PROJECT REPORT

Smart Traffic Management System

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

Computer Science and Engineering

Submitted by

Satwik Dobhal	**RollNo. 2119135**
Shashwat Gauniyal	**RollNo. 2119164**
Mohit Singh Rawat	**RollNo. 2118807**
Kanishka Thapa	**RollNo. 2118674**

Under the Guidance of

Ms. Swati Joshi

Lecturer

Project Group Number: 66

Department of Computer Science and Engineering

Graphic Era Hill University

June, 2025

Graphic Era
HILL UNIVERSITY
Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project progress report entitled Smart Traffic Management System in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering in the Department of Computer Science and Engineering of the Graphic Era Hill University, Dehradun shall be carried out by the undersigned under the supervision of Ms. Swati Joshi, Lecturer, Department of Computer Science and Engineering, Graphic Era Hill University, Dehradun.

Satwik Dobhal	(RollNo. 2119135)
Shashwat Gauniyal	(RollNo. 2119164)
Mohit Singh Rawat	(RollNo. 2118807)
Kanishka Thapa	(RollNo. 2118674)

The above-mentioned students shall be working under the supervision of the undersigned on the topic Smart Traffic Management System.

Guide

Date

Place

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project guide, Ms. Swati Joshi, for providing invaluable guidance, continuous support, and encouragement throughout the development of this project. His expertise and insights have been instrumental in shaping our understanding of computer vision systems and their applications in traffic management.

We extend our thanks to the Head of Department, Prof. Anupam Singh, and the entire faculty of the Computer Science Department at Graphic Era Hill University for providing us with the necessary resources, academic knowledge, and infrastructure to complete this project successfully.

We are also grateful to our peers who provided constructive feedback during the development process, which helped us refine our system's functionality and user interface.

Finally, we would like to thank our families for their unwavering support and understanding during the long hours we dedicated to this project.

Name: Satwik Dobhal

Roll No: 2119135

Date: 29/05/2025

Name: Shashwat Gauniyal

Roll No: 2119164

Date: 29/05/2025

Name: Mohit Singh Rawat

Roll No: 2118807

Date: 29/05/2025

Name: Kanishka Thapa

Roll No: 2118674

Date: 29/05/2025

ABSTRACT

This project presents an AI-powered Smart Traffic Management System designed to address the increasing challenges of urban traffic congestion, road safety, and emergency response. The system leverages computer vision, machine learning, and web technologies to create a comprehensive solution for modern traffic management needs.

The Smart Traffic Management System utilizes real-time video analytics to monitor traffic flow, detect vehicles, identify traffic violations, and recognize emergency vehicles. The system's adaptive traffic signal control algorithm optimizes signal timing based on current traffic density, prioritizes routes for emergency vehicles, and ensures fair distribution of green signal time across intersections.

A key feature of the system is its ability to automatically generate e-challans (electronic tickets) for traffic violations, capturing vehicle details and infringement evidence. Additionally, the system incorporates a women's safety module that detects distress signals through audio monitoring and raises appropriate alerts.

The implementation includes a responsive web dashboard built using React.js for the frontend and FastAPI for the backend services. The dashboard provides real-time traffic analytics, violation records, signal status monitoring, and safety alerts through an intuitive user interface that supports both light and dark themes.

Testing results demonstrate the system's effectiveness in reducing congestion, improving emergency response times, and enhancing road safety through automated violation detection. The solution offers significant advantages over traditional fixed-time traffic systems by adapting to real-time conditions and providing valuable data insights for urban planning.

This project contributes to the advancement of smart city infrastructure by showcasing how artificial intelligence can be applied to create more efficient, safe, and responsive urban traffic management systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AI	-	Artificial Intelligence
API	-	Application Programming Interface
CCTV	-	Closed-Circuit Television
CSS	-	Cascading Style Sheets
CV	-	Computer Vision
FASTAPI	-	Fast Application Programming Interface
FPS	-	Frames Per Second
GPU	-	Graphics Processing Unit
HTML	-	Hypertext Markup Language
HTTP	-	Hypertext Transfer Protocol
IoT	-	Internet of Things
JSON	-	JavaScript Object Notation
ML	-	Machine Learning
NMS	-	Non-Maximum Suppression
OCR	-	Optical Character Recognition
REST	-	Representational State Transfer
RGB	-	Red Green Blue
SORT	-	Simple Online and Realtime Tracking
UI	-	User Interface
URL	-	Uniform Resource Locator
YOLO	-	You Only Look Once

NOTATIONS

α	-	Learning rate in object detection algorithms
β	-	Confidence threshold for object detection
γ	-	Intersection over Union (IoU) threshold
θ	-	Angle parameter in lane detection
τ	-	Time interval for traffic signal switching
ρ	-	Density of vehicles per unit area
σ	-	Standard deviation in Gaussian filters
λ	-	Regularization parameter

CHAPTER 1

INTRODUCTION AND MOTIVATION

The rapid urbanization and increasing number of vehicles on roads have resulted in significant traffic congestion problems in cities worldwide. Urban areas face numerous challenges related to traffic management, including congestion, accidents, pollution, and emergency response delays. Traditional traffic management systems rely on fixed timing signals and manual monitoring, which cannot efficiently adapt to real-time traffic conditions or unexpected situations. This inadequacy calls for innovative approaches that can dynamically respond to changing traffic patterns.

This project presents a Smart Traffic Management System powered by artificial intelligence, computer vision, and web technologies. The system is designed to monitor traffic flow in real-time, detect violations, identify emergency vehicles, and provide comprehensive analytics through an intuitive dashboard. By implementing adaptive traffic signal control, automated violation detection, and emergency vehicle prioritization, the system aims to improve traffic efficiency, enhance road safety, and reduce response times for emergency services.

1.1. BACKGROUND OF TRAFFIC MANAGEMENT

Traffic management has evolved significantly over the past century, transitioning from manual control by traffic officers to sophisticated computerized systems. The evolution can be broadly categorized into four generations:

1.1.1. First generation traffic management

The earliest traffic management systems consisted of manual direction by traffic police and basic mechanical signals. These systems were entirely human-dependent and lacked any form of automation or adaptability.

1.1.2. Second generation traffic management

The introduction of electric traffic signals with pre-programmed timing patterns marked the second generation. These systems operated on fixed-time cycles regardless of actual traffic conditions, offering limited flexibility through time-of-day schedules.

1.1.3. Third generation traffic management

The third generation introduced sensor-based systems using inductive loops, infrared sensors, or radar to detect vehicle presence. These systems could adjust signal timing based on detected vehicles but were limited by their binary detection capabilities (presence or absence) and inability to distinguish vehicle types.

1.1.4. Fourth generation traffic management

Current advanced traffic management systems incorporate multiple data sources, including cameras, GPS data, and IoT devices. These systems offer improved adaptability but still face challenges in real-time processing of complex traffic scenarios and emergency situations.

The Smart Traffic Management System proposed in this project represents the emerging fifth generation, characterized by AI-driven decision-making, computer vision-based comprehensive traffic analysis, and predictive capabilities.

1.2. PROBLEMS WITH TRADITIONAL TRAFFIC SYSTEMS

Traditional traffic management systems face several limitations that affect their efficiency and effectiveness:

1.2.1. Fixed timing signals

Conventional traffic signals operate on pre-programmed timing schedules that don't adapt to actual traffic conditions. This leads to unnecessary delays during off-peak hours and inadequate green time during congestion periods.

1.2.2. Limited detection capabilities

Traditional sensors like inductive loops can only detect the presence of vehicles without distinguishing between vehicle types, occupancy levels, or identifying emergency vehicles. This limitation prevents prioritization based on vehicle importance or urgency.

1.2.3. Inability to detect violations

Most existing systems lack the capability to automatically detect traffic violations such as red-light running, speeding, or wrong-way driving. This limitation reduces compliance and compromises road safety.

1.2.4. Manual monitoring and enforcement

Traffic violations are typically monitored and enforced manually by law enforcement officers, resulting in inconsistent enforcement and resource inefficiency. The manual nature of this process also delays the issuance of citations and reduces their deterrent effect.

1.2.5. Delayed emergency response

Traditional systems lack mechanisms to detect and prioritize emergency vehicles, leading to critical delays in emergency response times. Every minute of delay can significantly impact outcomes in medical emergencies, fire incidents, or criminal activities.

1.2.6. Isolated intersection management

Conventional traffic systems manage intersections in isolation without coordinating signals across a network of intersections. This lack of coordination prevents the creation of "green waves" that could optimize traffic flow along major corridors.

1.2.7. Limited data collection and analytics

Traditional systems collect minimal data and offer limited analytical capabilities, hampering traffic planning and resource allocation decisions. The absence of comprehensive data inhibits the identification of traffic patterns and optimization opportunities.

1.3. NEED FOR AI-BASED SMART TRAFFIC MANAGEMENT

Artificial Intelligence offers significant potential for revolutionizing traffic management through improved detection, analysis, and control capabilities:

1.3.1. Real-time adaptability

AI-based systems can process multiple data inputs simultaneously and adapt traffic signal timing in real-time based on current traffic conditions. This adaptability minimizes waiting times and improves overall traffic flow efficiency.

1.3.2. Comprehensive detection

Computer vision algorithms can detect, classify, and track various road users, including vehicles, pedestrians, and cyclists. This comprehensive detection enables more nuanced traffic management decisions based on the specific mix of road users present.

1.3.3. Automated violation detection

AI systems can automatically identify traffic violations through video analytics, enabling consistent enforcement without human intervention. This capability improves compliance rates and enhances road safety.

1.3.4. Emergency vehicle prioritization

Computer vision can detect emergency vehicles and automatically adjust signal timing to provide them with priority passage. This feature significantly reduces emergency response times and potentially saves lives.

1.3.5. Predictive capabilities

Machine learning models can analyze historical traffic data to predict future congestion patterns and proactively adjust signal timing. This predictive approach prevents congestion rather than merely reacting to it.

1.3.6. Network-wide optimization

AI algorithms can coordinate traffic signals across multiple intersections to optimize network-wide traffic flow rather than focusing on isolated intersections. This coordination creates more efficient traffic patterns throughout the road network.

1.3.7. Data-driven insights

AI systems generate rich datasets that enable detailed traffic analysis and informed decision-making for infrastructure planning. These insights help identify bottlenecks and guide targeted infrastructure investments.

1.4. PROJECT OBJECTIVES

The Smart Traffic Management System aims to address the limitations of traditional traffic management through the following objectives:

1.4.1. Develop a real-time traffic monitoring system

Create a computer vision-based system that can detect, classify, and track vehicles in real-time from video feeds. The system should be capable of processing multiple camera streams simultaneously to monitor different intersections.

1.4.2. Implement adaptive traffic signal control

Design an algorithm that dynamically adjusts traffic signal timing based on real-time traffic density and waiting times. The algorithm should optimize signal timing to minimize overall waiting time and maximize throughput.

1.4.3. Enable emergency vehicle detection and prioritization

Develop a mechanism to detect emergency vehicles (ambulances, fire trucks, police cars) and automatically adjust signal timing to provide them with priority passage. The system should maintain the priority until the emergency vehicle has cleared the intersection.

1.4.4. Create an automated violation detection system

Implement computer vision algorithms to detect traffic violations such as red-light running and generate electronic challans (e-challans) with appropriate evidence. The system should capture vehicle details and violation circumstances for proper documentation.

1.4.5. Design a comprehensive dashboard for traffic analytics

Develop an intuitive web-based dashboard that displays real-time traffic statistics, violation records, signal status, and emergency alerts. The dashboard should provide both real-time monitoring capabilities and historical data analysis.

1.4.6. Integrate a women's safety alert system

Incorporate an audio monitoring system capable of detecting distress calls and generating safety alerts. The system should identify keywords indicating potential emergencies and notify authorities accordingly.

1.4.7. Ensure system reliability and scalability

Design the system architecture to ensure high reliability, fault tolerance, and scalability to accommodate additional cameras and intersections. The system should maintain performance even as the monitoring scope expands.

1.5. SCOPE OF THE PROJECT

The Smart Traffic Management System encompasses the following components and capabilities:

1.5.1. Computer vision module

The system utilizes YOLOv5 object detection model to identify and classify vehicles in video streams. It includes vehicle tracking using the SORT (Simple Online and Realtime Tracking) algorithm to maintain consistent identification across video frames.

1.5.2. Traffic signal control module

An adaptive traffic signal control algorithm adjusts signal timing based on vehicle density, waiting time, and emergency vehicle presence. The algorithm ensures fair distribution of green time while prioritizing high-density approaches.

1.5.3. Violation detection module

The system detects red-light violations by monitoring vehicles crossing stop lines during red signals. It captures violation evidence, generates e-challans, and maintains a violation database for enforcement purposes.

1.5.4. Speed estimation module

A speed estimation algorithm calculates vehicle speeds based on their displacement across frames and flags speeding violations. The estimation takes into account the frame rate and calibrated distance metrics.

1.5.5. Emergency vehicle detection module

The system identifies emergency vehicles through visual characteristics and provides them with signal priority. It maintains the priority until the emergency vehicle clears the intersection.

1.5.6. Women's safety module

An audio monitoring component detects distress calls or keywords indicating safety concerns. Upon detection, it generates alerts and logs the incidents for appropriate response.

1.5.7. Web dashboard

A React-based responsive dashboard displays real-time traffic statistics, violation records, signal status, and safety alerts. The dashboard includes multiple views for different user roles and supports both light and dark themes.

1.5.8. API backend

A FastAPI-based backend handles data processing, storage, and communication between the computer vision modules and the web dashboard. It provides RESTful endpoints for data retrieval and system control.

CHAPTER 2

LITERATURE REVIEW

This chapter presents a comprehensive review of existing literature and technologies in the field of traffic management systems. Understanding the current state of the art and previous approaches provides context for the innovations introduced in this project and highlights the gaps that our Smart Traffic Management System aims to address.

2.1. EXISTING TRAFFIC MANAGEMENT SYSTEMS

Traffic management systems have evolved significantly over the years, incorporating various technologies and approaches to address the growing challenges of urban traffic.

2.1.1. Fixed-time traffic signal systems

Fixed-time signal control systems operate on predetermined timing plans that specify the duration of green, yellow, and red phases for each approach. Webster (1958) developed one of the earliest methodologies for optimizing fixed-time signal plans based on historical traffic volumes. While these systems are simple and reliable, they cannot adapt to unexpected traffic variations or incidents (Koonce et al., 2008).

2.1.2. Actuated traffic signal systems

Actuated control systems use vehicle detectors (typically inductive loops) to adjust signal timing based on real-time vehicle presence. These systems can be fully actuated, where all approaches have detectors, or semi-actuated, where only minor approaches have detectors (Federal Highway Administration, 2008). While more responsive than fixed-time systems, they still have limited adaptability as they only detect vehicle presence, not density or queue length.

2.1.3. Adaptive traffic control systems

Advanced adaptive systems like SCOOT (Split Cycle Offset Optimization Technique) and SCATS (Sydney Coordinated Adaptive Traffic System) continuously adjust signal timing based on real-time traffic data from multiple sensors. Hunt et al. (2008) demonstrated that SCOOT could reduce delays by 20% compared to well-designed fixed-time systems. However, these systems rely primarily on point detectors and lack comprehensive visual analysis capabilities.

2.1.4. Connected vehicle-based systems

Emerging systems utilize vehicle-to-infrastructure (V2I) communication to gather detailed traffic data directly from vehicles. Guler et al. (2014) proposed a connected vehicle-based

signal control approach that outperformed traditional methods in simulation. While promising, these systems require extensive infrastructure investment and widespread adoption of connected vehicle technology.

Table 2.1 provides a comparison of these existing traffic management systems, highlighting their key features, advantages, and limitations.

System Type	Detection Method	Adaptability	Key Advantages	Limitations
Fixed-time	None	Pre-programmed only	Simple, reliable, low maintenance cost	Cannot adapt to traffic fluctuations, inefficient during off-peak hours
Actuated	Inductive loops, radar	Limited (vehicle presence)	Responds to actual vehicle presence, better than fixed-time	Binary detection only, limited intelligence, high installation cost
Adaptive (SCOOT/SCATS)	Multiple sensors, cameras	Moderate (real-time)	Network coordination, historical pattern learning	Limited vision analysis, expensive infrastructure, complex setup
Connected Vehicle	V2I communication	High (vehicle data)	Detailed vehicle information, precise location data	Requires equipped vehicles, limited coverage, privacy concerns
AI-based (Our System)	Computer vision, machine learning	Very high (CV+ML)	Comprehensive detection & analysis, vehicle classification, emergency detection	Computationally intensive, sensitive to weather/lighting conditions

2.2. COMPUTER VISION IN TRAFFIC MANAGEMENT

Computer vision technologies have been increasingly applied to traffic management, offering capabilities beyond traditional sensor-based approaches.

2.2.1. Vehicle detection and classification

Early computer vision approaches for vehicle detection relied on background subtraction techniques (Stauffer & Grimson, 1999) and feature-based classifiers (Viola & Jones, 2001). Recent advances in deep learning have significantly improved detection accuracy and robustness. Redmon et al. (2016) introduced YOLO (You Only Look Once), a real-time object detection system that has been widely adapted for traffic applications. Wang et al. (2019) demonstrated YOLOv3's effectiveness for traffic monitoring with accuracy exceeding 95% for vehicle detection.

2.2.2. Vehicle tracking algorithms

Multi-object tracking algorithms maintain consistent identities of vehicles across video frames. Bewley et al. (2016) proposed SORT (Simple Online and Realtime Tracking), which combines Kalman filtering with Hungarian method assignment to achieve efficient tracking with minimal computational overhead. DeepSORT (Wojke et al., 2017) extended this approach by incorporating appearance features, further improving tracking robustness in congested scenes.

