

**A PROJECT REPORT**  
**on**  
**Traffic Violation Detection System**

**Submitted to**  
**KIIT Deemed to be University**

**In Partial Fulfilment of the Requirement for the Award of**  
**BACHELOR'S DEGREE IN**  
**INFORMATION TECHNOLOGY**

**BY:**

**Tanishq Nimje - 2206061**  
**Aakriti Jha - 2206066**  
**Aakriti Arora - 2206067**  
**Anushka Tripathi - 2206076**  
**Madhur Mishra - 2206100**  
**Anmol Jaiswal - 2206165**

**UNDER THE GUIDANCE OF**

**Dr. Kumar Devdutta**



**SCHOOL OF COMPUTER ENGINEERING**  
**KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY**  
**BHUBANESWAR, ODISHA - 751024**

**KIIT Deemed to be University**

**School of Computer Engineering  
Bhubaneswar, ODISHA 751024**



## **CERTIFICATE**

**This is certify that the project entitled  
Traffic Violation Detection System**

**submitted by**

<b>Tanishq Nimje</b>	<b>2206061</b>
<b>Aakriti Jha</b>	<b>2206066</b>
<b>Aakriti Arora</b>	<b>2206067</b>
<b>Anushka Tripathi</b>	<b>2206076</b>
<b>Madhur Mishra</b>	<b>2206100</b>
<b>Anmol Jaiswal</b>	<b>2206165</b>

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Information Technology) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2024-2025, under our guidance.

Date: 05/04/2025

**Dr. Kumar Devdutta  
Project Guide**

## **Acknowledgements**

We are profoundly grateful to **Dr. Kumar Devdutta** of **Affiliation** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion. ....

**Tanishq Nimje**

**Aakriti Jha**

**Aakriti Arora**

**Anushka Tripathi**

**Madhur Mishra**

**Anmol Jaiswal**

# ABSTRACT

The Traffic Violation Detection System aims to enhance road safety by automatically identifying helmet and seatbelt violations using CCTV and dashcam footage. The system processes video feeds in real-time through machine learning models to detect violations, stores records in a structured database, and provides a user-friendly web interface for authorities to review and manage violations.

This project integrates computer vision techniques, deep learning models for detection, and a full-stack web application to streamline traffic violation monitoring. It addresses the critical need for automated enforcement systems that can operate continuously without manual intervention.

**Keywords:** Traffic violation detection, Helmet detection, Seatbelt detection, YOLOv8, Computer Vision, React.js, MongoDB

# CONTENTS

Chapter	Title	Page No.
1	Introduction	1
2	Basic Concepts/Literature Review	2
3	Problem Statement/Requirement Specs	3
	3.1 Project Planning	
	3.2 Project Analysis (SRS)	
	3.3 System Design	
4	Implementation	4
	4.1 Methodology/Proposal	
	4.2 Testing/Verification Plan	
	4.3 Result Analysis/Screenshots	

	4.4 Quality Assurance	
5	Standards Adopted	5
	5.1 Design Standards	
	5.2 Coding Standards	
	5.3 Testing Standards	
6	Conclusion and Future Scope	6
	References	7

# Chapter 1: Introduction

Road safety remains a critical concern globally, with traffic violations such as riding two-wheelers without helmets and driving cars without seatbelts contributing significantly to fatalities and injuries. Despite stringent regulations, compliance remains low in many regions due to inadequate enforcement mechanisms.

Traditional traffic violation monitoring relies heavily on manual observation by traffic police, which is both labor-intensive and prone to human error. Additionally, the limited coverage of manual monitoring means many violations go undetected, undermining the effectiveness of safety regulations.

The Traffic Violation Detection System addresses these challenges by employing computer vision and machine learning techniques to automatically detect traffic violations from CCTV and dashcam footage. By leveraging state-of-the-art object detection models, the system can identify whether motorcyclists are wearing helmets and whether car occupants are using seatbelts.

This report provides a comprehensive overview of the system's design, implementation, and evaluation. It begins with an exploration of the basic concepts and technologies used, followed by detailed requirements specifications. The implementation section outlines the methodologies employed, while subsequent sections cover testing procedures, adherence to industry standards, and potential future enhancements.

# Chapter 2: Basic Concepts/Literature Review

## 2.1 Computer Vision for Traffic Monitoring

Computer vision encompasses techniques for acquiring, processing, and analyzing digital images to extract meaningful information. In traffic monitoring, computer vision algorithms process video feeds to detect vehicles, identify their types, and monitor compliance with safety regulations.

## 2.2 Object Detection Using YOLOv8

You Only Look Once (YOLO) is a state-of-the-art real-time object detection system. YOLOv8, the latest iteration, offers improved accuracy and processing speed, making it ideal for real-time traffic violation detection. Unlike traditional computer vision approaches that use sliding windows, YOLO processes the entire image in a single forward pass of a neural network, enabling processing speeds of over 30 frames per second.

## 2.3 MongoDB for Database Management

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents. Its schema-less nature allows for storing complex hierarchical relationships between entities such as vehicle owners, vehicles, and violations, making it ideal for our application.

## 2.4 React.js for Frontend Development

React.js is a JavaScript library for building user interfaces, particularly single-page applications. Its component-based architecture enables the creation of reusable UI elements, facilitating the development of interactive dashboards for monitoring traffic violations.

## 2.5 RESTful API Design

Representational State Transfer (REST) is an architectural style for designing networked applications. RESTful APIs use HTTP requests to perform CRUD operations (Create, Read, Update, Delete) on resources, enabling communication between the frontend, backend, and ML components of the system.



# Chapter 3: Problem Statement / Requirement Specifications

## 3.2.1 Conceptual Design

The conceptual design of the Traffic Violation Detection System follows a three-layered architecture:

1. Data Acquisition Layer:
  - Processes video feeds from CCTV cameras and dashcams
  - Extracts frames at configurable intervals (default: 3 frames/second)
  - Performs initial preprocessing to normalize lighting and resolution
2. Detection Layer:
  - YOLOv8-based object detection for vehicles, helmets, and seatbelts
  - Classification of detected objects to identify violations
  - Confidence scoring mechanism (minimum threshold: 90% for helmet violations, 85% for seatbelt violations)
3. Application Layer:
  - REST API framework for communication between components
  - Database for violation storage and retrieval
  - Web interface for monitoring and reporting

The system employs a microservices architecture where each major function operates independently but communicates through well-defined interfaces.

