NextJS)
    - Charts, summaries, event review

5.2 Why tracking is central

Queue and violation reasoning requires temporal continuity:
- a vehicle must be tracked reliably to measure dwell-time in queue zones

- a red-light jump depends on *when* a vehicle crosses the stop line

- rash driving requires speed/acceleration/trajectory patterns over time

Therefore, **tracking consistency is treated as a first-class requirement**, not an optional enhancement.

---

6. Methodology (Conceptual Design)

This section describes the *ideas and logic* used in the system—without implementation details or code.

6.1 Vehicle detection

Goal: Detect vehicles of multiple categories under occlusion and mixed traffic.

Expected categories (minimum):
- Car
- Two-wheeler (bike/scooter)
- Auto-rickshaw
- Bus
- Truck

Outputs per frame:
- Bounding box coordinates

- Class label

- Confidence score

Design note: Even if detection occasionally misses vehicles under heavy occlusion, tracking can help recover continuity across frames.

6.2 Multi-object tracking (MOT)

Goal: Maintain consistent identities (IDs) across frames.

Core requirements:
- Stable track IDs for vehicles inside the queue zone and near the stop line
- Track lifecycle: spawn → update → terminate when lost
- Trajectory recording for analytics (centroids over time)

Association logic (conceptual):
- Combine motion continuity (prediction + gating) with spatial overlap between frames
- Use temporal smoothing to reduce jitter and short false tracks
- Persist tracks across brief occlusions (common in dense intersections)

Why it matters:

Red-light jumping detection is highly sensitive to ID switches near the stop line. Queue metrics also depend on not double-counting the same vehicle due to ID fragmentation.

6.3 Queue analytics

Queue analytics are computed using one or more **Queue Measurement Zones (QMZ)** placed before the stop line.

6.3.1 Queue length (count-based)

Definition: number of *tracked vehicles* that are considered queued in QMZ.

A vehicle is counted in queue if:
1. Its centroid lies inside the QMZ
2. Its speed is below a threshold (`v_queue`)
3. It remains in the QMZ for a minimum dwell time (`dwell_min_seconds`)

This avoids counting vehicles that simply pass through the region.

6.3.2 Queue density / occupancy

Two robust representations are proposed:

1. **Density:** `queued_count / QMZ_area` (pixel-area if no calibration)
2. **Occupancy:** fraction of QMZ area covered by vehicle boxes

Occupancy is often more stable in Indian traffic because two-wheelers cluster tightly and may inflate counts without proportionate area usage.

6.3.3 Aggregation and smoothing

- Compute metrics at a fixed interval (e.g., per second)
- Apply a moving average (small window) to reduce noise
- Provide raw + smoothed trend views for clarity

6.4 Violation detection

6.4.1 Red-light jump detection

This violation requires:
- A defined **stop line** in image coordinates
- A **signal state timeline** (manual intervals or optional signal-ROI detection)

Core logic:
1. Detect a *crossing event* when a vehicle's tracked centroid moves from pre-stop to post-stop side
2. Confirm crossing over multiple frames to avoid jitter false positives
3. If crossing timestamp lies within a **RED** interval, flag as red-light violation

Evidence-first design:
Each violation stores:
- timestamp
- track ID + class
- evidence frames (before / at crossing / after)
- optional short clip
- clear reason string: "Crossed stop line during RED phase"

Precision-first approach:

The system is conservative to avoid false accusations, prioritizing precision over recall.

6.4.2 Rash driving detection (Explainable risk flags)

Rash driving is context-dependent. In this project, we treat it as **risk flags** using explainable heuristics rather than legal judgment.

Features computed per track:
- speed (pixel/sec or relative)

- acceleration (change in speed)

- lateral movement / curvature (trajectory shape)

- density context (queue occupancy when event occurs)

Example triggers (heuristic set):
- sudden acceleration or deceleration beyond threshold

- high speed through a dense region

- erratic lateral weaving (frequent heading changes)

- aggressive cut-in behavior (optional, conservative)

Explainability requirement:
For every rash flag:
- list which triggers fired

- show trigger values (speed/accel/curvature/occupancy)

- provide trajectory overlay evidence

---

7. System Design and Explainability

7.1 Modular design

The system is organized into clear modules:

- Video ingestion & configuration
- Detection
- Tracking
- Queue analytics
- Violation analytics
- Visualization & export
- Dashboard UI

This structure supports:
- easy replacement of detection/tracking models

- clearer debugging (module-wise outputs)

- better judge interpretability

7.2 Explainability principles used

To ensure the system is "audit-friendly," every output includes:

- a defined region (QMZ, stop line)
- an explicit rule (thresholds, dwell-time, crossing confirmation)
- a trace (track ID + frames/clip)
- a timestamp and confidence score

This makes the system a reasoning engine rather than "just model predictions."