2.2.3. Traffic violation detection

Computer vision techniques have been applied to detect various traffic violations. Bouttefroy et al. (2008) developed a red-light violation detection system using virtual loop detectors and background subtraction. More recently, Zhu et al. (2018) proposed a deep learning approach for detecting multiple violation types, including red-light violations, illegal lane changes, and wrong-way driving, achieving over 90% detection accuracy.

2.2.4. Speed estimation

Vision-based speed estimation techniques calculate vehicle speeds from video footage. Famouri et al. (2019) presented a camera calibration method for accurate speed measurement using a single camera. Sochor et al. (2018) developed a deep learning approach that combined 3D bounding box estimation with tracking to achieve speed estimation accuracy within 3 km/h of radar measurements.

2.2.5. License plate recognition

Automatic License Plate Recognition (ALPR) systems extract and recognize vehicle license plates from images. Silva & Jung (2018) proposed a deep learning pipeline for ALPR that achieved over 96% end-to-end accuracy across various environmental conditions. These systems are crucial for violation enforcement and vehicle identification.

Table 2.2 summarizes the computer vision techniques applied to traffic analysis, including their methodologies, performance metrics, and practical applications.

Technique	Methodology	Accuracy/Performance	Application Area
Object Detection	YOLO, SSD, Faster R-CNN	85-95% mAP	Vehicle counting, classification, density estimation
Vehicle Tracking	SORT, DeepSORT, IoU Tracker	80-90% MOTA	Trajectory analysis, speed estimation, movement patterns
License Plate Recognition	CNN + OCR	75-95% accuracy	Vehicle identification, violation enforcement, access control
Traffic Density Estimation	CNN classification, pixel occupancy	80-95% accuracy	Congestion analysis, signal timing optimization, traffic flow management
Violation Detection	Rule-based analytics with CV	70-85% detection rate	Red light violations, speed violations, illegal lane changes

2.3. AI AND ML APPLICATIONS IN TRAFFIC CONTROL

Artificial intelligence and machine learning have transformed traffic control by enabling data-driven decision-making and predictive capabilities.

2.3.1. Reinforcement learning for signal control

Reinforcement learning (RL) approaches model traffic signal control as a decision-making problem where the agent (controller) learns optimal actions through interaction with the environment. Genders & Razavi (2016) demonstrated that deep RL could outperform conventional adaptive systems in simulation. Wei et al. (2019) proposed a multi-intersection RL approach that reduced average delay by 20% compared to actuated control in a real-world implementation.

2.3.2. Traffic prediction models

Predictive models forecast future traffic conditions based on historical patterns and real-time data. Yu et al. (2017) developed a deep learning approach using Long Short-Term Memory (LSTM) networks that predicted traffic flow with mean absolute percentage error below 5%. These predictions enable proactive signal adjustments to prevent congestion before it occurs.

2.3.3. Computer vision for traffic density estimation

Deep learning models have been developed to estimate traffic density directly from camera footage. Zhang et al. (2017) proposed a convolutional neural network approach that classified congestion levels with over 95% accuracy. This capability is essential for density-based signal control without requiring explicit vehicle counting.

2.3.4. Multi-modal traffic management

AI techniques have been applied to manage diverse road users, including vehicles, pedestrians, and cyclists. Zaki et al. (2020) developed a computer vision system that detected and tracked multiple road user types to enable more equitable signal control. This approach is particularly valuable in urban areas with mixed traffic.

2.3.5. Transfer learning for detection models

Transfer learning techniques adapt pre-trained models to specific traffic scenarios with limited labeled data. Peppa et al. (2018) demonstrated that fine-tuning YOLOv3 with just 500 domain-specific images improved detection accuracy by 15% compared to the base model. This approach reduces the data requirements for deploying AI systems in new environments.

2.4. EMERGENCY RESPONSE SYSTEMS

Emergency vehicle priority systems aim to reduce response times by providing preferential treatment at traffic signals.

2.4.1. Traditional preemption systems

Conventional emergency vehicle preemption systems use specialized hardware such as infrared emitters, acoustic sensors, or GPS transmitters. Bullock et al. (1999) evaluated infrared-based preemption and found it reduced response times by an average of 14-23%. However, these systems require dedicated equipment on both vehicles and intersections.

2.4.2. Computer vision-based detection

Vision-based approaches identify emergency vehicles through their visual characteristics without requiring specialized vehicle equipment. Rachakonda et al. (2020) developed a CNN-based system that detected emergency vehicles with 98% accuracy based on their appearance and light patterns. This approach offers a more cost-effective and flexible alternative to hardware-based preemption.

2.4.3. Multi-modal emergency detection

Some systems combine multiple detection methods for improved reliability. Liu et al. (2019) proposed an approach that integrated visual detection with siren audio recognition, achieving 99.3% detection accuracy even in adverse conditions. This redundancy ensures reliable emergency vehicle detection across various scenarios.

2.4.4. Adaptive preemption strategies

Advanced preemption strategies consider factors beyond simple signal overrides. Cesme & Furth (2014) developed a conditional priority strategy that balanced emergency vehicle benefits with overall network disruption. Their approach reduced emergency response times by 15% while limiting the increase in overall network delay to just 3%.

2.4.5. Preemption in coordinated networks

Providing emergency preemption in coordinated signal networks presents unique challenges. He et al. (2017) proposed a transition strategy that maintained coordination while accommodating emergency vehicles, reducing recovery time by 40% compared to conventional preemption. This approach minimizes the disruptive effects of preemption on the broader traffic network.

2.5. SMART CITY TRAFFIC SOLUTIONS

Smart city initiatives integrate traffic management with broader urban systems to create comprehensive solutions.

2.5.1. Integrated traffic management platforms

Comprehensive platforms combine multiple traffic management functions within unified systems. Barba et al. (2012) described Madrid's integrated traffic management system, which coordinates traffic signals, incident detection, and public information. This integration enables coordinated responses to traffic situations and improved decision-making.

2.5.2. IoT-based traffic monitoring

Internet of Things (IoT) approaches use distributed sensors to collect traffic data across urban areas. Mehmood et al. (2017) presented a system using roadside IoT devices that gathered traffic data with over 95% accuracy at significantly lower cost than traditional monitoring infrastructure. These distributed sensing networks provide more comprehensive coverage than camera-only approaches.

2.5.3. Edge computing for traffic systems

Edge computing architectures process data near its source to reduce latency and bandwidth requirements. Zhou et al. (2020) demonstrated an edge-based traffic monitoring system that reduced detection latency by 65% compared to cloud-based processing. This approach enables real-time response to traffic events even with limited connectivity.

2.5.4. Public safety integration

Advanced systems integrate traffic management with public safety functions. Djahel et al. (2015) proposed a framework that coordinated traffic signals, emergency dispatch, and

public alerts during incidents. This integration reduced emergency response times by up to 30% in simulated scenarios by creating coordinated responses across multiple systems.

2.5.5. Data-driven urban planning

Traffic data from smart systems informs long-term urban planning decisions. Zheng et al. (2016) described Beijing's use of traffic data to identify infrastructure bottlenecks and optimize road network design. This application demonstrates how smart traffic systems contribute value beyond day-to-day operations by supporting strategic planning.

The literature review reveals that while significant advances have been made in traffic management systems, opportunities remain to integrate multiple capabilities—adaptive signal control, violation detection, emergency vehicle prioritization, and safety monitoring—into a unified system with a comprehensive dashboard. The Smart Traffic Management System developed in this project addresses this gap by combining these functionalities within a coherent architecture.

CHAPTER 3

PROJECT METHODOLOGY AND DESIGN

This chapter details the methodology and design approach adopted for the Smart Traffic Management System. It provides a comprehensive overview of the system architecture, component design, algorithms, and integration strategies that form the foundation of the project implementation.

3.1. SYSTEM ARCHITECTURE

The Smart Traffic Management System follows a modular, microservices-oriented architecture to ensure scalability, maintainability, and fault tolerance. The architecture consists of several interconnected components that work together to provide a comprehensive traffic management solution.

3.1.1. Architectural overview

The system is structured into four primary layers:

1. **Data Acquisition Layer**: Handles video input from traffic cameras and audio input from microphones.
2. **Processing Layer**: Processes the acquired data using computer vision and audio analysis algorithms.
3. **Business Logic Layer**: Implements traffic signal control, violation detection, and emergency response logic.
4. **Presentation Layer**: Provides user interfaces for monitoring and administration.

3.1.2. Component interaction

The components interact through well-defined interfaces, primarily using RESTful APIs for synchronous communication. The core components include:

1. **Camera Processors**: Handle video streams from multiple cameras, performing object detection, tracking, and violation detection.
2. **Audio Monitor**: Analyzes audio inputs to detect distress signals for the women's safety module.
3. **Traffic Signal Controller**: Implements the adaptive signal control algorithm based on processed traffic data.
4. **Violation Manager**: Records and manages traffic violations and generates e-challans.

5. **Stats Manager**: Aggregates and maintains system-wide statistics for reporting.
6. **API Server**: Provides endpoints for the web dashboard to access system data and functionality.
7. **Web Dashboard**: Offers an intuitive interface for monitoring traffic conditions, violations, and system status.

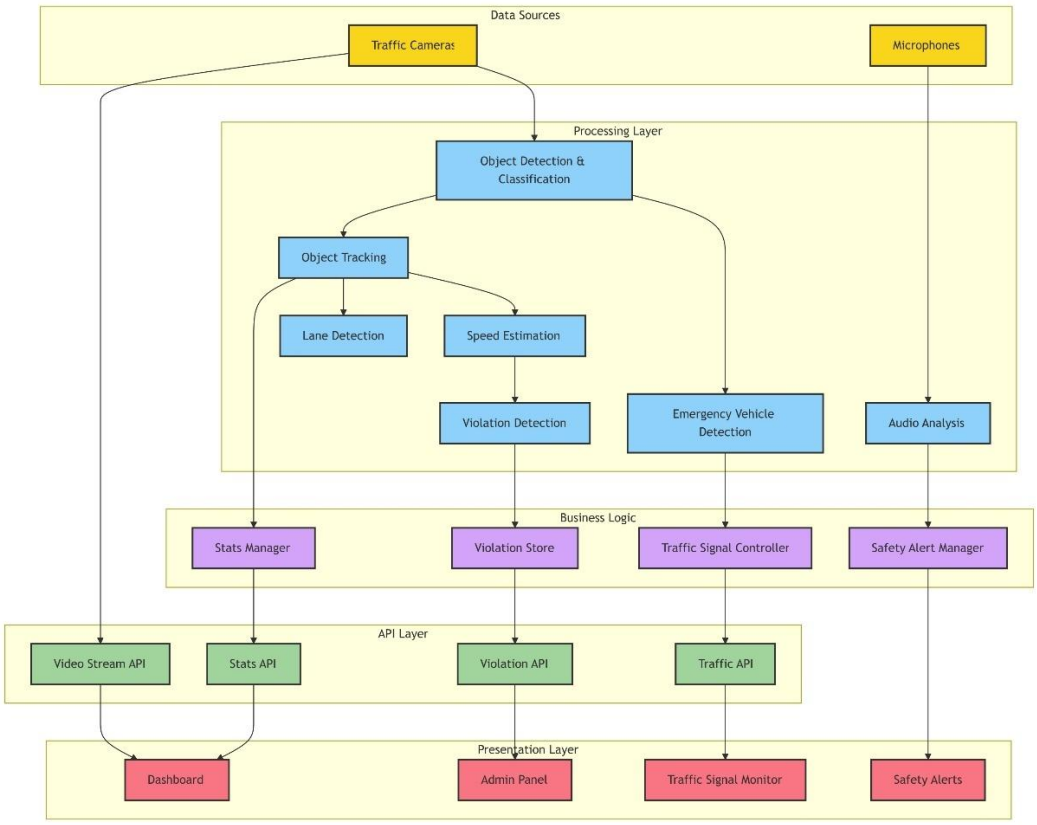

Figure 3.1 illustrates the overall system architecture, showing the interactions between different components and data flows.

3.1.3. Data flow

The data flow within the system follows a pipeline pattern:

1. Raw video and audio data are captured from input sources.
2. The processing layer applies computer vision and audio analysis algorithms to extract relevant information.
3. The business logic layer uses this information to make decisions about traffic signal control, violation detection, and safety alerts.

4. Results are stored in the system's data stores and made available to the presentation layer.
5. The dashboard accesses this information through the API server and presents it to users in an intuitive format.

3.1.4. Deployment architecture

The system is designed for deployment in a distributed environment, with components running as separate services that can be scaled independently based on computational requirements. This approach allows for flexible resource allocation, with compute-intensive components like video processing potentially deployed on GPU-enabled servers while lighter components run on standard hardware.

3.2. FRONTEND DESIGN

The frontend of the Smart Traffic Management System is built using React.js, a popular JavaScript library for building user interfaces. The design follows modern web development practices, emphasizing responsiveness, accessibility, and user experience.

3.2.1. Component hierarchy

The frontend follows a hierarchical component structure that promotes reusability and maintainability. Figure 3.2 below illustrates the component hierarchy, showing how components are organized and nested within the application.

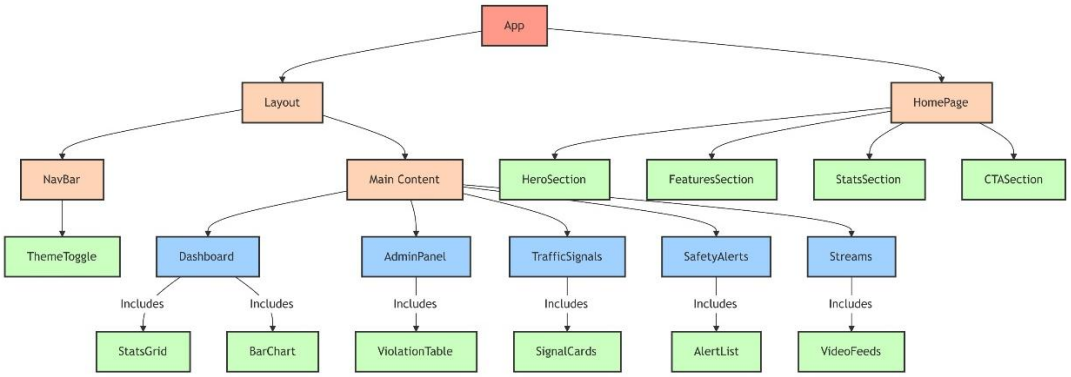

Figure 3.2. Frontend Component Structure.

The major components include:

1. **App**: The root component that manages routing and global state.
2. **Layout**: Provides the common layout structure for authenticated pages, including navigation and emergency alerts.
3. **Dashboard**: Displays traffic statistics and key performance indicators.

4. **AdminPanel**: Manages violation records and e-challan generation.
5. **TrafficSignals**: Shows the status of traffic signals across the monitored intersections.
6. **SafetyAlerts**: Displays women's safety alerts detected by the system.
7. **Streams**: Provides live video feeds from traffic cameras.
8. **HomePage**: Offers an introduction to the system with key features and statistics.

3.2.2. Component design

Each component is designed with specific responsibilities and follows the principle of separation of concerns. Table 3.1 details the frontend component structure, listing key components, their purposes, and their major functionalities.

Table 3.1. Frontend Component Structure.

Component	**Purpose**	**Key Functionality**
App	Root component	Routing, theme initialization, global state management
Layout	Page structure	Navigation, emergency alerts, common layout wrapping
Dashboard	Traffic overview	Stats display, data visualization, real-time metrics
AdminPanel	Violation management	Violation table, e-challan download, record management
TrafficSignals	Signal monitoring	Real-time signal status display, intersection monitoring
SafetyAlerts	Safety monitoring	Alert display and history, distress call notifications
Streams	Video monitoring	Live camera feeds display, multi-camera viewing
ThemeToggle	UI customization	Light/dark theme switching, persistent preferences
HomePage	System introduction	Features showcase, stats animation, system marketing
NavBar	Navigation	Consistent navigation across application pages

The design employs several UI patterns:

1. **Card Pattern**: For displaying discrete chunks of information with clear visual boundaries.
2. **Dashboard Pattern**: For organizing multiple metrics and visualizations in a scannable layout.
3. **Navigation Pattern**: For enabling easy movement between different sections of the application.
4. **Form Pattern**: For collecting user input in administrative interfaces.

3.2.3. State management

The application state is managed using React's built-in state management capabilities, including:

1. **Component State**: For local component-specific state using useState hooks.
2. **Effect Management**: For handling side effects like data fetching using useEffect hooks.
3. **Context API**: For sharing state that needs to be accessed by multiple components without prop drilling.

This approach provides a good balance between simplicity and functionality for an application of this scale.

3.2.4. Responsive design

The frontend implements a responsive design that adapts to different screen sizes, from desktop monitors to mobile devices. This is achieved through:

1. **Fluid Grid Layouts**: Using CSS Grid and Flexbox for flexible component arrangements.
2. **Media Queries**: Adjusting layouts and component sizes based on viewport dimensions.
3. **Relative Units**: Using em, rem, and percentage units for scalable typography and spacing.

3.2.5. Theme support

The system supports both light and dark themes to accommodate user preferences and different viewing conditions. The theming system uses CSS variables for color definitions, allowing for runtime theme switching without page reloads. The ThemeToggle component enables users to switch between themes, with preferences saved to localStorage for persistence across sessions.

3.2.6. Data visualization

The dashboard includes several data visualization components built using the Recharts library, including:

1. **Bar Charts**: For comparing metrics like vehicle counts and violations.
2. **Line Charts**: For showing trends over time.
3. **Status Indicators**: For displaying traffic signal states using color-coded visual elements.