## 3.2.2 Scrum Framework Implementation

### Scrum Overview

This project implements the Scrum framework, an empirical process control model based on transparency, inspection, and adaptation. Scrum was selected because:

- It enables rapid iteration on the ML models
- It accommodates changing requirements as violation detection accuracy improves
- It facilitates cross-functional collaboration between ML specialists, backend developers, and frontend developers

### Key Scrum Elements Implemented

- Sprint Duration: Two-week sprints to deliver incremental value
- Scrum Ceremonies: Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective
- Scrum Artifacts: Product Backlog, Sprint Backlog, and Increment with Definition of Done

### 3.2.3 Team Formation and Roles

The Scrum Team consists of 7 members with cross-functional skills and follows the self-managing principle, meaning they internally decide who does what, when, and how.

#### Scrum Roles

1. Product Owner:
  - Responsible for maximizing product value
  - Manages the Product Backlog, including ordering items based on priority
  - Makes final decisions about feature implementation
2. Scrum Master:
  - Ensures Scrum practices are understood and followed
  - Facilitates Scrum events and removes impediments
  - Coaches the team in self-organization and cross-functionality
3. Development Team:
  - Cross-functional group responsible for delivering the product increment
  - Includes expertise in ML model development, backend systems, frontend interfaces, and testing
  - Collectively responsible for meeting sprint goals

### 3.2.4 Responsibility Sharing and Delegation

A formal Delegation Board was established to clarify decision-making authorities using the following levels:

1. Tell (Level 1): Decision made by authority figure who informs the team
2. Sell (Level 2): Decision made by authority figure who convinces the team
3. Consult (Level 3): Decision made after consulting with the team
4. Agree (Level 4): Consensus-based decision making
5. Advise (Level 5): Team decides, leader advises
6. Inquire (Level 6): Team decides, then informs
7. Delegate (Level 7): Full delegation, no reporting required

#### Delegation Examples

Task	Assigned To	Delegation Level
ML Model Development	ML Specialists	Level 1 (Tell)
Backend API Design	Backend Developers	Level 3 (Consult)
Frontend Implementation	Frontend Developers	Level 4 (Agree)
Testing Strategy	Testers	Level 5 (Advise)

## 3.2.5 Process to Get Work Done

### Sprint Planning

1. Preparation: Product Owner ensures Product Backlog items are refined
2. Sprint Goal Definition: Team collaboratively defines the goal for the upcoming sprint
3. Work Selection: Development Team selects Product Backlog items they can complete
4. Task Breakdown: Selected items are decomposed into tasks of 4-8 hours each
5. Capacity Planning: Team ensures selected work matches available capacity

### Daily Work Execution

1. Daily Scrum: 15-minute synchronization meeting focused on progress toward Sprint Goal
2. Visualization: Tasks tracked on digital board with "To Do," "In Progress," and "Done" columns
3. Impediment Management: Issues raised to Scrum Master for quick resolution
4. Pair Programming: Critical components developed in pairs to ensure quality

## 3.2.6 Reviewing Work and Retrospecting

### Sprint Review

- Timeboxed to 2 hours at the end of each sprint
- Stakeholders invited to view working product increment
- Demonstration of completed features with real-time violation detection
- Feedback captured for Product Backlog refinement

### Sprint Retrospective

The team follows a structured retrospective format after each sprint:

1. Set the Stage: Brief check-in to establish context (5 minutes)

2. Data Gathering: Team identifies what went well and what could improve (15 minutes)
3. Insight Generation: Team analyzes root causes of issues (15 minutes)
4. Decision Making: Team selects one improvement to implement in next sprint (15 minutes)
5. Closure: Team commits to action items (10 minutes)

Sample retrospective questions:

- "What did we do well that we should continue doing?"
- "What did we learn in this sprint?"
- "What should we do differently next time?"
- "What still puzzles us?"

### 3.2.7 Testing and Approaches

#### Testing Strategy

The project implements a balanced testing approach:

1. Shift-Left Testing: Early testing throughout development
  - Unit testing for individual components
  - Integration testing for component interactions
  - ML model validation against labeled datasets
2. Shift-Right Testing: Testing in production-like environments
  - Performance testing under various lighting conditions
  - User acceptance testing with traffic authorities
  - A/B testing of detection algorithms

#### Test Automation

- Unit Tests: PyTest for ML components, Jest for frontend
- Integration Tests: API testing with Postman collections
- End-to-End Tests: Selenium for web interface validation
- Continuous Integration: Tests run automatically on each commit

### 3.2.8 Test Documentation

Testing activities are documented using the following formats:

1. Test Plan: High-level document outlining testing strategy and scope
2. Test Cases: Detailed scenarios with steps, expected results, and actual results
3. Defect Reports: Structured documentation of issues found during testing

Sample test documentation template:

Test ID	Test Case Title	Test Condition	System Behavior	Expected Result
T01	Helmet Detection Accuracy	Motorcyclist without helmet	Detect violation	Violation detected $\geq 90\%$ confidence
T02	Seatbelt Detection	Car driver without seatbelt	Detect violation	Violation detected $\geq 85\%$ confidence

### 3.2.9 Delegation of Required Designs

The design delegation followed a structured approach:

1. High-Level Architecture: Collaborative design in a workshop with entire team
2. ML Model Architecture: Delegated to ML specialists with regular reviews
3. Database Schema: Initially proposed by backend developers, refined with team input
4. API Design: Created by backend developers following REST principles
5. UI/UX Design: Wireframes developed collaboratively, detailed design by UI specialists

The delegation process used the "Consult" level (Level 3) for most design decisions, ensuring experts could make decisions while incorporating team feedback.