---

8. Dashboard and Output Artifacts

8.1 NextJS dashboard

Pages/Tabs:
1. Upload & configure (ROI, stop line, signal schedule)
2. Queue analytics (charts + summary tiles)
3. Violations review (filterable event list + evidence viewer)
4. Outputs (annotated video + CSV/JSON downloads)

Queue analytics visuals:
- queue length vs time
- density/occupancy vs time
- peak queue timestamp
- optional class-wise distribution

Violation review visuals:
- red-light events list with timestamps and evidence
- rash risk flags list with triggers and trajectory overlay

8.2 Output artifacts (export)

For each run, export:
- `annotated.mp4` — with boxes, IDs, QMZ, stop line, and event markers
- `metrics_queue.csv` — time series metrics
- `violations.json/csv` — event logs with evidence references
- `config_used.json` — reproducible configuration

---

9. Sample Outputs (Schema / Mock Examples)

> The submission may include mock outputs if complete processing is not available. The structure below demonstrates what the system produces.

9.1 Queue metrics (example table)

Time (s)	Queue Count	Occupancy	Notes
10	6	0.22	queue forming
20	12	0.41	congestion rising
30	18	0.55	peak
40	9	0.30	dispersing (green phase)

9.2 Red-light violation record (example)

- **Type:** Red-light jump
- **Time:** 27.4s
- **Track ID:** 41
- **Class:** Two-wheeler
- **Reason:** Crossed stop line during RED phase
- **Evidence:** Frames at 26.9s (before), 27.4s (cross), 27.9s (after)

9.3 Rash driving risk flag (example)

- **Type:** Rash driving (risk flag)
- **Time:** 52.1s
- **Track ID:** 12
- **Class:** Car
- **Triggers:** sudden acceleration + high speed in dense region
- **Evidence:** trajectory overlay + speed curve snippet

---

10. Limitations

This is a prototype-grade vision analytics system. Key limitations include:

1. **Camera perspective effects:** pixel speed is not real-world speed without calibration.
2. **Occlusions in dense traffic:** heavy occlusions can cause ID switches or missed detections.
3. **Signal visibility:** if the traffic light is not visible, manual signal schedules are needed.
4. **Lane ambiguity:** weak lane discipline makes lane-specific queue estimation harder (bonus scope).
5. **Rash driving subjectivity:** heuristics can only provide risk flags, not legal conclusions.

The system therefore includes confidence scores and emphasizes evidence visualization.

---

11. Future Enhancements (Bonus Opportunities)

- **Multi-lane queue estimation** using multiple QMZs (per lane/approach)
- **Automatic signal state detection** using traffic-light ROI classification
- **Additional violations:** wrong-way driving, illegal U-turn, blocking pedestrian crossing
- **Calibration:** homography for approximate real-world speed estimation
- **Batch reporting:** summary PDF/CSV for multiple videos (intersection benchmarking)

---

12. Conclusion

TRAQ-Vision translates a real-world traffic challenge into a measurable, explainable video analytics pipeline. By combining multi-class vehicle detection, consistent multi-object tracking, queue analytics, and evidence-backed violation detection, the system supports both traffic operations and enforcement review.

The design prioritizes:
- interpretability and auditability
- modular pipeline structure
- conservative event flagging to reduce false positives

- clear visualization through annotated video and a NextJS dashboard

This aligns directly with the hackathon’s focus on **computer vision accuracy, tracking consistency, and explainable analytics**.

---

13. References (General)

- Multi-object tracking concepts (SORT/DeepSORT/ByteTrack families)
- ROI-based traffic metrics and occupancy estimation approaches
- Motion feature analysis for event detection

End of Report

> TEAM ORBIT