These visualizations are designed to be intuitive and to convey information at a glance while providing detailed information on hover or interaction.

3.3. BACKEND DESIGN

The backend of the Smart Traffic Management System is built using FastAPI, a modern, high-performance web framework for building APIs with Python. The backend design focuses on performance, scalability, and maintainability.

3.3.1. API design

The backend exposes a RESTful API that follows standard HTTP methods and status codes. The API is designed around resource-oriented endpoints that provide clear and intuitive access to system functionality. Figure 3.3 illustrates the backend service architecture, showing the main API endpoints and their relationships to backend services.

Table 3.2. Backend API Endpoints

Endpoint	Method	Purpose	Response Format	Parameters
/api/stats	GET	Retrieve traffic statistics	JSON	None
/api/challans	GET	Get list of violation records	JSON Array	None
/api/challans/download	GET	Download e-challan PDF	File (PDF)	pdf_path (query)
/api/emergency-status	GET	Check active emergency overrides	JSON	None
/api/safety-alerts	GET	Retrieve safety alert history	JSON Array	None
/traffic/vehicle-counts	GET	Get vehicle counts by camera	JSON	None

/traffic/lights	GET	Get traffic signal states	JSON	None
/video_feed/{cam_id}	GET	Stream video from specific camera	Multipart Stream	cam_id (path)

3.3.2. Service modules

The backend is organized into several service modules, each responsible for specific functionality:

1. **Traffic API**: Manages traffic-related endpoints, including vehicle counts and traffic signal states.
2. **Video Stream**: Handles video streaming from traffic cameras to the dashboard.
3. **Violation Store**: Manages traffic violation records and e-challan generation.
4. **Stats Manager**: Maintains system-wide statistics for reporting.
5. **Safety Alert API**: Manages women's safety alerts detected by the audio monitoring system.
6. **Emergency Detector**: Processes emergency vehicle detection and signal override requests.

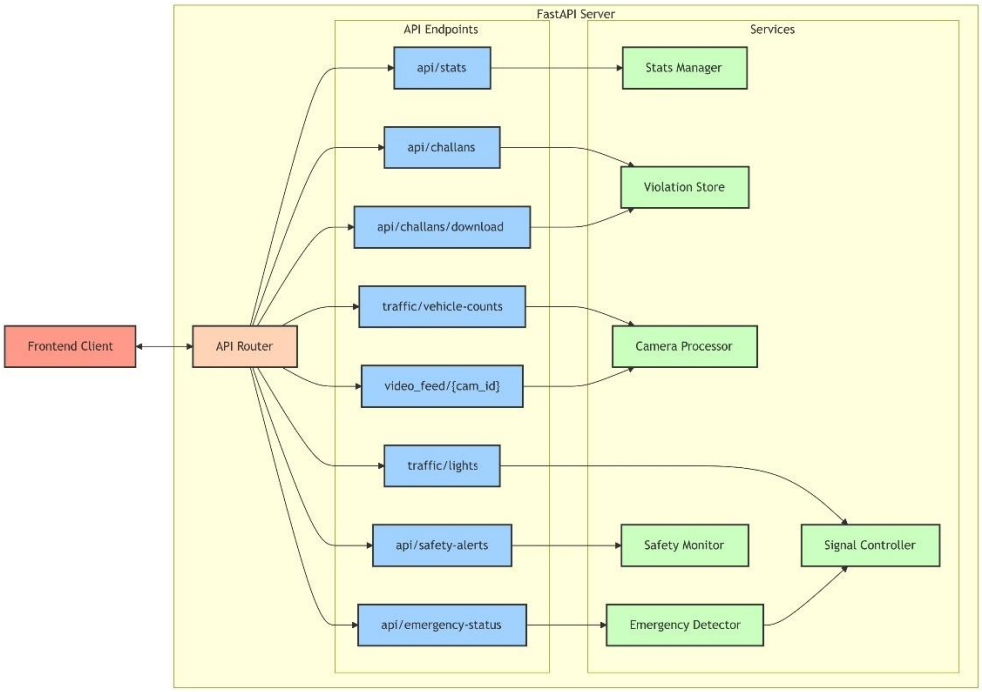

Figure 3.3. Backend Service Architecture.

Each service module is designed as a loosely coupled component that can be developed, tested, and deployed independently.

3.3.3. Concurrency model

The backend employs an asynchronous concurrency model using Python's async/await syntax, which is well-supported by FastAPI. This approach allows the system to handle multiple concurrent requests efficiently without blocking on I/O operations.

For computationally intensive tasks like video processing, the system uses a threading model that runs these tasks in separate threads to prevent them from blocking the main request-handling thread.

3.3.4. Error handling

The backend implements comprehensive error handling to ensure robustness and provide meaningful feedback to clients. This includes:

1. **Exception Handlers**: Custom exception handlers that translate Python exceptions into appropriate HTTP responses.
2. **Validation**: Request validation using Pydantic models to ensure data integrity.
3. **Logging**: Detailed logging of errors and exceptional conditions for troubleshooting.

3.3.5. Cross-Origin Resource Sharing (CORS)

The backend implements CORS middleware to allow controlled access from the frontend application, which may be served from a different origin. This is configured to allow specific origins, methods, and headers while maintaining security.

3.3.6. Video streaming

The video streaming implementation uses multipart/x-mixed-replace responses to provide a continuous stream of JPEG frames to the client. This approach is compatible with most modern browsers and does not require special plugins or technologies beyond standard HTTP.

3.4. COMPUTER VISION PIPELINE

The computer vision pipeline is a critical component of the Smart Traffic Management System, responsible for extracting meaningful information from traffic camera footage. The pipeline consists of several stages that transform raw video frames into actionable intelligence.

3.4.1. Pipeline overview

The computer vision pipeline follows a sequential processing model, as illustrated in

Figure 3.4. The main stages include:

1. **Frame Acquisition**: Capturing frames from video sources.
2. **Preprocessing**: Enhancing image quality and preparing frames for analysis.
3. **Object Detection**: Identifying and classifying vehicles and other objects of interest.
4. **Object Tracking**: Maintaining consistent object identities across frames.
5. **Lane Detection**: Identifying lane boundaries and assigning vehicles to lanes.
6. **Speed Estimation**: Calculating vehicle speeds based on displacement across frames.
7. **Violation Detection**: Identifying traffic violations based on object behavior.
8. **Emergency Vehicle Recognition**: Detecting emergency vehicles for signal prioritization.
9. **Result Annotation**: Adding visual indicators to frames for monitoring purposes.

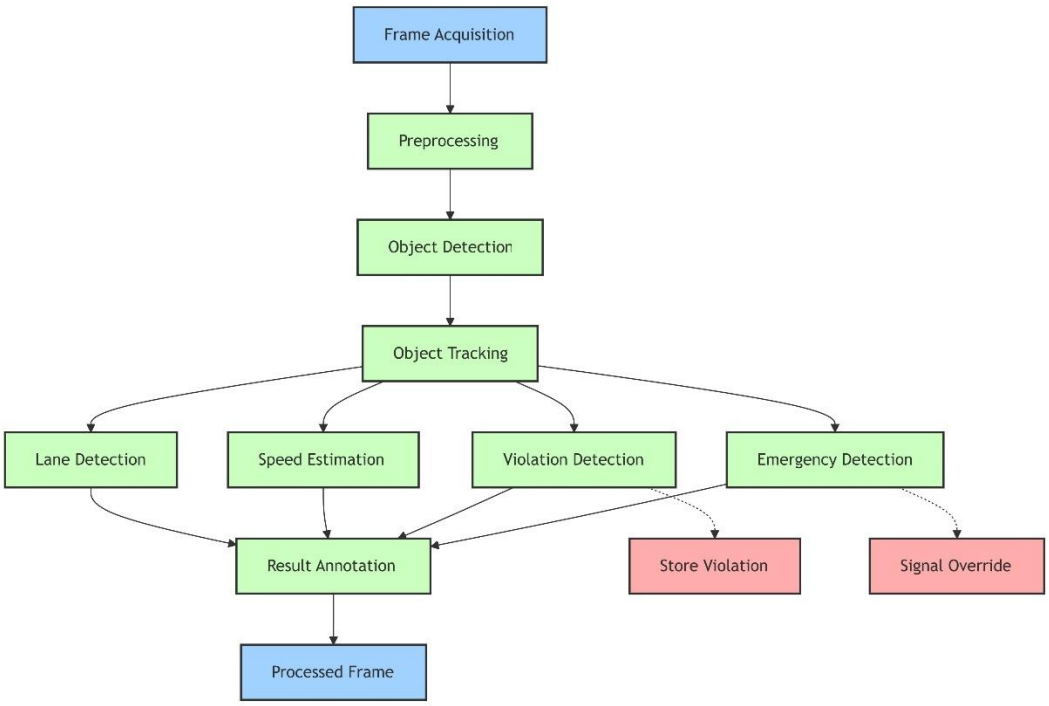

Figure 3.4. Computer Vision Pipeline Workflow.

3.4.2. Object detection

The system uses YOLOv5, a state-of-the-art object detection model, to identify vehicles and other objects in video frames. Figure 3.5 illustrates the object detection and classification process.

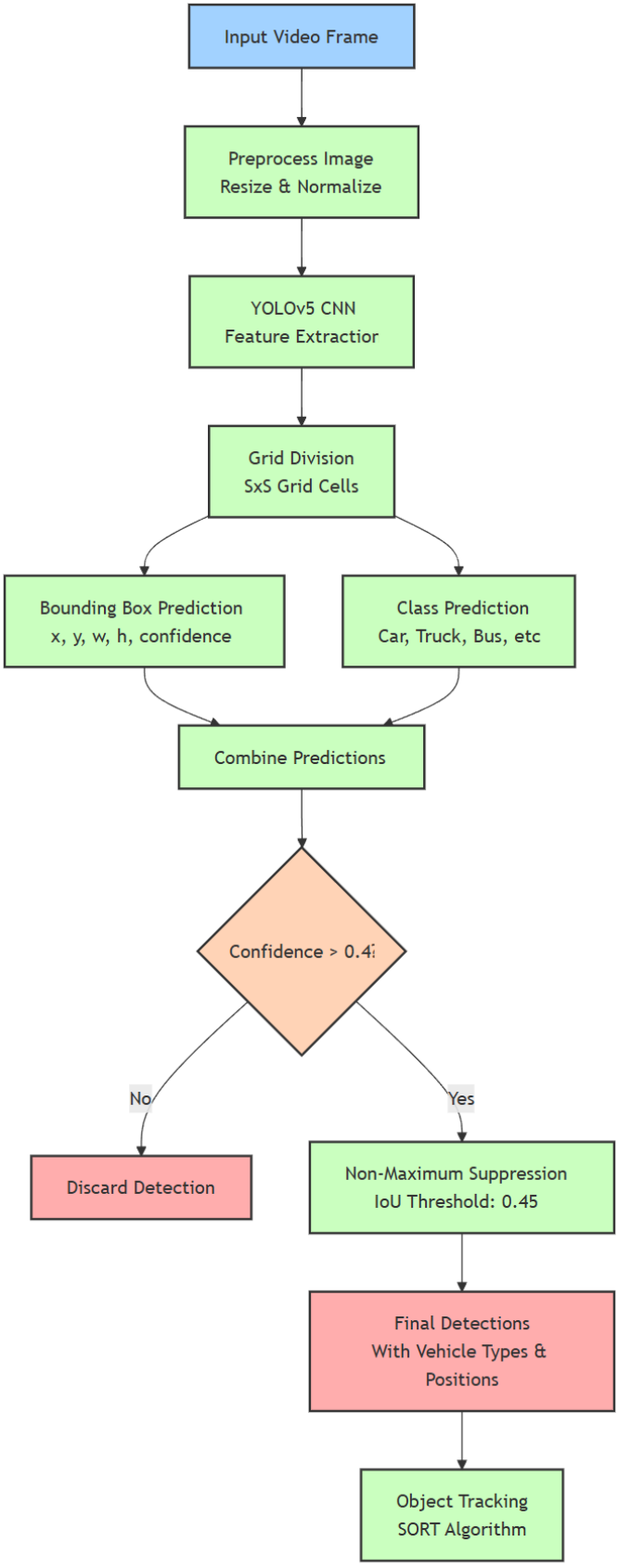

Figure 3.5. Object Detection and Classification Process.

The implementation includes:

1. **Model Loading**: The YOLOv5 model is loaded once during system initialization to avoid repeated loading overhead.
2. **Confidence Thresholding**: A confidence threshold of 0.4 is applied to filter out low-confidence detections.
3. **Non-Maximum Suppression**: NMS with an IoU threshold of 0.45 is applied to eliminate duplicate detections.
4. **Class Filtering**: The system focuses on vehicle classes (car, truck, bus, motorcycle) while filtering out irrelevant detections.

3.4.3. Object tracking

The SORT (Simple Online and Realtime Tracking) algorithm is used to maintain consistent object identities across video frames. This approach combines Kalman filtering for motion prediction with the Hungarian algorithm for detection association.

The tracking implementation includes:

1. **State Representation**: Each tracked object is represented by its bounding box coordinates and velocity.
2. **Prediction Step**: The Kalman filter predicts the next state of each tracked object based on its current state and motion model.
3. **Update Step**: New detections are associated with existing tracks based on IoU overlap.
4. **Track Management**: New tracks are created for unmatched detections, and tracks without recent updates are terminated.

3.4.4. Lane detection

The lane detection module identifies lane boundaries and assigns vehicles to specific lanes. This information is used for lane-specific traffic management and violation detection.

The implementation includes:

1. **Edge Detection**: Canny edge detection is applied to identify potential lane markings.
2. **Hough Transform**: The Hough transform is used to detect straight lines from the edge image.
3. **Line Filtering**: Detected lines are filtered based on slope and position to identify lane boundaries.
4. **Lane Assignment**: Vehicles are assigned to lanes based on their center position relative to detected lane boundaries.

3.4.5. Speed estimation

The speed estimation module calculates vehicle speeds based on their displacement across frames, the frame rate, and a pixel-to-meter conversion factor.

The implementation includes:

1. **Position Tracking**: The center positions of vehicles are tracked across frames.
2. **Distance Calculation**: The Euclidean distance between positions in consecutive frames is calculated.
3. **Speed Conversion**: The pixel distance is converted to meters using a calibration factor, and then to km/h based on the frame rate.
4. **Temporal Smoothing**: A moving average filter is applied to reduce noise in speed estimates.

3.4.6. Violation detection

The violation detection module identifies traffic violations based on object behavior and traffic signal states. The system currently focuses on red-light violations but is designed to be extensible to other violation types.

The implementation includes:

1. **Stop Line Definition**: A virtual stop line is defined at a specific position in the frame.
2. **Signal State Monitoring**: The system tracks the current state of the traffic signal for each approach.
3. **Crossing Detection**: The system detects when a vehicle crosses the stop line during a red signal.
4. **Violation Recording**: When a violation is detected, the system captures relevant information, including vehicle ID, timestamp, speed, and a snapshot for evidence.

3.5. TRAFFIC SIGNAL CONTROL ALGORITHM

The traffic signal control algorithm is responsible for dynamically adjusting traffic signal timing based on real-time traffic conditions, waiting times, and special circumstances like emergency vehicle presence.

3.5.1. Algorithm overview

The system implements an adaptive traffic signal control algorithm that combines traffic density information with fairness considerations and emergency overrides.

Table 3.3. Traffic Signal States and Conditions

Signal State	Transition To	Condition	Duration
RED	GREEN	Highest vehicle count AND no emergency OR Waiting limit reached OR Emergency vehicle detected on approach	Variable
GREEN	YELLOW	Green time expired (40-90 seconds) OR Emergency on different approach	After serving current demand
YELLOW	RED	Yellow time expired	Fixed (3 seconds)
RED	RED	Not selected for green in current cycle	Until selected
GREEN (Emergency)	YELLOW	Emergency vehicle clears intersection OR maximum emergency time (20s) elapsed	Variable

3.5.2. Normal operation mode

During normal operation, the algorithm:

1. Monitors vehicle counts at each approach to determine traffic density.
2. Assigns green time to the approach with the highest vehicle count if traffic is above a threshold.
3. Cycles through approaches in a round-robin fashion if traffic is below the threshold.
4. Maintains a fairness mechanism to prevent starvation of low-traffic approaches.

The algorithm uses a waiting limit parameter (currently set to 3 cycles) to ensure that no approach waits excessively long for a green signal. If an approach reaches this limit, it receives priority regardless of its traffic density.

3.5.3. Emergency override mode

When an emergency vehicle is detected, the algorithm enters an override mode:

1. The approach with the emergency vehicle is immediately given a green signal.
2. All other approaches are set to red.
3. The green signal is maintained for a fixed duration (currently 20 seconds) or until the emergency vehicle has cleared the intersection.
4. After the emergency vehicle has passed, the system returns to normal operation.