### 3.2.10 Scope of Work

#### In Scope:

- Detection of helmet violations for two-wheeler riders
- Detection of seatbelt violations for car drivers and front passengers
- Real-time alerts for traffic authorities
- Historical violation data storage and reporting
- Web-based administrative interface

#### Out of Scope:

- Speed violation detection
- License plate recognition (planned for future iterations)
- Automated ticketing system integration
- Mobile application for traffic officers (planned for future iterations)

### 3.2.11 Communication Plan

#### Meetings Schedule:

- Sprint Planning: Bi-weekly, 4 hours, entire team
- Daily Scrum: Daily, 15 minutes, entire team
- Sprint Review: Bi-weekly, 2 hours, team and stakeholders
- Sprint Retrospective: Bi-weekly, 1 hour, entire team
- Backlog Refinement: Weekly, 2 hours, Product Owner and selected team members

#### Communication Channels:

- Slack: Daily communication and updates
- Jira: Task tracking and status updates
- Confluence: Documentation repository
- Email: Formal communication with external stakeholders

### 3.2.12 Tech Stack

Component	Technology	Justification
ML Framework	PyTorch/YOLOv8	High accuracy object detection capabilities
Backend	Django/Python	Robust API development
Database	MongoDB	Flexible schema design
Frontend	React.js	Component-based architecture
Cloud Infrastructure	AWS EC2/S3	Scalable hosting

The system avoids subscription-based services where possible, relying on open-source alternatives to reduce operational costs.

### 3.2.13 Data Storage Architecture

## **User Data Storage:**

- Traffic officers' accounts and access levels
- Administrative user management
- Authentication and authorization records

## **Violation Data Storage:**

Designed with atomic data principles ensuring each piece of data cannot be further decomposed:

1. Vehicle Records: Make, model, color, license plate (if detected)
2. Violation Records: Type, timestamp, location, confidence score
3. Evidence Storage: Image snapshots, video clips, metadata

### **3.2.14 Scrum Language Key Terms**

To ensure common understanding across the team, these key Scrum terms are defined:

- Sprint: A time-boxed period of 2 weeks during which specific work is completed and made ready for review
- Product Backlog: The prioritized list of features and requirements for the system
- Sprint Backlog: The subset of Product Backlog items selected for implementation in the current Sprint
- Daily Scrum: A 15-minute daily synchronization meeting for the Development Team
- Definition of Done: The shared understanding of what it means for work to be complete
- Sprint Review: A meeting held at the end of each Sprint to inspect the increment and adapt the Product Backlog
- Sprint Retrospective: A meeting held after the Sprint Review to inspect and adapt the process
- Increment: The sum of all completed Product Backlog items during a Sprint and all previous Sprints

### **3.2.15 System Capabilities and Offerings**

The Traffic Violation Detection System offers:

1. Automated Violation Detection: Real-time identification of traffic violations without manual monitoring
2. Evidence Collection: Automatic capture and storage of violation evidence
3. Reporting Dashboard: Visualization of violation trends and statistics
4. Alert Management: Real-time notifications for traffic authorities
5. Historical Analysis: Tools for analyzing violation patterns over time

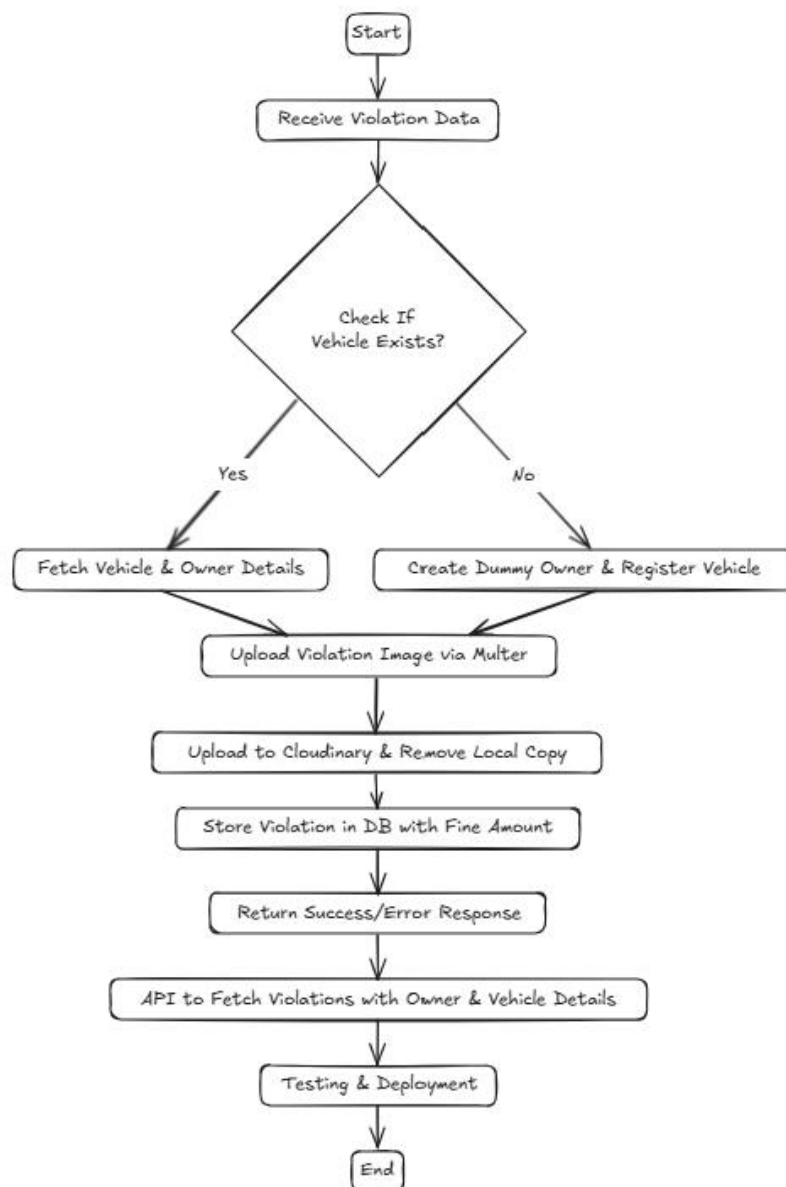
## 6. Audit Trail: Complete record of system activities for transparency

The system operates continuously, processing video feeds from multiple sources simultaneously to provide authorities with actionable insights about traffic violations.

### 3.2.16 System Architecture

#### Flowchart1: End-to-End Workflow

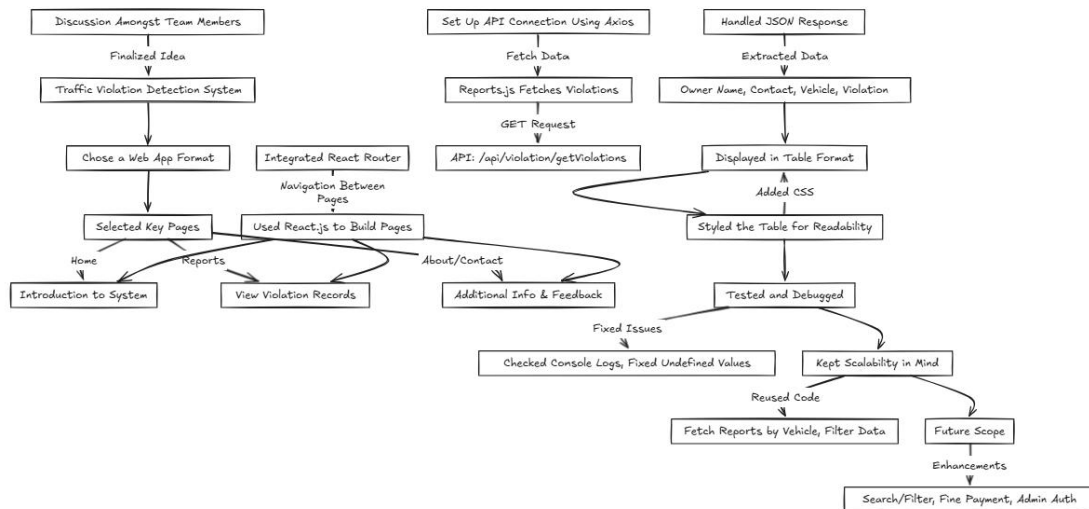
The following flowchart illustrates the complete end-to-end workflow of the Traffic Violation Detection System, from input data preprocessing to report generation:





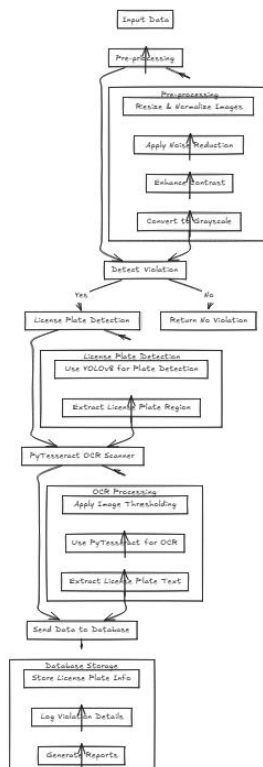
## Flowchart 2: Backend Workflow

The following flowchart represents the backend workflow for handling violation data, including vehicle registration checks, image uploads, and database storage:



## Flowchart 3: Web Application Workflow

The following flowchart outlines the development process of the web application, including frontend design, API integration, data handling, and testing:

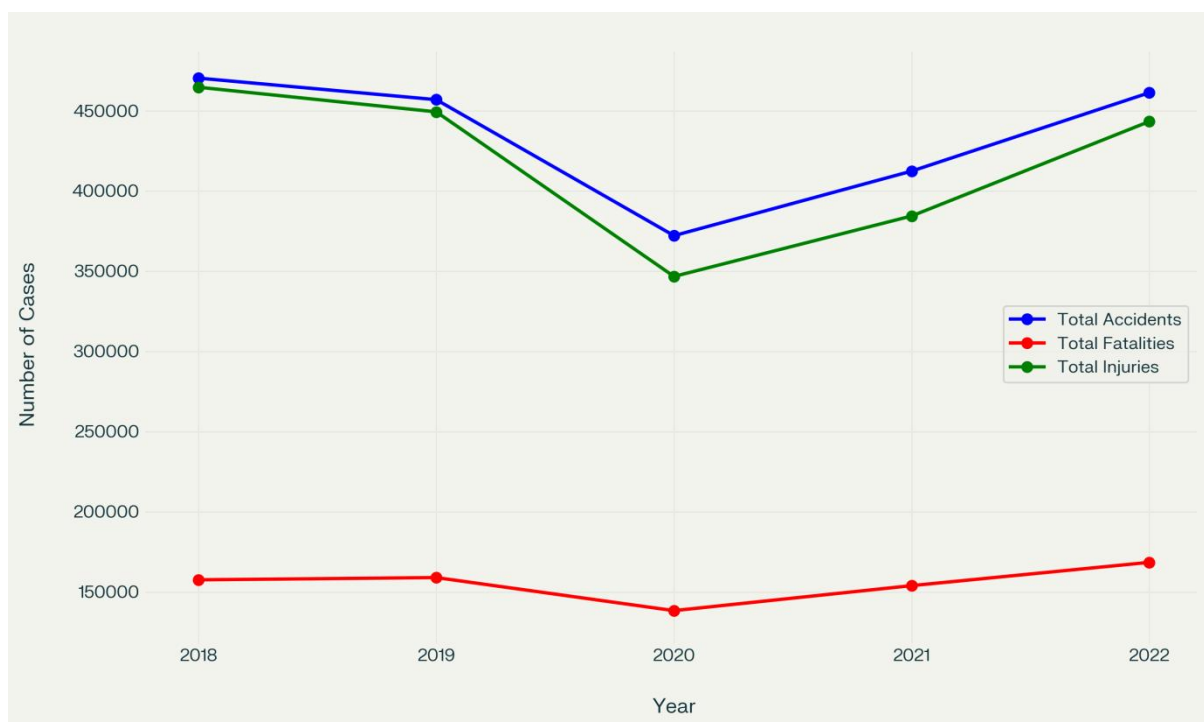


# Chapter 4: Implementation

## 4.1 Methodology / Proposal

### 4.1.1 Road Accidents and Trends in India:

The following graph illustrates the trends in road accidents, fatalities, and injuries in India over five years (2018–2022):



### Explanation of the Graph

#### 1. Key Observations:

- The total number of road accidents decreased significantly in 2020 due to COVID-19 lockdowns but increased again in 2021 and 2022.
- Fatalities have risen steadily from 2018 to 2022, reaching a peak of 168,491 deaths in 2022.
- Injuries followed a similar trend, with a sharp drop in 2020 but increasing again in subsequent years.