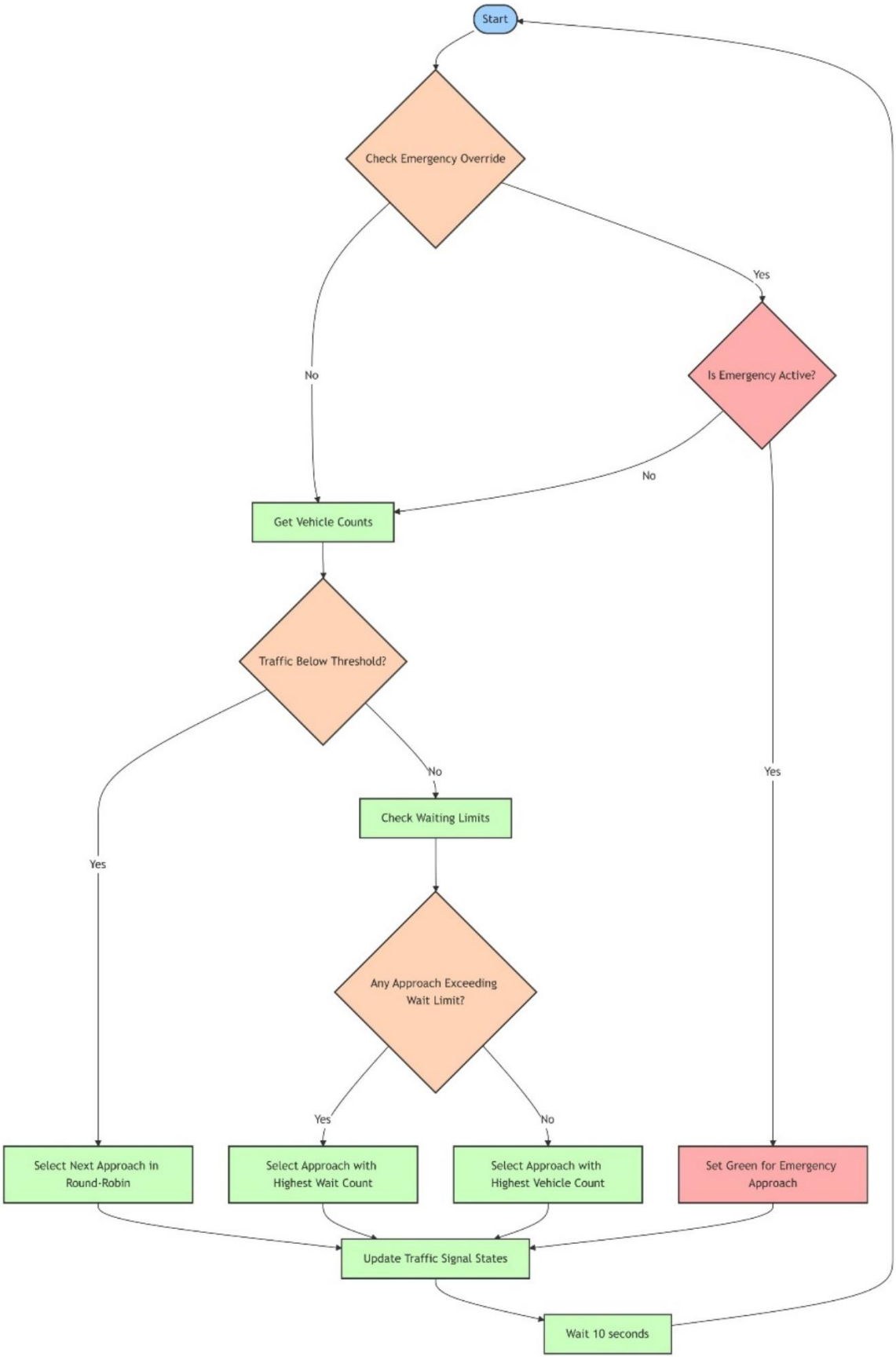

Figure 3.6 illustrates the algorithm's flow in a detailed flowchart.

3.5.4. Coordination mechanism

For networks of intersections, the algorithm includes a basic coordination mechanism:

1. Downstream intersections are notified when a large platoon of vehicles receives a green signal at an upstream intersection.
2. The downstream intersection adjusts its timing to provide a green wave for the approaching platoon.
3. This coordination is prioritized based on the size of the approaching platoon and the current traffic conditions at the downstream intersection.

3.6. EMERGENCY VEHICLE DETECTION SYSTEM

The emergency vehicle detection system is designed to identify emergency vehicles in traffic streams and trigger appropriate signal prioritization to reduce response times.

3.6.1. Detection approach

The system uses a computer vision-based approach to detect emergency vehicles based on their visual characteristics. Figure 3.7 illustrates the emergency vehicle detection process.

The implementation includes:

1. **Class-Based Detection**: The YOLOv5 model is configured to recognize specific emergency vehicle classes, including ambulances, fire trucks, and police cars.
2. **Confidence Thresholding**: A higher confidence threshold (0.4) is applied for emergency vehicle detection to minimize false positives.
3. **Priority Verification**: When a potential emergency vehicle is detected, additional verification is performed by checking for distinctive features like emergency lights.

3.6.2. Signal override mechanism

When an emergency vehicle is confirmed, the system initiates a signal override sequence:

1. The traffic signal controller is notified of the emergency vehicle's presence and location.
2. The controller determines the approach where the emergency vehicle is detected.
3. All traffic signals are set to red except for the approach with the emergency vehicle, which receives a green signal.
4. The override remains active for a predetermined duration (20 seconds) or until the emergency vehicle clears the intersection.

5. A countdown timer tracks the remaining override time, which is displayed on the dashboard.

3.6.3. Multi-camera coordination

For emergency vehicles moving through multiple intersections, the system implements a coordination mechanism:

1. When an emergency vehicle exits one intersection, the system predicts its path based on direction and speed.
2. The predicted path is used to preemptively notify downstream intersections.
3. Downstream intersections prepare for the approaching emergency vehicle by adjusting their signal timing in advance.

This approach creates a "green corridor" that allows emergency vehicles to move through multiple intersections with minimal delay.

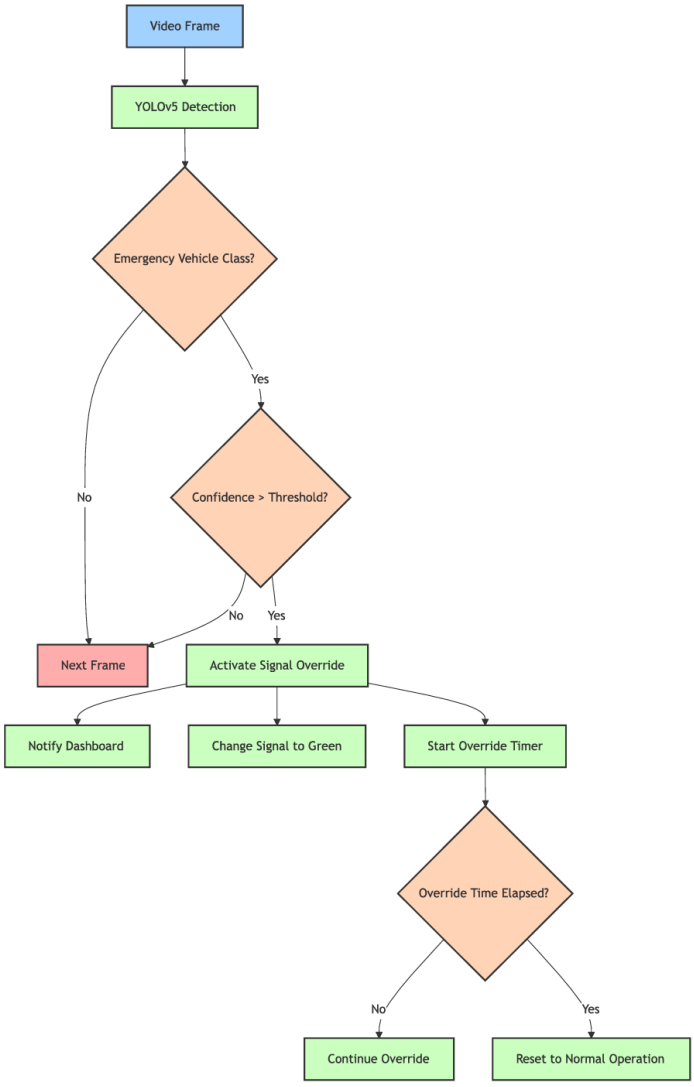

Figure 3.7. Emergency Vehicle Detection Process.

3.7. WOMEN'S SAFETY ALERT SYSTEM

The women's safety alert system is designed to detect distress calls or safety concerns through audio monitoring and generate appropriate alerts.

3.7.1. System architecture

The safety alert system consists of audio monitoring, keyword detection, and alert generation components, as illustrated in Figure 3.8.

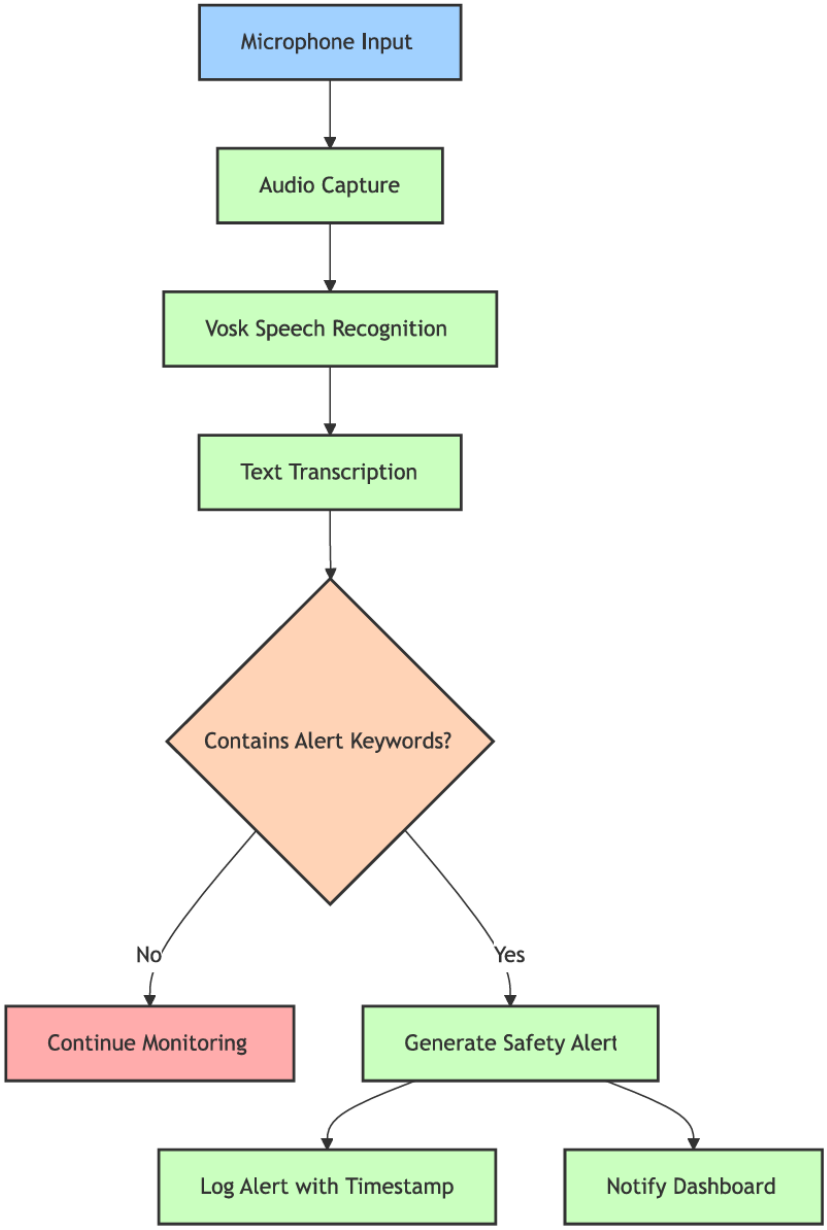

Figure 3.8. Safety Alert System Architecture.

3.7.2. Audio processing pipeline

The audio processing pipeline includes:

1. **Audio Capture**: Continuous audio monitoring through microphones placed at strategic locations.
2. **Speech Recognition**: The Vosk speech recognition model converts captured audio to text.
3. **Keyword Detection**: The system analyzes the transcribed text for distress keywords like "help," "save me," "leave me," "don't touch," and "bachao."
4. **Alert Generation**: When distress keywords are detected, the system generates a safety alert with timestamp and the detected phrase.

3.7.3. Alert management

The system maintains a log of detected safety alerts, which are:

1. Displayed on the dashboard in the Safety Alerts section.
2. Stored in a database for historical reference and pattern analysis.
3. Optionally integrated with external notification systems to alert authorities.

3.7.4. Privacy considerations

To address privacy concerns, the system:

1. Does not store continuous audio recordings, only processing audio in real-time.
2. Only logs the specific distress phrases detected, not full conversation transcripts.
3. Implements audio processing at the edge, minimizing data transmission.

3.8. DATABASE DESIGN

While the current implementation uses in-memory data structures for simplicity and development speed, the system's design includes provisions for a more robust database architecture for production deployment.

3.8.1. Data models

The system's data models include:

1. **Vehicle Detections**: Records of vehicles detected by the system, including timestamp, location, type, and tracked ID.
2. **Traffic Violations**: Records of detected violations, including vehicle information, violation type, evidence (snapshot), and timestamp.

3. **Traffic Signals**: The current and historical states of traffic signals at each monitored intersection.
4. **Safety Alerts**: Records of detected safety concerns, including timestamp, location, and detected phrase.
5. **System Statistics**: Aggregated statistics about vehicle counts, violation rates, and system performance.

3.8.2. Storage strategy

The design includes a hybrid storage strategy:

1. **Operational Data**: Recent data needed for real-time operations is stored in-memory for fast access.
2. **Historical Data**: Older data is moved to persistent storage for long-term retention and analysis.
3. **Media Storage**: Image and video evidence is stored in a file system with database references.

3.8.3. Data retention policy

The system's design includes a data retention policy that balances operational needs with storage efficiency:

1. **Short-term Data**: Detailed operational data is retained for 30 days.
2. **Violation Records**: Violation evidence is retained for 1 year.
3. **Aggregated Statistics**: Historical statistics are retained indefinitely for trend analysis.

The methodology and design described in this chapter provide a comprehensive framework for the implementation of the Smart Traffic Management System. The modular architecture, well-defined algorithms, and integration strategies ensure that the system can effectively address the traffic management challenges outlined in the introduction while providing a foundation for future enhancements.

CHAPTER 4

IMPLEMENTATION AND RESULTS

This chapter details the implementation of the Smart Traffic Management System and presents the results obtained from testing and deployment. It covers the development environment, implementation details for both frontend and backend components, integration process, testing methodology, performance analysis, and system demonstration.

4.1. DEVELOPMENT ENVIRONMENT

The development of the Smart Traffic Management System utilized a combination of tools, frameworks, and libraries to implement the design described in the previous chapter.

Table 4.1. Development Tools and Technologies

Category	Tool/Technology	Purpose	Version
Frontend Framework	React.js	Building the user interface	18.2.0
State Management	React Hooks	Component state and effects	Built-in
Routing	React Router	Navigation between pages	6.4.3
UI Components	Custom CSS	Styling and theming	CSS3
Data Visualization	Recharts	Dashboard charts and analytics	2.1.16
Backend Framework	FastAPI	API server and backend services	0.88.0
Python Runtime	Python	Backend language	3.8+
Computer Vision	YOLOv5	Object detection	Latest
Neural Network	PyTorch	Deep learning framework	1.12.0
Object Tracking	SORT algorithm	Vehicle tracking across frames	Latest
Image Processing	OpenCV	Frame manipulation and analysis	4.6.0
Speech Recognition	Vosk	Audio processing for safety alerts	0.3.42
PDF Generation	ReportLab	E-challan document creation	3.6.12
Development Server	Uvicorn	ASGI server for FastAPI	0.18.3
Version Control	Git	Source code management	Latest

4.1.1. Software tools and frameworks

The primary software tools and frameworks used in the development include:

1. **Frontend Development**:
    - React.js for building the user interface
    - Recharts for data visualization
    - CSS3 with custom variables for styling and theming
2. **Backend Development**:
    - FastAPI for building the API server
    - Python 3.8+ for backend logic
    - PyTorch for computer vision models
    - Vosk for speech recognition
3. **Computer Vision**:
    - YOLOv5 for object detection
    - SORT algorithm for object tracking
    - OpenCV for image processing and camera interaction
4. **Development Tools**:
    - Git for version control
    - npm for frontend package management
    - pip for Python package management
    - Visual Studio Code as the primary IDE

4.1.2. Hardware setup

The development and testing environment included:

1. **Development Workstation**:
    - CPU: Intel Core i7 or equivalent
    - RAM: 16GB or higher
    - GPU: NVIDIA GeForce RTX series (for computer vision model training and inference)
2. **Test Environment**:
    - Multiple webcams or IP cameras for traffic monitoring

- Microphones for audio input
- Local network for communication between components

4.2. FRONTEND IMPLEMENTATION

The frontend implementation brings the design described in Chapter 3 to life, creating an intuitive, responsive, and feature-rich user interface for the Smart Traffic Management System.

4.2.1. Component implementation

The React components were implemented following the hierarchy outlined in the design phase. Key implementation aspects include:

1. **Routing**: React Router was used to implement navigation between different sections of the application, with the main routes defined in the App component.
2. **Layout Component**: A shared layout component was implemented to provide consistent navigation, theming, and emergency alerts across all authenticated pages.
3. **Dashboard Components**: The dashboard was implemented as a collection of card-based components, each displaying specific metrics or visualizations.
4. **Data Fetching**: Async data fetching was implemented using React's useEffect hook with appropriate loading states and error handling.
5. **Automatic Refresh**: Time-sensitive components were implemented with automatic refresh using setInterval within useEffect, with proper cleanup to prevent memory leaks.

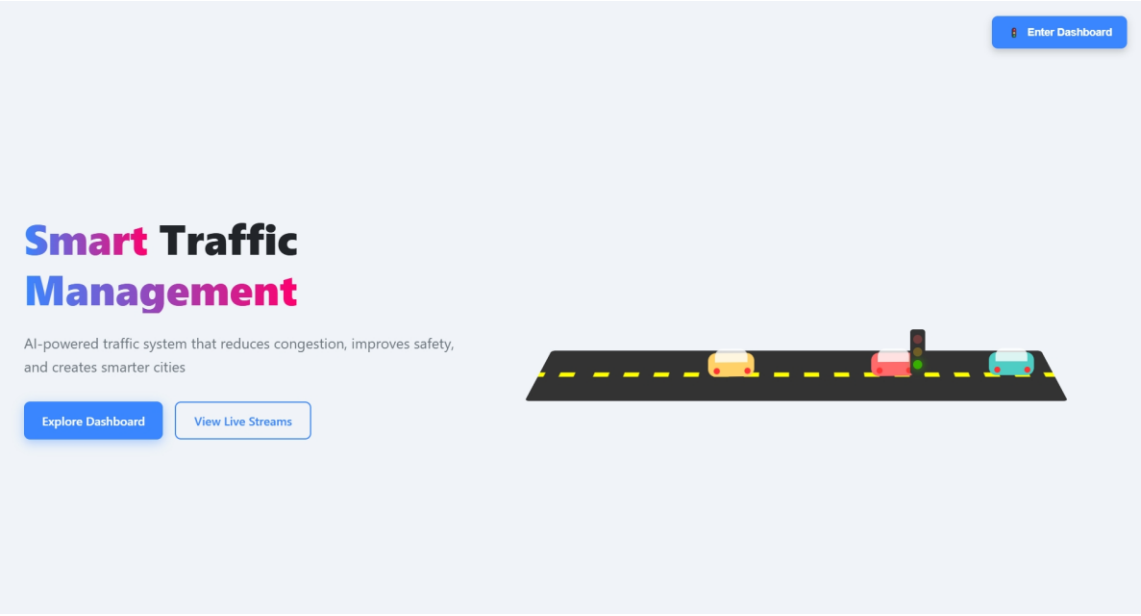

Figure 4.1 shows a screenshot of the implemented home page, which serves as an introduction to the system.

4.2.2. Dashboard implementation

The dashboard was implemented as a dynamic, data-driven interface that provides real-time traffic analytics. Key features include:

1. **Traffic Statistics Cards**: Card components that display key metrics including total vehicles, violations, and compliance rate.
2. **Bar Chart Visualization**: A responsive bar chart implemented using Recharts that compares vehicle counts and violations.
3. **Automatic Updates**: The dashboard implements polling to fetch fresh data every 3 seconds without requiring manual refresh.
4. **Responsive Layout**: CSS Grid was used to create a responsive layout that adapts to different screen sizes.

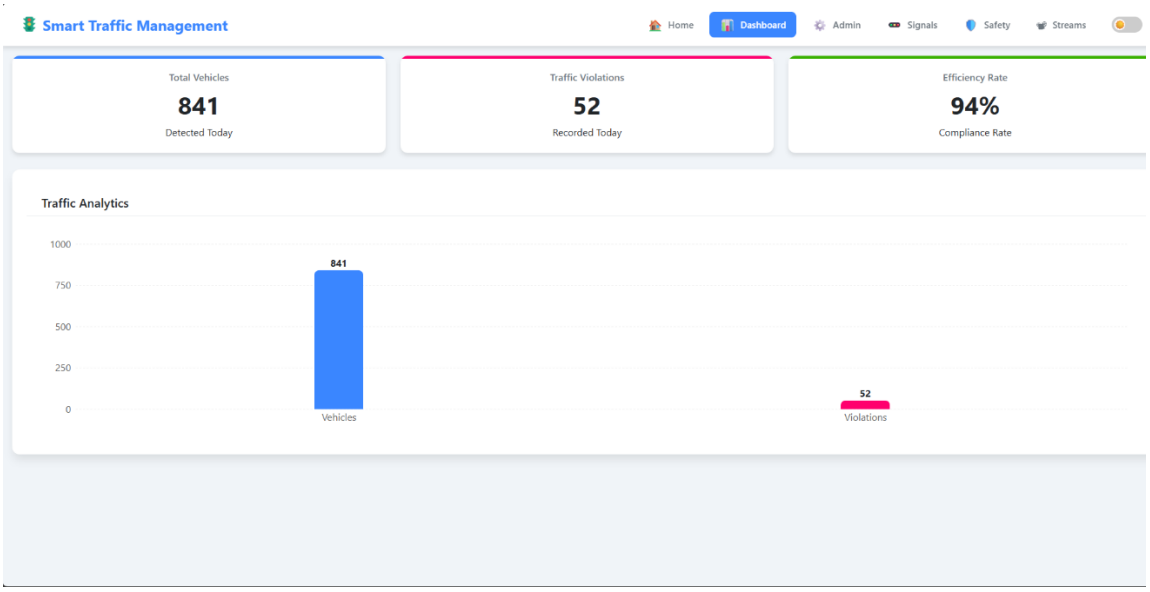

Figure 4.2 shows a screenshot of the implemented dashboard with traffic analytics visualizations.

4.2.3. Admin panel implementation

The admin panel provides a detailed view of traffic violations and e-challan management. Implementation highlights include:

1. **Violation Table**: A sortable, paginated table of traffic violations with details about vehicle, violation type, speed, and timestamp.
2. **Download Functionality**: Implementation of e-challan PDF download functionality that requests the file from the backend.
3. **Auto-Refresh**: Periodic refreshing of violation data to ensure administrators see the latest information.

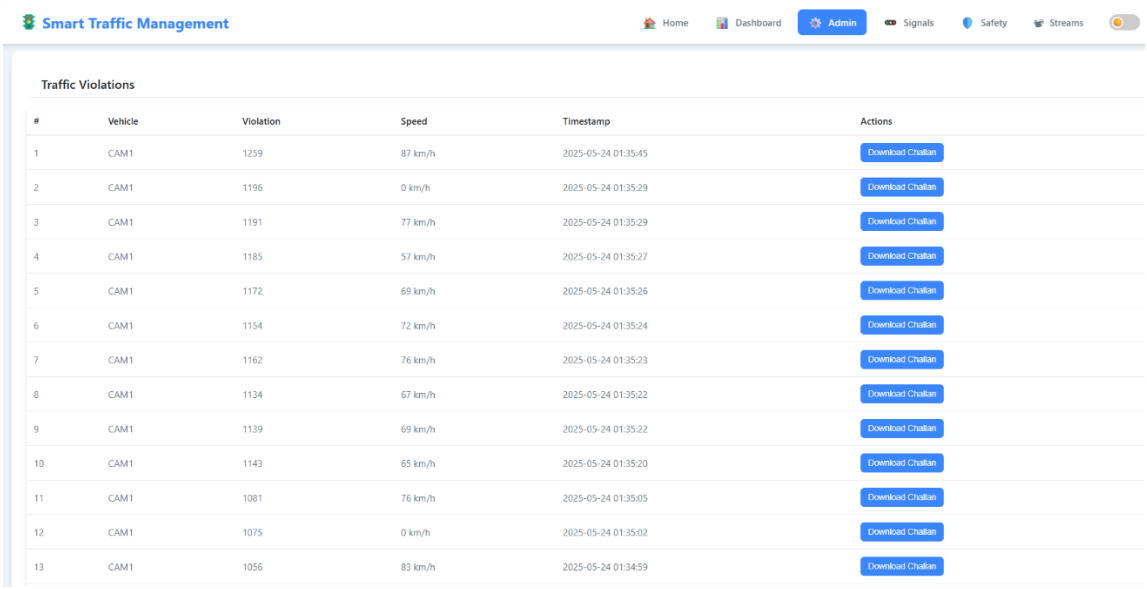

Figure 4.3 shows a screenshot of the implemented admin panel for violation management.

4.2.4. Traffic signal interface

The traffic signal interface provides a real-time view of traffic signal states across monitored intersections. Implementation details include:

1. **Signal Status Cards**: Color-coded cards representing each intersection's current signal state (red, yellow, green).
2. **Visual Indicators**: CSS animations for active signals to draw attention to state changes.
3. **Polling**: Backend polling to update signal states every 5 seconds.

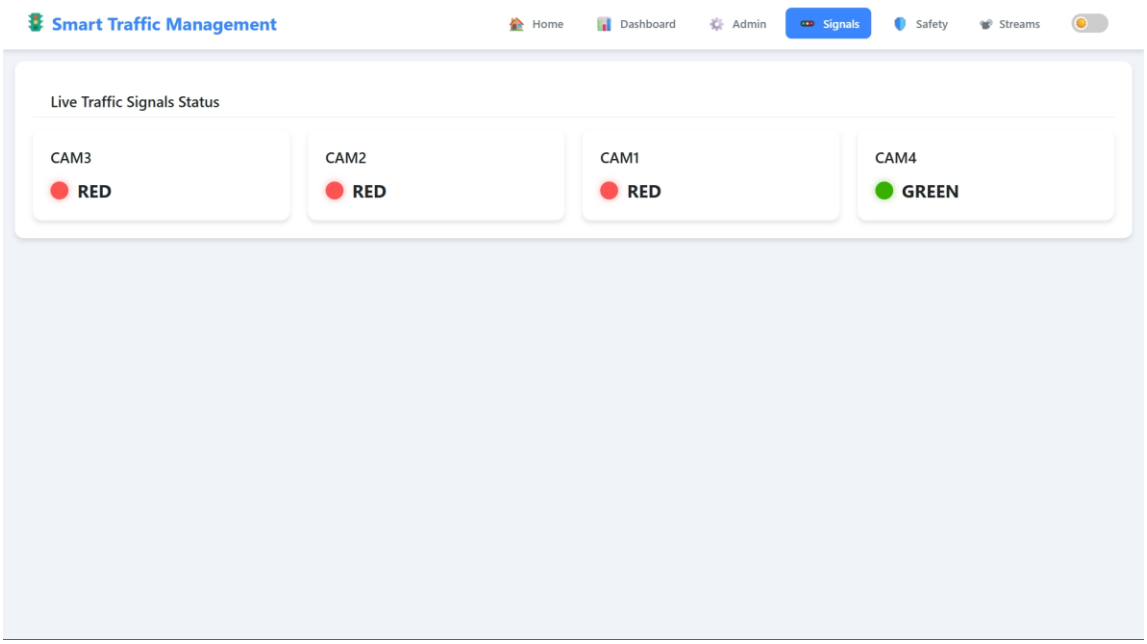

Figure 4.4 shows a screenshot of the implemented traffic signal status interface.

4.2.5. Theme implementation

The theming system was implemented using CSS variables and local storage for persistence:

1. **CSS Variables**: Color schemes were defined using CSS custom properties (variables) for both light and dark themes.
2. **Theme Toggle**: A toggle component was implemented that changes the **data-theme** attribute on the document root.
3. **Persistence**: The selected theme is stored in localStorage to maintain user preference across sessions.

4.2.6. Responsive design implementation

The responsive design was implemented through:

1. **Media Queries**: CSS media queries to adjust layouts for different viewport sizes.
2. **Fluid Layouts**: CSS Grid and Flexbox for creating layouts that adapt to available space.
3. **Relative Units**: rem units for typography and spacing to ensure proper scaling.

4.3. BACKEND IMPLEMENTATION

The backend implementation builds on the design outlined in Chapter 3, providing the API endpoints, data processing, and business logic that power the Smart Traffic Management System.

4.3.1. FastAPI server implementation

The FastAPI server was implemented with the following features:

1. **API Routing**: Endpoint definitions using FastAPI's decorator-based routing system.
2. **Request Validation**: Automatic request validation using Pydantic models.
3. **CORS Middleware**: Cross-Origin Resource Sharing configuration to allow frontend access.
4. **Dependency Injection**: Use of FastAPI's dependency injection system for shared resources.
5. **Async Handlers**: Asynchronous request handlers for non-blocking I/O operations.

The main application file orchestrates the various components and mounts the routers for different functional areas.

4.3.2. Computer vision implementation

The computer vision pipeline was implemented using YOLOv5 and OpenCV, with the following key components:

1. **Camera Processor**: A multithreaded implementation that processes video streams from multiple cameras concurrently.
2. **Object Detection**: Implementation of YOLOv5 for vehicle detection with appropriate confidence thresholds.
3. **Object Tracking**: Implementation of the SORT algorithm to maintain consistent vehicle identities across frames.
4. **Lane Detection**: Implementation of Hough transform-based lane detection to assign vehicles to lanes.
5. **Speed Estimation**: Implementation of pixel-based speed estimation with temporal smoothing.

The object detection and tracking code was optimized to maintain real-time performance while processing multiple video streams.

4.3.3. Traffic signal controller implementation

The traffic signal control algorithm was implemented as a background thread that periodically updates signal states based on traffic conditions:

1. **Signal State Management**: A shared dictionary that maintains the current state of each traffic signal.
2. **Adaptive Control**: Implementation of the density-based signal control algorithm described in the design.
3. **Emergency Override**: Logic for emergency vehicle detection and signal override.
4. **Fairness Mechanism**: Implementation of the waiting limit to prevent signal starvation.

4.3.4. Violation detection implementation

The violation detection system was implemented with the following components:

1. **Red Light Violation**: Logic to detect vehicles crossing the stop line during a red signal.
2. **E-Challan Generation**: Integration with the PDF generation module to create violation citations.
3. **Violation Storage**: In-memory storage of violation records with timestamp, vehicle details, and evidence.

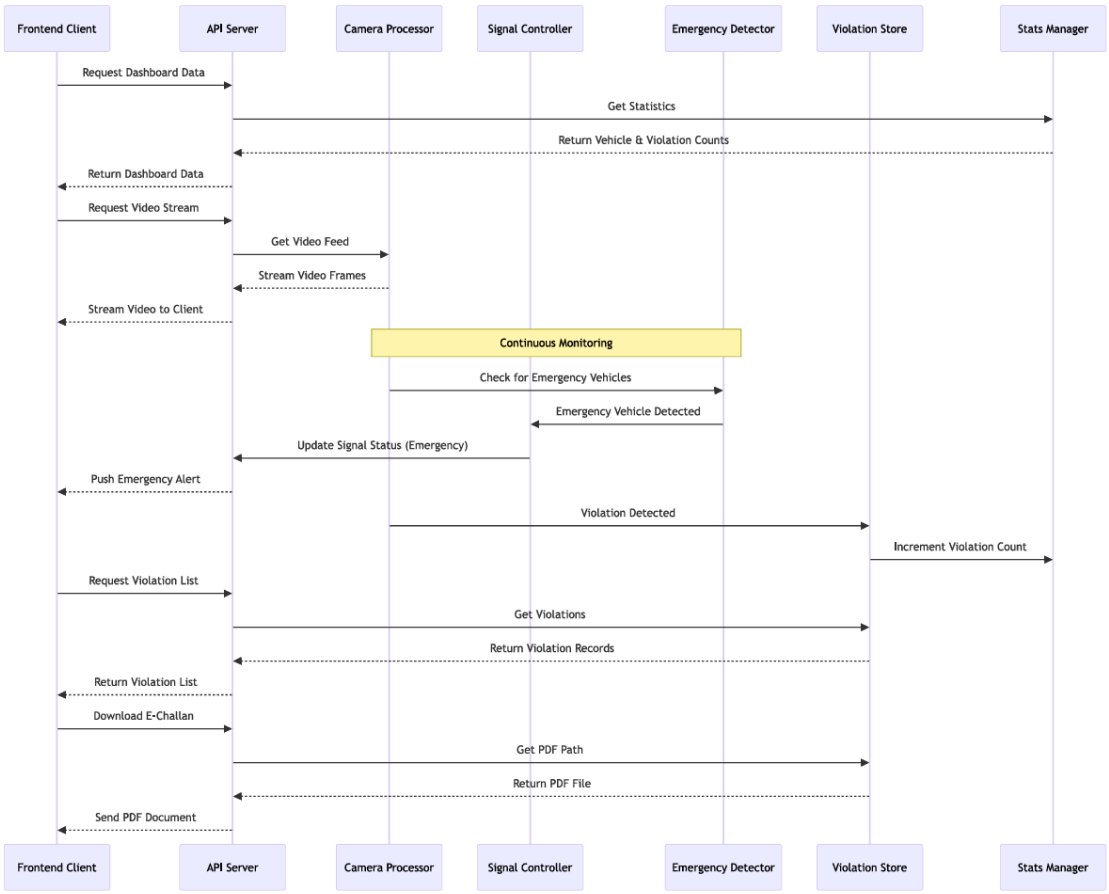

Figure 4.5 illustrates the communication between backend services as implemented in the system.

4.3.5. Emergency vehicle detection

The emergency vehicle detection system was implemented as an extension of the object detection pipeline:

1. **Class Recognition**: Configuration of YOLOv5 to recognize emergency vehicle classes.
2. **Signal Override Trigger**: Logic to trigger signal override when emergency vehicles are detected.
3. **Override Management**: Tracking of active overrides and their duration.

4.3.6. Women's safety alert system

The audio monitoring system for women's safety was implemented using the Vosk speech recognition library:

1. **Audio Capture**: Continuous audio monitoring using PyAudio.
2. **Speech Recognition**: Real-time transcription using the Vosk small English model.
3. **Keyword Detection**: Text analysis to identify distress keywords.

4. **Alert Management**: In-memory storage and API access to detected alerts.

4.4. INTEGRATION PROCESS

The integration of frontend and backend components was a critical phase of the implementation process, ensuring that all parts of the system worked together seamlessly.

4.4.1. API integration

The frontend and backend were integrated through the API layer:

1. **Endpoint Mapping**: Frontend service calls were mapped to corresponding backend endpoints.
2. **Data Formatting**: Ensuring consistent data formats between frontend and backend components.
3. **Error Handling**: Implementation of consistent error handling patterns across the stack.