#### 2. Insights:

- The consistent rise in fatalities highlights the urgent need for stricter enforcement of traffic regulations.

- The increase in accidents and injuries post-2020 indicates a return to pre-pandemic traffic volumes and challenges.
3. **Relevance to the Project:**
- The Traffic Violation Detection System aims to reduce such incidents by enforcing helmet and seatbelt compliance.
  - Real-time detection and reporting can help authorities address violations proactively, potentially reducing fatalities and injuries.

### 4.1.2 Machine Learning Model Development

Multiple approaches were explored for helmet and seatbelt detection:

1. YOLOv8 Implementation:
  - Customized YOLOv8 models trained on datasets containing images of motorcyclists with/without helmets and car occupants with/without seatbelts
  - Training configuration: 100 epochs, batch size 16, image size 640×640
  - Data augmentation techniques applied: random flip, rotation, brightness adjustment
2. Sequential Model Testing:
  - Implemented alternative CNN architectures for comparative analysis
  - Evaluated based on precision, recall, and F1-score metrics
3. Model Selection:
  - YOLOv8 chosen for its superior balance between speed (30+ FPS) and accuracy
  - Final model achieved 92% accuracy for helmet detection and 88% for seatbelt detection

### 4.1.3 Backend Development

The backend system followed a structured development approach:

1. Data Modeling:
  - Three primary entities: Owner, Vehicle, and Violation
  - Relationships defined: Vehicle belongs to Owner, Violation linked to Vehicle
  - MongoDB schemas implemented using Mongoose
2. API Development:
  - RESTful endpoints created for core operations:
    - `/api/violation/create`: Processes new violations

- [/api/violation/getViolations](#): Retrieves violation records
- Image processing using Multer for handling uploads and Cloudinary for storage

### 3. Integration with ML Model:

- API endpoints developed to expose ML model functionality
- Video feeds processed frame-by-frame
- Detection results stored with relevant metadata

## 4.1.4 Frontend Development

The frontend implementation followed the workflow illustrated in the provided flowchart:

### 1. Component Design:

- Home.js: Project introduction and overview
- Reports.js: Violation data display in tabular format
- About/Contact.js: Additional information and feedback options

### 2. Data Retrieval:

- Axios library used for API communication
- Violation data fetched from [/api/violation/getViolations](#) endpoint

### 3. User Interface:

- Tabular display of violation data including owner details, vehicle information, violation type, and images
- CSS styling for enhanced readability
- Responsive design for cross-device compatibility

## 4.2 Testing / Verification Plan

Test ID	Test Case Title	Test Condition	System Behavior	Expected Result
---------	-----------------	----------------	-----------------	-----------------

T01	Helmet Detection Accuracy	Motorcyclist without helmet	System processes video frame	Violation detected with $\geq 90\%$ confidence
-----	---------------------------	-----------------------------	------------------------------	--

T02	Seatbelt Detection Accuracy	Car driver without seatbelt	System processes video frame	Violation detected with $\geq 85\%$ confidence
T03	Real-time Processing	Multiple camera feeds	System handles simultaneous streams	All streams processed at $\geq 30$ FPS
T04	Vehicle Registration Check	New license plate submitted	System creates placeholder owner	Vehicle and owner records created successfully
T05	Violation Creation	Valid violation data submitted	System processes and stores violation	Violation stored with correct references
T06	Image Processing	Image uploaded with violation	System uploads to Cloudinary	Image URL stored with violation record
T07	API Response Format	GET request to <code>/api/violation/getViolations</code>	System queries database	JSON with populated vehicle and owner details
T08	Frontend Rendering	Reports page loaded	System fetches and displays violations	Data presented in formatted table

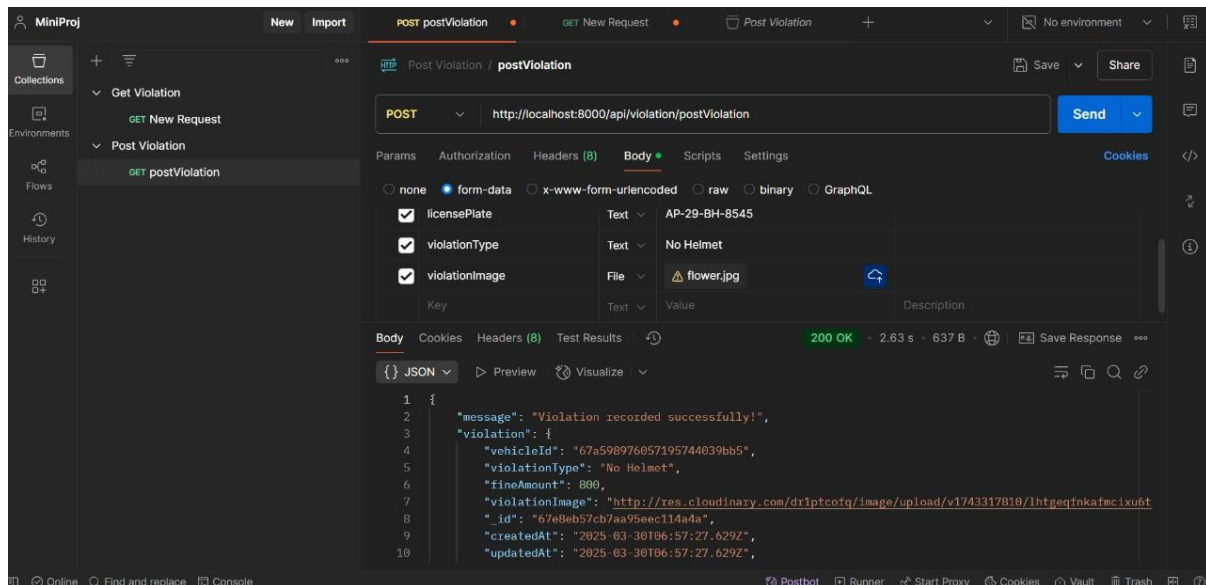
### 4.3 Result Analysis / Screenshots

### 4.3.1 Backend API Testing :

The following screenshots demonstrate the functionality of the backend APIs for the Traffic Violation Detection System. These APIs handle violation data retrieval and submission.

#### Image 1: GET Request - Retrieve Violations

This screenshot shows the backend API endpoint `/api/violation/getViolations` being tested in Postman. The API retrieves all recorded violations along with detailed information, including vehicle ID, license plate, owner ID, and violation type.

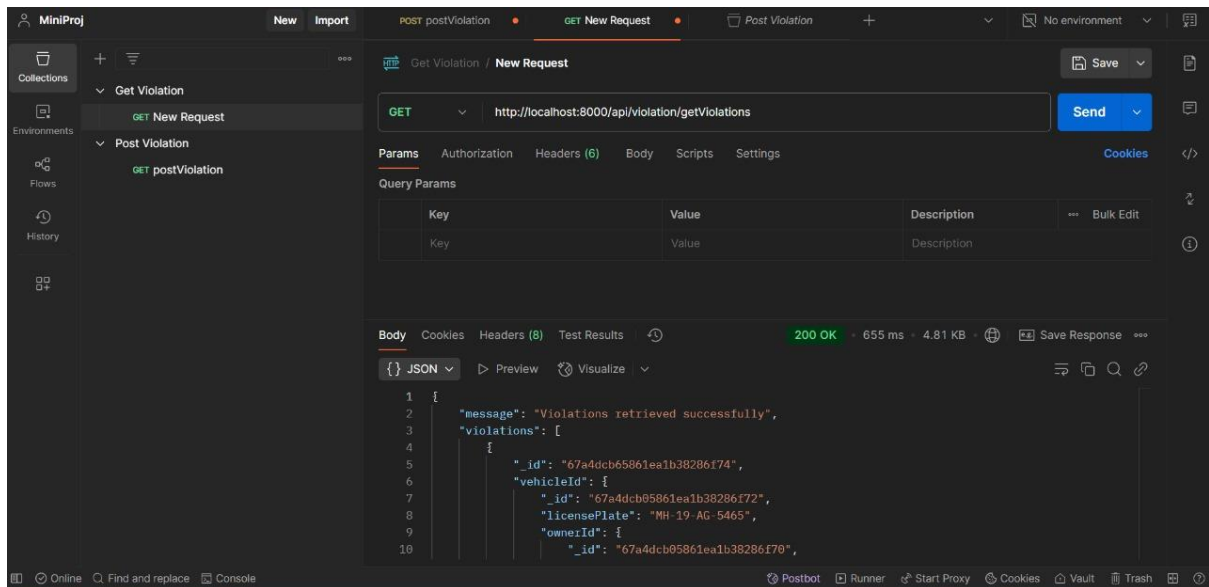


#### Explanation:

- The API successfully returns a JSON response containing:
  1. A message confirming successful retrieval.
  2. An array of violations with metadata such as vehicle ID, license plate number, owner ID, and violation type.
- HTTP Status Code: 200 OK.

#### Image 2: POST Request - Record Violation

This screenshot demonstrates the backend API endpoint `/api/violation/postViolation` being tested in Postman. The API records a new violation by accepting inputs such as license plate, violation type, and an image file.



### Explanation:

- The API accepts the following inputs:
  1. licensePlate: Vehicle's license plate number.
  2. violationType: Type of violation (e.g., "No Helmet").
  3. violationImage: Image evidence of the violation.
- The API successfully stores the violation in MongoDB and uploads the image to Cloudinary.
- HTTP Status Code: 200 OK.
- The response includes:
  1. A success message.
  2. Metadata such as vehicle ID, violation type, fine amount, and Cloudinary image URL.

### 4.3.2 Frontend Interface :

The following screenshots demonstrate the user interface of the Traffic Violation Detection System:

#### Image 1: Registration Page

This screenshot shows the registration page where users can create an account by providing their details, including optional vehicle and dashcam information.:

# Register

☐ Do you own a vehicle?

☐ Do you have a dashcam?

Register

[Already have an account? Login](#)

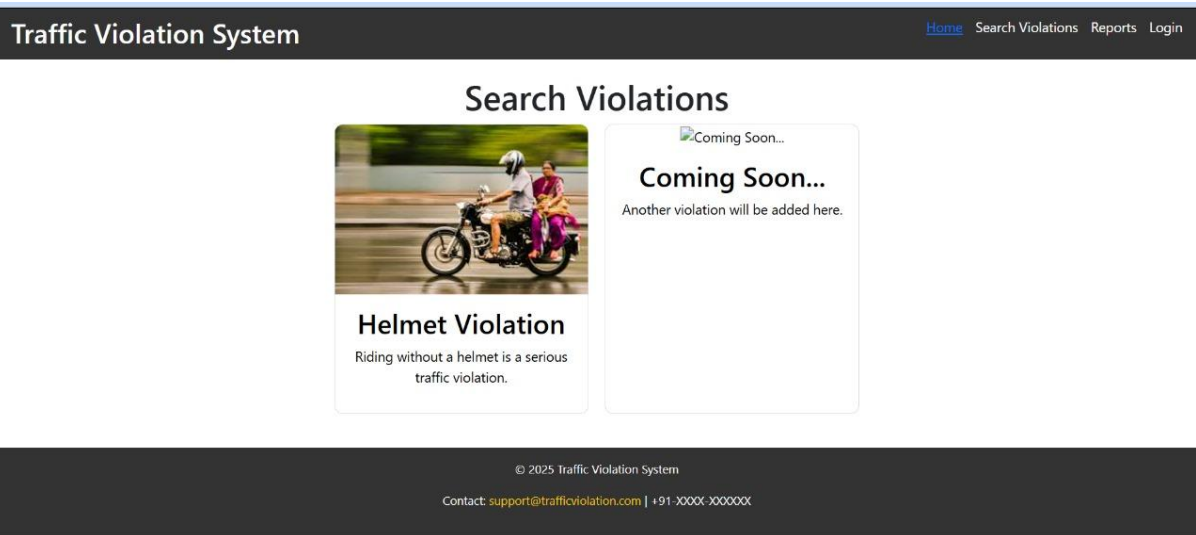
## Explanation:

- Fields include Full Name, Email, Username, Password, and Confirm Password
- Users can specify whether they own a vehicle or have a dashcam.
- Optional input for license plate details is provided for vehicle owners.