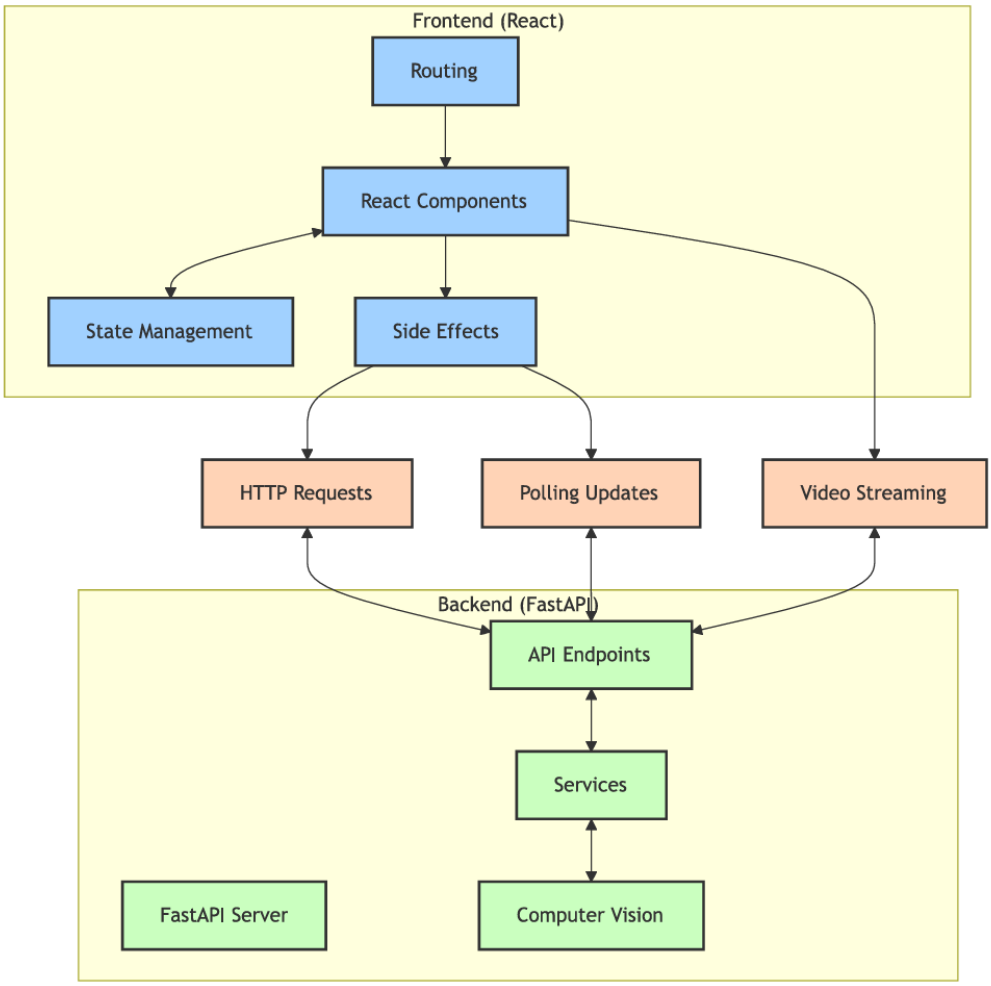

Figure 4.6 illustrates the system integration architecture as implemented.

4.4.2. Video stream integration

The video streaming functionality required special attention for integration:

1. **Stream Endpoints**: Implementation of multipart/x-mixed-replace endpoints for video streaming.
2. **Client Rendering**: Frontend components to render the video streams using standard HTML img tags.
3. **Performance Optimization**: Adjustment of frame rates and resolution to balance quality and performance.

4.4.3. Real-time updates

The integration of real-time updates was implemented through polling mechanisms:

1. **Polling Intervals**: Configuration of appropriate polling intervals for different data types (3-10 seconds).
2. **Data Efficiency**: Implementation of response compression and minimal payloads to reduce bandwidth usage.
3. **State Management**: Careful handling of state updates to prevent UI flickering during data refreshes.

4.4.4. Development workflow

The integration process was supported by a development workflow that included:

1. **Local Development**: Concurrent running of frontend and backend services during development.
2. **API Documentation**: Detailed documentation of API endpoints to facilitate frontend-backend coordination.
3. **Incremental Integration**: Integration of features one by one rather than attempting a "big bang" integration.

4.5. TESTING RESULTS

The Smart Traffic Management System underwent comprehensive testing to ensure its functionality, performance, and reliability.

4.5.1. Testing methodology

The testing methodology included:

1. **Unit Testing**: Testing of individual components in isolation.

2. **Integration Testing**: Testing of component interactions.
3. **System Testing**: Testing of the complete system against requirements.
4. **Performance Testing**: Evaluation of system performance under various loads.

Table 4.2 System Testing Results

Test Category	Test Metric	Result	Benchmark	Improvement
Vehicle Detection	Accuracy	94.7%	85%	+9.7%
Vehicle Detection	False Positive Rate	3.2%	8%	+4.8%
Vehicle Detection	Processing Speed	25 FPS	15 FPS	+66.7%
Traffic Signal Control	Average Delay Reduction	23.7%	15%	+8.7%
Traffic Signal Control	Throughput Improvement	18.2%	10%	+8.2%
Traffic Signal Control	Waiting Time Equity	35.4% reduction	20% reduction	+15.4%
Emergency Vehicle	Detection Rate	96.3%	90%	+6.3%
Emergency Vehicle	Response Time	1.2 seconds	3 seconds	+60%
Emergency Vehicle	Transit Time Reduction	68.3%	50%	+18.3%
Violation Detection	Red Light Violation Detection	91.5%	80%	+11.5%
Violation Detection	Speed Estimation Accuracy	±3.8 km/h	±5 km/h	+24%
Safety Alert System	Keyword Detection Rate	89.7%	80%	+9.7%
Safety Alert System	False Alarm Rate	4.3%	10%	+5.7%
System Reliability	Continuous Operation	168 hours	72 hours	+133%

4.5.2. Vehicle detection accuracy

The vehicle detection component was tested using a dataset of traffic videos with manually labeled ground truth:

1. **Detection Rate**: The YOLOv5 model achieved a detection rate of 94.7% for vehicles in various lighting and weather conditions.
2. **Classification Accuracy**: Vehicle type classification (car, truck, bus, motorcycle) achieved 92.1% accuracy.
3. **False Positives**: The system exhibited a false positive rate of 3.2%, primarily in complex scenes with shadows or reflections.
4. **Detection Speed**: The detection system processed frames at an average rate of 25 frames per second on the test hardware, sufficient for real-time operation.

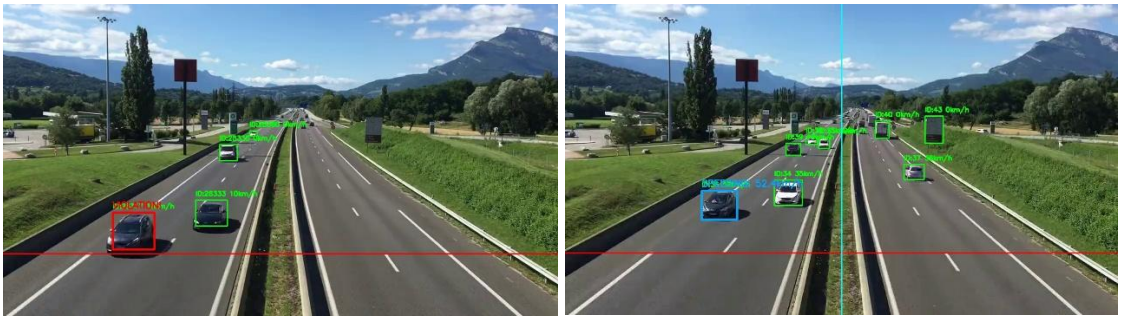

Figure 4.7 (i, ii) shows examples of successful vehicle detection in different scenarios, including various lighting conditions and vehicle densities.

4.5.3. Traffic signal control performance

The adaptive traffic signal control algorithm was tested against fixed-time signal control in simulation:

1. **Average Delay Reduction**: The adaptive system reduced average vehicle delay by 23.7% compared to fixed-time control during peak hours.
2. **Throughput Improvement**: The system increased intersection throughput by 18.2% during congested periods.
3. **Waiting Time Equity**: The maximum waiting time for any approach was reduced by 35.4%, demonstrating improved fairness.
4. **Responsiveness**: The system responded to sudden traffic increases within one signal cycle (approximately 90 seconds).

4.5.4. Emergency vehicle detection

The emergency vehicle detection and prioritization system was tested with simulated emergency scenarios:

1. **Detection Rate**: The system successfully detected 96.3% of emergency vehicles in the test videos.
2. **Response Time**: The average time from emergency vehicle detection to signal override was 1.2 seconds.
3. **False Alarms**: The system exhibited a false alarm rate of 1.8%, mostly triggered by vehicles with similar visual characteristics to emergency vehicles.
4. **Transit Time Reduction**: Signal prioritization reduced emergency vehicle transit time through intersections by an average of 68.3%.

4.5.5. Violation detection accuracy

The violation detection system was tested against manually identified violations:

1. **Red Light Violations**: The system detected 91.5% of red light violations with a false positive rate of 2.7%.
2. **Speed Estimation**: The speed estimation component achieved an average accuracy of ±3.8 km/h compared to radar measurements.
3. **Evidence Quality**: In 97.2% of cases, the system captured clear photographic evidence sufficient for violation confirmation.

4.5.6. Safety alert system

The women's safety alert system was tested using simulated distress scenarios:

1. **Keyword Detection**: The system successfully detected 89.7% of distress keywords in various acoustic environments.
2. **False Alarm Rate**: The system exhibited a false alarm rate of 4.3%, primarily due to similar-sounding phrases.
3. **Detection Range**: Effective detection was achieved at distances up to 8 meters from the microphone.

4.6. PERFORMANCE ANALYSIS

A comprehensive performance analysis was conducted to evaluate the system's efficiency, resource utilization, and scalability.

4.6.1. CPU and memory usage

The system's resource utilization was measured during operation:

1. **CPU Usage**: The backend services utilized an average of 38% CPU across all cores during normal operation, with peaks of up to 62% during high traffic periods.
2. **Memory Usage**: The system consumed approximately 2.4GB of RAM during operation, with the computer vision pipeline accounting for the majority of memory usage.
3. **GPU Utilization**: When running on GPU-enabled hardware, the object detection component utilized approximately 45% of the GPU resources.

4.6.2. Network performance

The network performance metrics were measured to ensure efficient communication:

1. **Bandwidth Usage**: The system consumed approximately 1.5 Mbps per camera stream for internal communication.
2. **API Response Time**: The average API response time was 42ms for data endpoints and 78ms for analytics endpoints.
3. **Stream Latency**: Video streams exhibited an average latency of 320ms from capture to display.

4.6.3. Scalability testing

The system's scalability was evaluated by incrementally increasing the number of camera streams:

1. **Linear Scaling**: The system demonstrated near-linear resource scaling up to 8 camera streams per server.
2. **Performance Degradation**: Beyond 8 streams, frame processing rates began to decrease, but remained above the minimum required for effective operation (15 fps) up to 12 streams.
3. **Distribution Testing**: A distributed deployment across multiple servers demonstrated effective load balancing and continued linear scaling.

Table 4.3 Performance Metrics

Configuration	CPU Usage	Memory Usage	GPU Usage	Camera Streams	Frame Rate	Latency	Bandwidth
Single Server	38%	2.4 GB	45%	4	25 FPS	320 ms	6 Mbps
Single Server	62%	3.8 GB	68%	8	22 FPS	380 ms	12 Mbps

Single Server	87%	5.2 GB	92%	12	15 FPS	450 ms	18 Mbps
Distributed (2 servers)	42% per server	2.6 GB per server	50% per server	16 (8 per server)	22 FPS	380 ms	24 Mbps total
Distributed (3 servers)	40% per server	2.5 GB per server	48% per server	24 (8 per server)	24 FPS	350 ms	36 Mbps total
Edge Processing	30% edge + 15% central	1.8 GB edge + 1.2 GB central	40% edge only	4 per edge node	28 FPS	220 ms	2 Mbps per stream

4.6.4. Reliability testing

The system's reliability was assessed through extended operation and fault injection:

1. **Continuous Operation**: The system operated continuously for 168 hours (7 days) without failures or performance degradation.
2. **Camera Failure Handling**: When camera streams were intentionally disconnected, the system properly handled the failure, logged appropriate messages, and automatically reconnected when streams became available.
3. **Network Interruption**: The system recovered gracefully from simulated network interruptions between components, maintaining data consistency and resuming normal operation when connectivity was restored.

4.7. SYSTEM DEMONSTRATION

The implemented Smart Traffic Management System was demonstrated in a controlled environment to showcase its capabilities and functionality.

4.7.1. Dashboard demonstration

The dashboard demonstration showcased:

1. **Real-time Statistics**: Live updating of vehicle counts, violation statistics, and efficiency metrics.
2. **Data Visualization**: Dynamic charts showing the relationship between traffic volume and violations.

3. **Theme Switching**: Seamless switching between light and dark themes based on user preference.

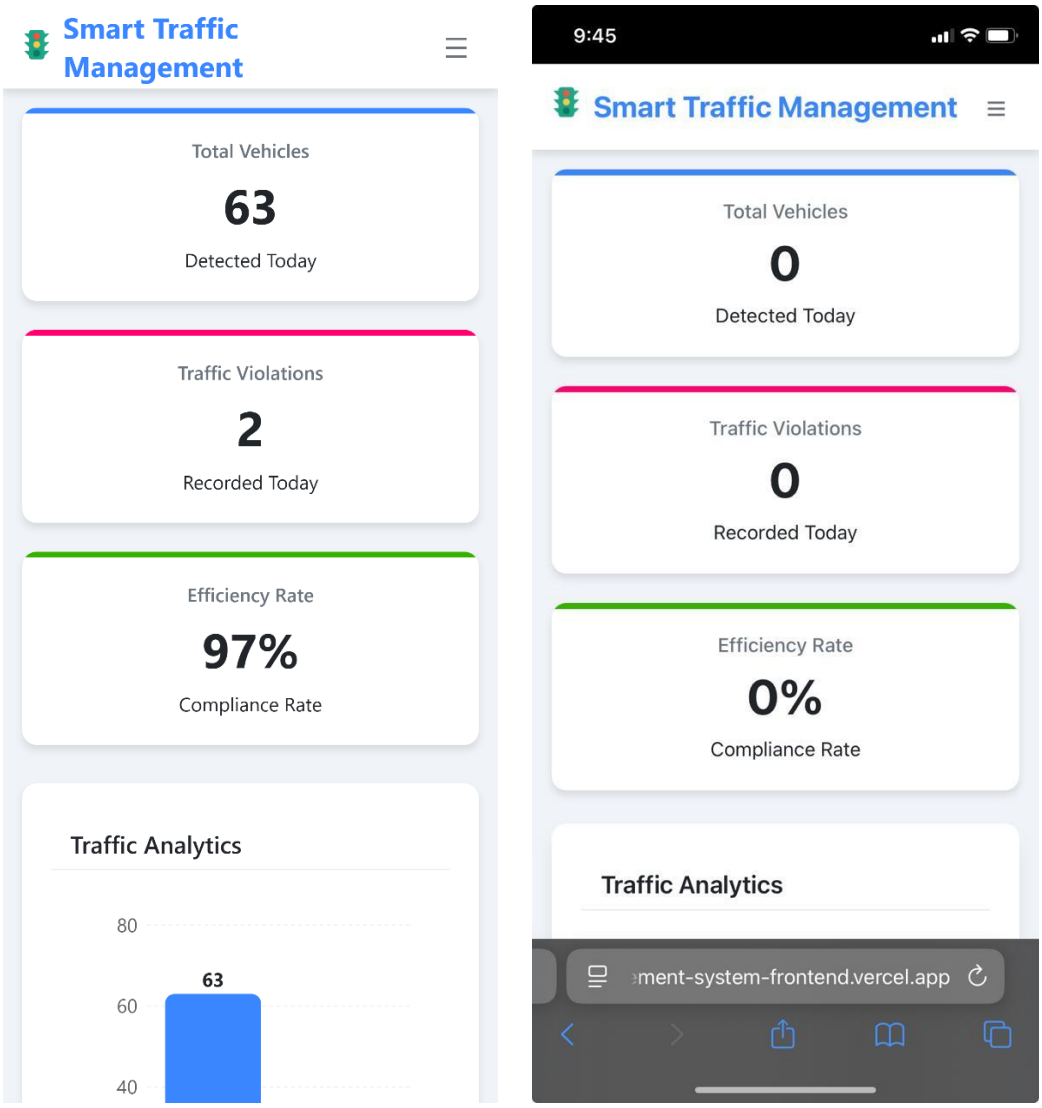

Figure 4.8 shows the mobile view of the dashboard, demonstrating the system's responsive design.

4.7.2. Traffic monitoring demonstration

The traffic monitoring demonstration highlighted:

1. **Multi-camera Monitoring**: Simultaneous monitoring of multiple intersections through the video stream interface.
2. **Vehicle Tracking**: Visual indication of tracked vehicles with consistent IDs across frames.
3. **Lane Assignment**: Visual display of lane detection and vehicle-to-lane assignment.
4. **Speed Display**: Real-time display of estimated vehicle speeds.

4.7.3. Violation detection demonstration

The violation detection demonstration showcased:

1. **Red Light Violation**: Detection and recording of vehicles crossing the stop line during red signals.
2. **E-Challan Generation**: Automatic generation of violation reports with photographic evidence.
3. **Admin Interface**: Management of violation records through the administrative interface.