## Image 2: Search Violations Page

This screenshot displays the "Search Violations" page, where users can view detected violations such as helmet violations.





**Explanation:**

- The page highlights helmet violations with an image and description.
- Placeholder for future violations (e.g., seatbelt violations) is included with a "Coming Soon" message.
- Navigation options are visible at the top (Home, Search Violations, Reports, Login).

**Image 3: Home Page - Helmet Detection**

This screenshot showcases the home page of the Traffic Violation Detection System with an example of helmet violation detection.









**Explanation:**

- The system detects helmet violations using AI and dashcam footage.
- An example image is displayed with "No Helmet Detected" text overlay.
- Contact details for support are provided at the bottom of the page.

#### Image 4: Traffic Violations Report Page

This screenshot displays the "Traffic Violations Report" page, where users can view detailed records of detected violations.

Traffic Violation System						
<a href="#">Home</a> <a href="#">Search Violations</a> <a href="#">Reports</a> <a href="#">Login</a>						
Traffic Violations Report						
#	License Plate	Owner Name	Violation Type	Fine Amount (₹)	Violation Image	Date
1	MH-19-AG-5465	Anmol Jaiswal	Overloading Vehicle	600		2/6/2025
2	MH-19-AG-5465	Anmol Jaiswal	No Helmet	800		2/6/2025
3	MH-19-AG-5465	Anmol Jaiswal	Over-Speeding	1000		2/6/2025
4	AP-29-BH-8545	Madhur Mishra	Over-Speeding	1000		2/7/2025
5	AP-29-BH-8545	Madhur Mishra	Signal-Jumping	500		2/7/2025
6	AP-29-BH-8545	Madhur Mishra	Signal-Jumping	500		3/22/2025

#### Explanation:

- The table includes columns for License Plate, Owner Name, Violation Type, Fine Amount, Violation Image, and Date.
- Each row represents a unique violation with corresponding metadata.
- The page provides a structured view for easy analysis by traffic authorities.

#### 4.3.3 Input Codes :

The following screenshots show the backend code responsible for detecting violations, preprocessing license plates, and sending data to the API.:

#### Image 1: Sending Violation Data to the API

This screenshot demonstrates the Python function `send_violation_data`, which sends violation details (license plate, type of violation, and image) to the backend API.:

```
[ ] # Image Path
import requests
IMAGE_PATH = preprocess_image("/content/drive/MyDrive/FINAL/helmet.jpg")

# Step 1: Detect Helmet Violations
helmet_image, violation_regions = detect_helmet_violation(IMAGE_PATH)

# Step 2: Detect License Plates and Extract Texts if Violations are Found
if violation_regions:
    print("\nProcessing License Plates...")
    license_plate_texts = detect_license_plate(IMAGE_PATH)
    print(license_plate_texts)
    api_url = "https://liscenceplate.onrender.com/api/violation/postViolation"
    license_plate = license_plate_texts
    violation_type = "Helmet Violation"
    response = send_violation_data(api_url, license_plate, violation_type, IMAGE_PATH)
    print(response)
else:
    print("\nNo Violations Detected. Skipping License Plate Detection.")
```

### Explanation:

- The function uses Python's requests library to send a POST request with violation details.
- It include
  1. licensePlate: The detected license plate number.
  2. violationType: The type of violation (e.g., "Helmet Violation").
  3. violationImage: The image file associated with the violation.
- The function handles errors using exception handling and logs status codes and responses for debugging purposes.

### Image 2: Preprocessing License Plate Images

This screenshot shows the function preprocess\_license\_plate, which processes license plate images for better OCR accuracy.

```
[ ] import os
import requests

def send_violation_data(api_url, licenseplate, violationtype, image_path):
    if not os.path.exists(image_path):
        return {"error": f"File not found: {image_path}"}

    try:
        with open(image_path, "rb") as img_file:
            data = {"licensePlate": licenseplate, "violationType": violationtype}
            files = {"violationImage": img_file}

            print("📁 Sending Data:", data)
            print("📁 Sending File:", image_path)

            response = requests.post(api_url, data=data, files=files)
            print("📡 Status Code:", response.status_code)
            print("📡 Response Text:", response.text)

            response.raise_for_status()
            return response.json()

    except requests.exceptions.RequestException as e:
        return {"error": f"API request failed: {e}"}
```

Figure 4.1: Preprocessing License Plate Images

### Explanation:

- The function converts the image to grayscale and applies noise reduction using a bilateral filter.
- Adaptive thresholding is applied to enhance text clarity before OCR processing.
- This preprocessing step improves OCR performance for extracting text from license plates.

```
[ ] def preprocess_license_plate(cropped_image):  
    gray = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2GRAY) # Grayscale  
    blurred = cv2.bilateralFilter(gray, 11, 17, 17) # Noise reduction  
    thresh = cv2.adaptiveThreshold(  
        blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2  
    )  
    return thresh
```

Figure 4.2: Detecting Helmet Violations and License Plates

### Explanation:

- Step 1 - Helmet Detection: Adaptive thresholding is applied to enhance text clarity before OCR processing.
  1. The system detects helmet violations using a trained YOLO model.
  2. If no violations are detected, it skips further processing.
- Step 2 - License Plate Detection: The system detects helmet violations using a trained YOLO model.
  1. If violations are detected, the system proceeds to detect license plates in the same image.
  2. Extracted license plate text is sent to the backend API along with violation details.
- API Integration:
  1. The processed data is sent via a POST request to the backend API endpoint `/api/violation/postViolation`.

## 4.3.4 Output Images :

The following screenshots illustrate the output generated by the Traffic Violation Detection System:

### Image 1: License Plate Processing

This screenshot shows the system processing a license plate detected on a two-wheeler.



**Explanation:**

- The system successfully detects the license plate on the vehicle.
- The text "OD 33 P3754" is extracted from the license plate using OCR (Optical Character Recognition).
- This information is stored in the database along with other violation details.

**Image 2: Helmet Violation Detection**

This screenshot demonstrates the detection of a helmet violation.



**Explanation:**

- The system identifies that the rider is not wearing a helmet.

- A red bounding box is drawn around the rider's face, with a label "Without Helmet."
- The violation details, including license plate number and timestamp, are stored in the database for further action.

## 4.4 Quality Assurance

Quality assurance was implemented through:

1. Code Reviews:
  - All code changes underwent peer review before integration
  - Static code analysis tools used to identify potential issues
2. Automated Testing:
  - Unit tests for backend API endpoints
  - Integration tests for ML model accuracy
  - End-to-end tests for frontend functionality
3. Performance Monitoring:
  - System performance metrics tracked during testing
  - Optimization applied to components that didn't meet performance targets



# Chapter 5: Standards Adopted

## 5.1 Design Standards

1. UML Diagrams:
  - Use Case Diagrams for user interactions
  - Sequence Diagrams for system workflows
  - Class Diagrams for data models
2. REST API Design:
  - Resource-oriented architecture
  - Stateless communication
  - Standard HTTP methods (GET, POST, PUT, DELETE)
3. UI/UX Standards:
  - Material Design guidelines for consistent interface
  - WCAG 2.1 accessibility standards

## 5.2 Coding Standards

1. JavaScript/React:
  - ESLint configuration for code style enforcement
  - Component-based architecture
  - Proper state management using hooks
2. Python/ML:
  - PEP 8 style guide compliance
  - Docstrings for function documentation
  - Type hints for improved code clarity
3. Node.js/Express:
  - Modular architecture with separation of concerns
  - Consistent error handling
  - Asynchronous patterns (async/await)

## 5.3 Testing Standards

1. Unit Testing:
  - Jest for JavaScript/React components
  - PyTest for Python ML models
  - Code coverage targets of >80%

2. Integration Testing:

- API testing using Postman collections
- End-to-end testing with Cypress

3. Performance Testing:

- Load testing with Apache JMeter
- Benchmarking against defined performance metrics



# Chapter 6: Conclusion and Future Scope

## 6.1 Conclusion

The Traffic Violation Detection System successfully demonstrates the application of machine learning and computer vision techniques for automating traffic violation detection. By integrating YOLOv8-based object detection models with a web-based management interface, the system provides an effective solution for monitoring helmet and seatbelt compliance.

Key achievements include:

- Real-time detection of helmet and seatbelt violations with high accuracy
- Efficient data storage and retrieval through MongoDB
- User-friendly interface for reviewing violations using React.js
- Scalable architecture supporting multiple camera feeds

The system addresses the limitations of manual monitoring by providing continuous, automated surveillance that can potentially improve compliance with safety regulations and, consequently, reduce traffic-related injuries and fatalities.

## 6.2 Future Scope

The system has been designed with scalability in mind, allowing for several potential enhancements:

1. Extended Detection Capabilities:
  - Integration of license plate recognition
  - Detection of additional violations (e.g., mobile phone usage, traffic signal violations)
  - Vehicle speed estimation
2. Advanced User Interface:
  - Implementation of search and filtering functionality
  - Integration of maps for violation location visualization
  - Mobile application development for on-the-go access
3. System Integration:
  - Connection with traffic management systems
  - Integration with e-challan (electronic ticket) systems
  - Implementation of fine payment processing
4. Enhanced Security:
  - Implementation of role-based authentication
  - Advanced encryption for sensitive data

- Audit logging for system activities
5. Performance Optimization:
- Edge computing implementation for reduced latency
  - Model optimization for improved inference speed
  - Distributed processing for handling larger camera networks

## References

1. Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767.
2. Ultralytics. (2023). YOLOv8 Documentation. Retrieved from <https://docs.ultralytics.com/>
3. MongoDB, Inc. (2023). MongoDB Documentation. Retrieved from <https://docs.mongodb.com/>
4. React.js Documentation. (2023). Retrieved from <https://reactjs.org/docs/getting-started.html>
5. Express.js Documentation. (2023). Retrieved from <https://expressjs.com/>
6. World Health Organization. (2022). Global status report on road safety 2022.
7. Node.js Documentation. (2023). Retrieved from <https://nodejs.org/en/docs/>

# Individual Contributions

## Madhur Mishra:

- Worked on the helmet detection model alongside Tanishq Nimje.
- Explored multiple approaches for detection, including YOLO, sequential models, and APIs
- Selected the most accurate model based on performance metrics such as precision and recall
- Integrated the backend with the ML model to ensure seamless communication between components.

## Tanishq Nimje:

- Collaborated on developing the helmet detection model with Madhur Mishra.
- Tested various architectures and approaches to make the model more agile and efficient.
- Assisted in evaluating detection accuracy and optimizing the final YOLO-based solution.

## Aakriti Jha:

- Focused on backend development for the system.
- Implemented RESTful APIs for violation data submission (/postViolation) and retrieval (/getViolations).
- Designed database schemas in MongoDB for storing violation details, vehicle information, and evidence images.
- Integrated Cloudinary for image storage and ensured secure handling of uploaded data.

## Aakriti Arora:

- Worked on frontend development using React.js.
- Designed key pages such as:
  1. Registration Page
  2. Search Violations Page
  3. Traffic Violations Report Page
  4. Home Page showcasing helmet violation detection.

- Integrated Axios for API communication to fetch and display violation data in a structured format.
- Styled the frontend interface using CSS for readability and responsiveness.

### Anmol Jaiswal:

- Trained the machine learning models for helmet detection and license plate recognition.
- Prepared datasets containing labeled images of motorcyclists with/without helmets and license plates.
- Optimized YOLOv8 models for real-time performance, achieving high accuracy ( $\geq 90\%$  for helmet detection).
- Conducted testing to evaluate model precision, recall, and F1-score under varied conditions.

### Anushka Tripathi:

- Worked on project documentation, ensuring all technical aspects were clearly explained.
- Created user interface low-fidelity wireframes to guide frontend development.
- Compiled detailed reports on system workflows, testing strategies, and future enhancements.
- Ensured adherence to IEEE standards for Software Requirements Specification (SRS).

This report has been prepared in compliance with the requirements of KIIT Deemed to be University for the award of Bachelor's Degree in Information Technology.