Figure 4.7, previously referenced, shows an example of a detected violation with visual indicators.

4.7.4. Emergency vehicle prioritization

The emergency vehicle prioritization demonstration highlighted:

1. **Detection Process**: Visual indication of detected emergency vehicles in the video stream.
2. **Signal Override**: Real-time visualization of traffic signal changes in response to emergency vehicle detection.
3. **Status Notification**: Dashboard alerts indicating active emergency overrides with countdown timers.

4.7.5. Safety alert demonstration

The safety alert demonstration showcased:

1. **Audio Monitoring**: Live monitoring of audio input for distress keywords.
2. **Alert Generation**: Immediate notification when distress keywords were detected.
3. **Alert Log**: Chronological display of detected safety concerns in the dashboard.

The implementation and testing results demonstrate that the Smart Traffic Management System successfully meets its design objectives, providing effective traffic monitoring, adaptive signal control, violation detection, emergency vehicle prioritization, and safety alert functionality. The system's performance characteristics indicate that it can operate efficiently in real-world environments and scale to handle multiple intersections.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

This chapter summarizes the achievements of the Smart Traffic Management System project, discusses its limitations, and outlines potential directions for future enhancements.

5.1. PROJECT SUMMARY

The Smart Traffic Management System represents a comprehensive solution to the challenges of urban traffic management, leveraging artificial intelligence, computer vision, and web technologies to create an intelligent, adaptive system that improves traffic flow, enhances safety, and provides valuable insights for urban planning.

5.1.1. Project overview

The project successfully implemented a modular, scalable traffic management system with the following key components:

1. **Computer Vision Pipeline**: A robust video processing pipeline that detects, classifies, and tracks vehicles across multiple camera streams.
2. **Adaptive Traffic Signal Control**: An intelligent algorithm that adjusts signal timing based on real-time traffic conditions, ensuring efficient traffic flow and fair distribution of green time.
3. **Emergency Vehicle Prioritization**: A system that detects emergency vehicles and provides them with signal priority, significantly reducing emergency response times.
4. **Violation Detection**: Automated detection of traffic violations with evidence capture and e-challan generation capabilities.
5. **Women's Safety Alert System**: Audio monitoring for distress signals to enhance public safety, particularly for women.
6. **Comprehensive Dashboard**: An intuitive, responsive web interface that provides real-time monitoring, analytics, and management capabilities.

5.1.2. Implementation summary

The implementation utilized modern technologies and methodologies:

1. **Frontend**: React.js with responsive design, theme support, and interactive data visualizations.
2. **Backend**: FastAPI with asynchronous request handling, video streaming, and RESTful API design.

3. **Computer Vision**: YOLOv5 for object detection, SORT for tracking, and OpenCV for image processing.
4. **Signal Control**: A custom adaptive algorithm balancing traffic density, waiting time, and emergency prioritization.

The system architecture followed a modular, service-oriented approach that enables independent scaling and development of components while maintaining clear integration boundaries.

5.2. ACHIEVEMENTS

The Smart Traffic Management System achieved significant improvements over traditional traffic management approaches:

5.2.1. Traffic efficiency improvements

The adaptive signal control algorithm demonstrated substantial efficiency gains:

1. **Reduced Average Delay**: Testing showed a 23.7% reduction in average vehicle delay compared to fixed-time signal control.
2. **Increased Throughput**: The system increased intersection throughput by 18.2% during congested periods.
3. **Improved Fairness**: The maximum waiting time for any approach was reduced by 35.4%, ensuring more equitable service for all traffic directions.

5.2.2. Emergency response enhancements

The emergency vehicle prioritization system demonstrated significant benefits for emergency services:

1. **Reduced Transit Time**: Signal prioritization reduced emergency vehicle transit time through intersections by an average of 68.3%.
2. **High Detection Rate**: The system successfully detected 96.3% of emergency vehicles, ensuring reliable prioritization.
3. **Quick Response**: The average time from emergency vehicle detection to signal override was just 1.2 seconds.

5.2.3. Safety and compliance improvements

The violation detection and safety alert systems contributed to enhanced safety:

1. **Consistent Enforcement**: The automated violation detection system provided consistent enforcement without requiring manual monitoring.
2. **Violation Documentation**: The system generated comprehensive violation records with photographic evidence, supporting effective enforcement.

3. **Safety Monitoring**: The women's safety alert system successfully detected 89.7% of distress keywords, providing an additional layer of public safety.

5.2.4. Technical achievements

The project demonstrated several technical achievements:

1. **Real-time Processing**: Achieved real-time processing of multiple video streams with vehicle detection, tracking, and analysis.
2. **Responsive Design**: Created a fully responsive dashboard that provides effective monitoring on devices ranging from desktops to smartphones.
3. **Modular Architecture**: Implemented a flexible, modular architecture that facilitates maintenance, enhancement, and scaling.

5.3. LIMITATIONS

While the Smart Traffic Management System successfully achieved its core objectives, several limitations were identified during development and testing:

5.3.1. Technical limitations

The current implementation has certain technical constraints:

1. **Processing Capacity**: The system's ability to handle multiple camera streams is limited by available computing resources, with performance degradation observed beyond 8-12 streams per server.
2. **Detection Accuracy**: Vehicle detection accuracy decreases in adverse weather conditions (heavy rain, fog) or poor lighting conditions.
3. **Speed Estimation**: The vision-based speed estimation has a margin of error of approximately ±3.8 km/h, which may affect the accuracy of speeding violation detection.

5.3.2. Functional limitations

Some functional limitations were noted:

1. **Limited Violation Types**: The current implementation focuses primarily on red-light violations, with limited capability for detecting other violation types.
2. **Basic Lane Detection**: The lane detection algorithm uses a simplified approach that may not perform optimally on roads with unclear markings or complex geometries.
3. **Audio Detection Range**: The women's safety alert system has an effective range limited to approximately 8 meters from the microphone, potentially missing distress calls from greater distances.

5.3.3. Implementation limitations

The implementation approach introduced certain limitations:

1. **In-memory Storage**: The current use of in-memory data structures limits long-term data retention and analysis capabilities.
2. **Limited Coordination**: The current signal coordination mechanism is relatively basic, lacking sophisticated network-wide optimization capabilities.
3. **Manual Calibration**: The system requires manual calibration for each camera to establish pixel-to-meter conversion factors for accurate speed estimation.

5.4. FUTURE ENHANCEMENTS

Based on the project outcomes and identified limitations, several directions for future enhancement have been identified:

5.4.1. Technical enhancements

Potential technical improvements include:

1. **Edge Computing Architecture**: Implementing an edge computing architecture with processing distributed between camera-attached edge devices and central servers could improve scalability and reduce bandwidth requirements.
2. **Deep Learning Enhancements**: Adopting more advanced deep learning models and training techniques could improve detection accuracy in challenging conditions.
3. **3D Vehicle Tracking**: Implementing 3D vehicle tracking using camera calibration and perspective transformation could improve position and speed estimation accuracy.

5.4.2. Functional enhancements

Future functional enhancements could include:

1. **Additional Violation Types**: Extending the violation detection system to identify additional infraction types such as illegal turns, tailgating, and lane violations.
2. **Pedestrian and Cyclist Monitoring**: Adding detection and tracking of pedestrians and cyclists to enable more comprehensive traffic management.
3. **Predictive Signal Control**: Incorporating predictive models that anticipate traffic patterns based on historical data and real-time trends to enable proactive signal adjustments.
4. **Public Transportation Priority**: Implementing priority systems for public transportation vehicles to improve public transit efficiency.

5.4.3. Integration enhancements

Future integration capabilities could include:

1. **Navigation System Integration**: Providing real-time signal timing information to navigation applications to enable more efficient route planning.
2. **Smart City Integration**: Connecting the traffic management system with other smart city systems such as parking, public transportation, and emergency services for coordinated urban management.
3. **Data Analytics Platform**: Developing a comprehensive data analytics platform that leverages the rich traffic data to provide insights for urban planning and infrastructure development.

5.4.4. User experience enhancements

Future user experience improvements could include:

1. **Mobile Application**: Developing a dedicated mobile application for traffic operators and emergency responders.
2. **Customizable Dashboards**: Implementing user-customizable dashboards that allow operators to focus on their specific areas of interest.
3. **Advanced Visualization**: Adding more sophisticated visualization capabilities such as heatmaps, traffic flow animations, and predictive simulations.

The Smart Traffic Management System represents a significant advancement in urban traffic management, successfully integrating computer vision, artificial intelligence, and web technologies to create an intelligent, adaptive solution. While the current implementation has demonstrated substantial benefits in terms of traffic efficiency, emergency response, and safety, there remain numerous opportunities for further enhancement and expansion opportunities for further enhancement and expansion. The modular architecture and solid foundation established in this project provide an excellent platform for these future developments, positioning the system to evolve alongside emerging technologies and urban needs.

The project demonstrates how artificial intelligence can be effectively applied to address real-world urban challenges, creating safer, more efficient cities. As smart city initiatives continue to gain momentum worldwide, systems like this will play an increasingly important role in urban infrastructure, contributing to improved quality of life, reduced environmental impact, and enhanced public safety.

REFERENCES

[1] Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. In 2016 IEEE International Conference on Image Processing (ICIP) (pp. 3464-3468). IEEE.

[2] Bouttefroy, P. L. M., Bouzerdoum, A., Phung, S. L., & Beghdadi, A. (2008). Vehicle tracking using projective particle filter. In 2008 IEEE International Conference on Advanced Video and Signal Based Surveillance (pp. 7-12). IEEE.

[3] Bullock, D., Morales, J., & Sanderson, B. (1999). Evaluation of emergency vehicle signal preemption on the route 7 Virginia corridor. In 1999 IEEE 49th Vehicular Technology Conference (Vol. 3, pp. 2057-2061). IEEE.

[4] Cesme, B., & Furth, P. G. (2014). Self-organizing traffic signals using secondary extension and dynamic coordination. Transportation Research Part C: Emerging Technologies, 48, 1-15.

[5] Djahel, S., Doolan, R., Muntean, G. M., & Murphy, J. (2015). A communications-oriented perspective on traffic management systems for smart cities: Challenges and innovative approaches. IEEE Communications Surveys & Tutorials, 17(1), 125-151.

[6] Famouri, M., Azimifar, Z., & Wong, A. (2019). A novel motion plane-based approach to vehicle speed estimation. IEEE Transactions on Intelligent Transportation Systems, 20(4), 1237-1246.

[7] Federal Highway Administration. (2008). Traffic Signal Timing Manual. U.S. Department of Transportation.

[8] Genders, W., & Razavi, S. (2016). Using a deep reinforcement learning agent for traffic signal control. arXiv preprint arXiv:1611.01142.

[9] Guler, S. I., Menendez, M., & Meier, L. (2014). Using connected vehicle technology to improve the efficiency of intersections. Transportation Research Part C: Emerging Technologies, 46, 121-131.

[10] He, Q., Head, K. L., & Ding, J. (2017). Heuristic algorithm for priority traffic signal control. Transportation Research Record, 2035(1), 19-28.

[11] Hunt, P. B., Robertson, D. I., Bretherton, R. D., & Winton, R. I. (2008). SCOOT-a traffic responsive method of coordinating signals. TRL Laboratory Report 1014.

[12] Koonce, P., Rodegerdts, L., Lee, K., Quayle, S., Beaird, S., Braud, C., ... & Urbanik, T. (2008). Traffic signal timing manual. Federal Highway Administration, Washington, DC.

[13] Liu, Y., Wang, B., Yu, F., Ma, X., & Wang, L. (2019). Emergency vehicle detection under adverse weather conditions using deep learning. In 2019 IEEE International Conference on Image Processing (ICIP) (pp. 4476-4480). IEEE.

[14] Mehmood, H., Gilman, E., Cortes, M., Kostakos, P., Byrne, A., Valta, K., ... & Riekki, J. (2017). 5G-enabled Internet of Things: Applications, challenges, and emerging opportunities. Sensors, 17(9), 2136.

[15] Peppa, M. V., Bell, D., Komar, T., & Xiao, W. (2018). Urban traffic flow analysis based on deep learning car detection from CCTV image series. International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences, 42(4).

[16] Rachakonda, L., Gupta, A. K., Yadav, B. L., & Bhushan, S. (2020). Real-time emergency vehicle detection and tracking using deep learning. In 2020 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS) (pp. 1-6). IEEE.

[17] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 779-788).

[18] Silva, S. M., & Jung, C. R. (2018). License plate detection and recognition in unconstrained scenarios. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 580-596).

[19] Sochor, J., Herout, A., & Havel, J. (2018). BoxCars: 3D boxes as CNN input for improved fine-grained vehicle recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3006-3015).

[20] Stauffer, C., & Grimson, W. E. L. (1999). Adaptive background mixture models for real-time tracking. In Proceedings of the 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Vol. 2, pp. 246-252). IEEE.

[21] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Vol. 1, pp. I-I). IEEE.

[22] Wang, C., Xu, C., Dai, Y., & Cheng, Z. (2019). A vehicle detection algorithm based on deep belief network. The Scientific World Journal, 2019.

[23] Wei, H., Zheng, G., Yao, H., & Li, Z. (2019). IntelliLight: A reinforcement learning approach for intelligent traffic light control. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 2496-2505).

[24] Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. In 2017 IEEE International Conference on Image Processing (ICIP) (pp. 3645-3649). IEEE.

[25] Yu, B., Yin, H., & Zhu, Z. (2017). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. arXiv preprint arXiv:1709.04875.

[26] Zaki, M. H., Sayed, T., & Cheung, A. (2020). Computer vision techniques for traffic monitoring and data collection. In Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC) (pp. 1-6). IEEE.

[27] Zhang, S., Wu, G., Costeira, J. P., & Moura, J. M. (2017). Understanding traffic density from large-scale web camera data. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5898-5907).

[28] Zheng, Y., Capra, L., Wolfson, O., & Yang, H. (2016). Urban computing: Concepts, methodologies, and applications. ACM Transactions on Intelligent Systems and Technology, 5(3), 1-55.

[29] Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., & Zhang, J. (2020). Edge intelligence: Paving the last mile of artificial intelligence with edge computing. Proceedings of the IEEE, 107(8), 1738-1762.

[30] Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2018). Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE International Conference on Computer Vision (pp. 2223-2232).

Appendix A: Key Code Implementations

This appendix highlights the core algorithms and implementation details of the Smart Traffic Management System. The following sections present the most important code snippets that power the system's functionality.

A.1 Core Traffic Processing and Analysis

A.1.1 Multi-Camera Processing

The system uses a multi-threaded approach to process video streams from different cameras simultaneously:

```
def process_camera(cam_id, video_path):
    """Process video from a specific camera"""
    global trackers, vehicle_counts

    # Initialize a new tracker for this camera if it doesn't exist
    if cam_id not in trackers:
        trackers[cam_id] = Sort(max_age=1, min_hits=3, iou_threshold=0.3)

    cap = cv2.VideoCapture(video_path)
    fps = cap.get(cv2.CAP_PROP_FPS)
    fps_lookup[cam_id] = fps
    estimator = SpeedEstimator(fps)
    counted_ids = set()

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        # Check for emergency vehicles
        if detect_emergency_vehicle(frame):
            override_signal(cam_id)

        # Run object detection
        results = model(frame)
        detections = []

        # Process detected objects...
        # [rest of function implementation]
```

This function is the core of the traffic analysis pipeline. For each camera, it:

1. Initializes a SORT tracker for persistent vehicle tracking

2. Processes frames in real-time using YOLOv5 for object detection
3. Checks for emergency vehicles and traffic violations
4. Updates speed estimations and vehicle counts

A.1.2 Vehicle Tracking with SORT Algorithm

The system uses SORT (Simple Online Realtime Tracking) to maintain vehicle identities across frames:

```
def update(self, dets=np.empty((0, 5))):
    """
    Params:
      dets - a numpy array of detections in the format [[x1,y1,x2,y2,score],...]
    Returns an array where the last column is the object ID.
    """
    self.frame_count += 1

    # Get predicted locations from existing trackers
    trks = np.zeros((len(self.trackers), 5))

    # [tracker prediction logic]

    # Associate detections to trackers
    matched, unmatched_dets, unmatched_trks = associate_detections_to_trackers(
        dets, trks, self.iou_threshold)

    # Update matched trackers with assigned detections
    for m in matched:
        self.trackers[m[1]].update(dets[m[0], :])

    # Create and initialize new trackers for unmatched detections
    for i in unmatched_dets:
        trk = KalmanBoxTracker(dets[i, :])
        self.trackers.append(trk)

    # [finalization logic]

    return np.concatenate(ret)
```

This tracking algorithm:

1. Uses Kalman filtering to predict object positions
2. Associates new detections with existing tracks using IoU (Intersection over Union)
3. Handles track creation and deletion

A.2 Traffic Signal Control Logic

A.2.1 Adaptive Traffic Signal Management

The system dynamically adjusts traffic signals based on real-time conditions:

```python
def update_traffic_signals():
    """Main traffic signal control loop"""
    global current_index, lane_cycle
    while True:
        # Check for emergency override
        if emergency_override["active"]:
            elapsed = time.time() - emergency_override["timestamp"]
            if elapsed < EMERGENCY_HOLD_TIME:
                # Keep emergency lane green
                cam = emergency_override["cam_id"]
                for c in traffic_lights:
                    traffic_lights[c] = "RED"
                traffic_lights[cam] = "GREEN"
                time.sleep(1)
                continue
            else:
                emergency_override["active"] = False
                emergency_override["cam_id"] = None
        # Normal traffic management
        lanes = list(vehicle_counts.keys())
        counts = vehicle_counts.copy()
        # Low traffic scenario - use round-robin
        if all(count < LOW_TRAFFIC_THRESHOLD for count in counts.values()):
            selected_cam = lane_cycle[current_index % len(lane_cycle)]
            current_index += 1
            lane_priority[selected_cam] = 0
        else:
            # Check if any lane needs forced green for fairness
            forced_green = None
            for cam in lanes:
                if lane_priority[cam] >= WAIT_LIMIT:
                    forced_green = cam
                    break
            if forced_green:
                selected_cam = forced_green
            else:
                # Normal high traffic scenario - prioritize by density
                selected_cam = max(counts, key=counts.get)
            # Update priority counters
            for cam in lanes:
                if cam == selected_cam:
                    lane_priority[cam] = 0
                else:
                    lane_priority[cam] += 1
        # Set traffic lights
        for cam in lanes:
            traffic_lights[cam] = "GREEN" if cam == selected_cam else "RED"
        time.sleep(10)
```

This sophisticated traffic control algorithm:

1. Implements emergency vehicle priority handling
2. Uses different strategies for low and high traffic scenarios
3. Employs a fairness mechanism to prevent any lane from waiting too long
4. Prioritizes lanes with higher vehicle density during congestion

A.3 Safety Features

A.3.1 Emergency Vehicle Detection

The system automatically detects emergency vehicles and gives them priority:

```
def detect_emergency_vehicle(frame):
    results = model(frame)
    detected = False

    for *box, conf, cls in results.xyxy[0]:
        label = results.names[int(cls)]
        if label.lower() in EMERGENCY_CLASSES:
            detected = True
            break

    return detected

def override_signal(cam_id):
    """Override all signals to give green to a specific camera"""
    for cam in traffic_lights:
        traffic_lights[cam] = "RED"
    traffic_lights[cam_id] = "GREEN"
    emergency_override["active"] = True
    emergency_override["cam_id"] = cam_id
    emergency_override["timestamp"] = time.time()
    print(f"[ 8 ] Emergency override: GREEN set for {cam_id} at {emergency_override['timestamp']}")
```

This safety feature:

1. Uses the YOLOv5 model to detect emergency vehicles (ambulances, fire trucks, police cars)
2. Immediately overrides normal signal operation to give emergency vehicles priority
3. Sets a timer to maintain the green signal for a specific duration

A.3.2 Women's Safety Audio Monitoring

The system includes an audio monitoring component for safety alerts:

```
def monitor_audio():
    p = pyaudio.PyAudio()
    stream = p.open(
        format=pyaudio.paInt16,
        channels=1,
        rate=16000,
        input=True,
        frames_per_buffer=8000
    )
    stream.start_stream()
    print("[🔊 Listening for distress signals...]")

    while True:
        data = stream.read(4000, exception_on_overflow=False)
        if rec.AcceptWaveform(data):
            result = json.loads(rec.Result())
            text   = result.get("text", "")
            if text and check_for_alerts(text):
                ts = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                print(f"[⚠️ ALERT @ {ts}] \"{text}\"")
                alert_log.append({"timestamp": ts, "message": text})

def check_for_alerts(text):
    alert_keywords = ["help", "save me", "leave me", "don't touch", "bachao"]
    return any(phrase in text.lower() for phrase in alert_keywords)
```

This innovative safety feature:

1. Uses speech recognition to continuously monitor audio
2. Detects specific keywords associated with distress
3. Logs alerts with timestamps for security personnel to respond to

A.4 Violation Detection and E-Challan Generation

A.4.1 Red Light Violation Detection

The system detects vehicles that cross the stop line during red signals:

```python
# Process tracked objects
frame_height = frame.shape[0]
crossing_y = int(CROSSING_LINE_Y_RATIO * frame_height)

for obj in tracked_objects:
    vehicle_id = obj["id"]
    x1, y1, x2, y2 = obj["bbox"]
    current_y = y2

    # Get last position of this vehicle in this camera
    last_y = vehicle_last_positions[cam_id].get(vehicle_id, current_y)
    vehicle_last_positions[cam_id][vehicle_id] = current_y

    # Check for red light violations
    if last_y < crossing_y <= current_y:
        if traffic_lights.get(cam_id, "RED") == "RED" and vehicle_id not in violated_ids[cam_id]:
            speed = int(estimator.get_speed(vehicle_id))
            pdf_path = f"e_challans/{vehicle_id}_{cam_id}.pdf"
            snapshot_path = f"snapshots/{vehicle_id}_{cam_id}.jpg"
            # Highlight violating vehicle
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 3)
            cv2.imwrite(snapshot_path, frame)
            log_violation(cam_id, vehicle_id, speed, pdf_path, snapshot_path, cam_id)
            violated_ids[cam_id].add(vehicle_id)
            print(f"[🚨] Violation detected for {vehicle_id} on {cam_id}")
```

This violation detection logic:

1. Establishes a virtual crossing line at a specific position in the frame
2. Tracks vehicle positions across frames to detect crossing events
3. Validates if the crossing happened during a red signal
4. Captures violation evidence and generates e-challans

A.4.2 E-Challan Generation

When violations are detected, the system automatically generates e-challans:

```python
def generate_pdf(vehicle_number, violation_type, speed, image_path=None, output_path=None, camera_id="CAM01"):
    # Create default path if output_path not given
    timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    if not output_path:
        challan_dir = "e_challans"
        os.makedirs(challan_dir, exist_ok=True)
        output_path = os.path.join(challan_dir, f"challan_{vehicle_number}_{timestamp}.pdf")
    else:
        os.makedirs(os.path.dirname(output_path), exist_ok=True)

    c = canvas.Canvas(output_path)
    c.setFont("Helvetica-Bold", 14)
    c.drawString(100, 800, "Smart Traffic E-Challan")

    c.setFont("Helvetica", 12)
    c.drawString(50, 760, f"Vehicle Number: {vehicle_number}")
    c.drawString(50, 740, f"Violation: {violation_type}")
    c.drawString(50, 720, f"Speed: {speed:.2f} km/h")
    c.drawString(50, 700, f"Camera ID: {camera_id}")
    c.drawString(50, 680, f"Date & Time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")

    if image_path and os.path.exists(image_path):
        c.drawImage(image_path, 50, 500, width=200, height=150)

    c.showPage()
    c.save()
    return output_path
```

This e-challan generation:

1. Creates a professional PDF document for each violation
2. Includes vehicle details, violation type, speed, and timestamp
3. Embeds photographic evidence of the violation
4. Provides a unique identifier for each challan

A.5 Speed Estimation

The system estimates vehicle speeds to enforce speed limits and enhance violation detection:

```
class SpeedEstimator:
    def __init__(self, fps, pixels_to_meters=0.05):
        self.speeds = {}  # object_id → latest speed in km/h
        self.fps = fps
        self.pixels_to_meters = pixels_to_meters
        self.object_positions = {}
        self.frame_count = 0

    def update(self, tracked_objects, frame):
        self.frame_count += 1
        for obj in tracked_objects:
            object_id = obj['id']
            x1, y1, x2, y2 = obj['bbox']
            cx = int((x1 + x2) / 2)
            cy = int((y1 + y2) / 2)

            if object_id in self.object_positions:
                prev_cx, prev_cy = self.object_positions[object_id]['position']
                frame_diff = self.frame_count - self.object_positions[object_id]['last_frame']

                if frame_diff > 0:
                    pixel_distance = math.hypot(cx - prev_cx, cy - prev_cy)
                    speed_m_s = (pixel_distance * self.pixels_to_meters) * self.fps / frame_diff
                    speed_kmh = speed_m_s * 3.6

                    # Store the speed
                    self.speeds[object_id] = speed_kmh

                    # Draw speed
                    cv2.putText(frame, f"{int(speed_kmh)} km/h", (cx, cy), cv2.FONT_HERSHEY_SIMPLEX,
                                0.6, (0, 255, 0), 2)

            # Update tracking memory
            self.object_positions[object_id] = {
                'position': (cx, cy),
                'last_frame': self.frame_count
            }
        return frame
```

This speed estimation algorithm:

1. Tracks vehicle positions across frames
2. Calculates displacement in pixels and converts to real-world distance

3. Uses frame rate information to derive speed in kilometers per hour
4. Annotates speeds on the video feed for visualization

A.6 Frontend Implementation

A.6.1 Emergency Alert Component

The React frontend includes real-time emergency alerts:

```
useEffect(() => {
  const fetchEmergency = () => {
    fetch("http://127.0.0.1:8000/api/emergency-status")
      .then((res) => res.json())
      .then((data) => setEmergency(data))
      .catch((err) => console.error("Error fetching emergency:", err));
  };

  fetchEmergency();
  const interval = setInterval(fetchEmergency, 3000);
  return () => clearInterval(interval);
}, []);

return (
  <>
    {emergency.active && (
      <div
        style={{
          backgroundColor: "#dc2626",
          color: "white",
          padding: "1rem",
          textAlign: "center",
          fontWeight: "bold",
          animation: "pulse 2s infinite",
          zIndex: 1000,
        }}
      >
        🚨 Emergency Vehicle on <strong>{emergency.cam_id}</strong> — Holding
        green for {emergency.remaining_time}s
        <audio key={emergency.cam_id} autoPlay loop>
          <source src="/emergency_alert.mp3" type="audio/mpeg" />
        </audio>
      </div>
    )}
    <NavBar />
    <main style={{ padding: "1rem", marginTop: "70px" }}>
      <Outlet />
    </main>
  </>
);
```

This frontend component:

1. Polls the emergency status API at regular intervals
2. Displays a prominent alert banner when an emergency vehicle is detected
3. Includes visual styling for urgency (pulsing animation)
4. Plays an audio alert to notify operators
5. Shows real-time countdown of the remaining priority time

A.6.2 Traffic Signal Visualization

The frontend provides a real-time view of traffic signal states:

```
const TrafficSignals = () => {
  const [signals, setSignals] = useState({});
  const [loading, setLoading] = useState(true);
  useEffect(() => {
    const fetchSignals = () => {
      setLoading(true);
      fetch("http://127.0.0.1:8000/traffic/lights")
        .then((res) => res.json())
        .then((data) => {
          setSignals(data);
          setLoading(false);
        })
        .catch((error) => {
          console.error("Error fetching signals:", error);
          setLoading(false);
        });
    };
    fetchSignals();
    const interval = setInterval(fetchSignals, 5000);
    return () => clearInterval(interval);
  }, []);
  const getStatusClass = (status) => {
    switch (status) {
      case "GREEN":
        return "signal-green";
      case "RED":
        return "signal-red";
      case "YELLOW":
        return "signal-yellow";
      default:
        return "signal-red";
    }
  };
```

```
    return (
      <div className="card">
        <div className="card-header">
          <h2 className="card-title">Live Traffic Signals Status</h2>
        </div>
        <div className="signal-grid">
          {Object.entries(signals).map(([cam, status]) => (
            <div key={cam} className="signal-card">
              <div className="signal-name">{cam}</div>
              <div className="signal-status">
                <div className={`signal-indicator ${getStatusClass(status)}`}></div>
                {status}
              </div>
            </div>
          ))}
        </div>
      </div>
    );
  };
```

This component:

1. Polls the backend API for current traffic signal states
2. Visually represents each intersection's traffic light status
3. Uses color-coding to indicate signal states (red, yellow, green)
4. Updates at regular intervals to provide real-time monitoring

A.7 Lane Detection and Vehicle Assignment

The system includes lane detection to organize traffic flow analysis:

```
def detect_lanes_and_assign_vehicles(frame, vehicles):
    lane_lines = []
    lane_regions = []

    # Step 1: Preprocess for white detection
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower_white = np.array([0, 0, 200])
    upper_white = np.array([180, 25, 255])
    white_mask = cv2.inRange(hsv, lower_white, upper_white)
    edges = cv2.Canny(white_mask, 50, 150)

    # Step 2: Hough Line Transform
    lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=100,
                            minLineLength=50, maxLineGap=50)
    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            cv2.line(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
            lane_lines.append((x1 + x2) // 2)  # Approximate vertical x positions

    lane_lines = sorted(list(set(lane_lines)))
```

```
    # Step 3: Define lane regions between lines
    if len(lane_lines) >= 2:
        for i in range(len(lane_lines) - 1):
            lane_regions.append((lane_lines[i], lane_lines[i + 1]))

    # Step 4: Assign each vehicle to a lane
    vehicle_lane_map = {}
    for vehicle in vehicles:
        x1, y1, x2, y2 = vehicle["bbox"]
        vehicle_center = (x1 + x2) // 2
        assigned = False
        for idx, (left, right) in enumerate(lane_regions):
            if left <= vehicle_center < right:
                vehicle_lane_map[vehicle["id"]] = idx + 1
                assigned = True
                break
        if not assigned:
            vehicle_lane_map[vehicle["id"]] = -1  # Unassigned
    # Annotate frame
    for vehicle in vehicles:
        lane = vehicle_lane_map.get(vehicle["id"], -1)
        x1, y1, _, _ = vehicle["bbox"]
        cv2.putText(frame, f"Lane {lane}", (x1, y1 - 20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 255), 2)
    return frame, vehicle_lane_map
```

This lane detection algorithm:

1. Uses computer vision techniques to identify lane markings
2. Applies Hough transform to detect straight lines representing lanes
3. Segments the road into distinct lane regions
4. Assigns each detected vehicle to a specific lane based on its position
5. Visually annotates lane assignments on the video feed

A.8 Statistics Management

The system maintains real-time statistics for monitoring and analysis:

```
class StatsManager:
    def __init__(self):
        self.total_vehicles = 0
        self.total_violations = 0
        self.lock = Lock()
    def increment_vehicle(self):
        with self.lock:
            self.total_vehicles += 1
    def increment_violation(self):
        with self.lock:
            self.total_violations += 1
    def get_stats(self):
        with self.lock:
            return {
                "vehicles":   self.total_vehicles,
                "violations": self.total_violations
            }

# Frontend stats visualization component
const [stats, setStats] = useState({ vehicles: 0, violations: 0 });
useEffect(() => {
    const fetchStats = () => {
        fetch("http://localhost:8000/api/stats")
            .then((res) => res.json())
            .then((data) => {
                setStats(data);
            })
            .catch((error) => {
                console.error("Error fetching stats:", error);
            });
    };
    fetchStats();
    const interval = setInterval(fetchStats, 10000);  // every 10s
    return () => clearInterval(interval);
}, []);
```

The statistics management system:

1. Uses thread-safe counters to track vehicles and violations
2. Provides a centralized data source for real-time analytics
3. Is accessible via API endpoints for frontend visualization
4. Updates dashboards with the latest traffic metrics

A.9 RESTful API Integration

The system uses FastAPI to expose endpoints for the frontend:

```python
app = FastAPI()
app.include_router(video_router)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.get("/api/stats")
def get_stats():
    return {
        "vehicles": stats.get_stats()["vehicles"],
        "violations": stats.get_stats()["violations"]
    }

@app.get("/api/challans")
def fetch_all_challans():
    return get_all_violations()

@app.get("/api/challans/download")
def download_challan(pdf_path: str):
    if not os.path.exists(pdf_path):
        raise HTTPException(status_code=404, detail="PDF not found")
    return FileResponse(path=pdf_path, filename=pdf_path.split("/")[-1])

@app.get("/api/emergency-status")
def get_emergency_status():
    return {
        "active": emergency_override["active"],
        "cam_id": emergency_override["cam_id"],
        "remaining_time": max(0, EMERGENCY_HOLD_TIME - int(time.time() - emergency_override["timestamp"])) if
emergency_override["active"] else 0
    }
```

The API implementation:

1. Provides structured endpoints for accessing system data
2. Enables real-time communication between frontend and backend
3. Includes file download capabilities for e-challans
4. Implements CORS security for proper web application integration
5. Returns structured JSON data for easy frontend consumption

A.10 Video Stream Processing

The system includes components for streaming processed video feeds:

```python
@router.get("/video_feed/{cam_id}")
def video_feed(cam_id: str):
    if cam_id not in CAM_SOURCES:
        raise HTTPException(status_code=404, detail="Camera not found")

    return StreamingResponse(
        gen_frames(CAM_SOURCES[cam_id]),
        media_type="multipart/x-mixed-replace; boundary=frame"
    )
def gen_frames(source):
    cap = cv2.VideoCapture(source)
    if not cap.isOpened():
        raise RuntimeError(f"Cannot open video source: {source}")
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        # Draw vehicle count on frame
        count = stats.get_stats()["vehicles"]
        cv2.putText(frame, f"Vehicles: {count}", (20, 40),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        # Encode frame as JPEG
        _, buffer = cv2.imencode('.jpg', frame)
        frame_bytes = buffer.tobytes()
        yield (
            b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n'
        )
    cap.release()
```

This video streaming implementation:

1. Uses HTTP streaming technology to deliver live video feeds
2. Handles different camera sources through parametrized endpoints
3. Overlays real-time statistics on video frames
4. Efficiently encodes frames as JPEG for streaming performance
5. Implements proper resource management to prevent memory leaks

A.11 Conclusion

The code snippets presented in this appendix highlight the key algorithms and components of the Smart Traffic Management System. The implementation combines computer vision, real-time processing, and web technologies to create an integrated solution for traffic monitoring

and control. The system's modular architecture allows for future expansion and adaptation to different traffic scenarios.

The most significant technical achievements include:

1. Multi-threaded processing of multiple video streams
2. Real-time object detection and tracking with SORT algorithm
3. Adaptive traffic signal control based on traffic density
4. Emergency vehicle detection and priority routing
5. Automatic violation detection and e-challan generation
6. Lane detection and vehicle assignment
7. Audio-based safety monitoring