Road Safety Infrastructure Analysis System

Complete Documentation

National Road Safety Hackathon 2025
Team: CHIKKUZ
Members: Ananya Rai, Sumant Kumar Giri

Generated: November 29, 2025

Table of Contents

Section 1: README

Source: README.md

Road Safety Infrastructure Analysis System

National Road Safety Hackathon 2025

An AI-powered system for comparative analysis of road infrastructure elements using computer vision and deep learning.

■ Features

• **AI-Based Video Analysis**: Compare base vs present road videos to detect deterioration
• **Multi-Element Detection**: Identify pavement condition, road markings, road studs, signs, roadside furniture, and VRU paths
• **Geospatial Visualization**: Map-based dashboard with GPS-tagged detections
• **Automated Reporting**: Generate PDF reports with before-after comparisons
• **Real-time Processing**: Fast API backend for video processing and inference
• **Interactive Dashboard**: Web interface for playback, analysis, and validation
• **Database Integration**: Supabase PostgreSQL database for scalable data storage

■■ System Architecture

```
■■■■■■■■■■■■■■■■■■■■■ ■ Input Videos ■ (Base + Present) ■ + Metadata ■ (GPS,
Timestamps) ■■■■■■■■■■■■■■■■■■■■■ ■ ▼ ■■■■■■■■■■■■■■■■■■■■■ ■ Data Pipeline ■
(Frame Extraction, Alignment) ■■■■■■■■■■■■■■■■■■■■■ ■ ▼ ■■■■■■■■■■■■■■■■■■■■■ ■
AI Processing ■ (YOLOv8, Change Detection) ■■■■■■■■■■■■■■■■■■■■■ ■ ▼
■■■■■■■■■■■■■■■■■■■■■ ■ Aggregation ■ (GPS Segmentation, Scoring)
■■■■■■■■■■■■■■■■■■■■■ ■ ▼ ■■■■■■■■■■■■■■■■■■■■■ ■ Visualization ■ (Dashboard,
Maps, Reports) ■■■■■■■■■■■■■■■■■■■■■
```

■ Project Structure

```
RSH/ ■■■ backend/ # FastAPI backend ■ ■■■ api/ # API routes ■ ■■■ models/ # Database
models (Supabase) ■ ■■■ services/ # Business logic ■ ■■■ main.py # FastAPI app entry
■■■ ai_models/ # AI model scripts ■ ■■■ detection/ # Object detection models ■ ■■■
change_detection/ # Change detection logic ■ ■■■ training/ # Model training scripts
■■■ data_processing/ # Video processing pipeline ■ ■■■ frame_extraction.py ■ ■■■
alignment.py ■ ■■■ preprocessing.py ■■■ database/ # Database schema and setup ■ ■■■
schema.sql # Supabase database schema ■ ■■■ README.md # Database setup guide ■■■
```

```
frontend/ # React/Next.js dashboard ■ ■■■ components/ ■ ■■■ pages/ ■ ■■■ public/ ■■■
config/ # Configuration files ■■■ requirements.txt # Python dependencies ■■■ README.md
```

■ Quick Start

Prerequisites

- Python 3.9+
- Node.js 16+
- CUDA-capable GPU (optional, for faster inference)

Installation

- **Clone and setup Python environment:**

```
cd RSH python -m venv venv source venv/bin/activate # On Windows: venv\Scripts\activate
pip install -r requirements.txt
```

- **Setup frontend:**

```
cd frontend npm install
```

- **Download AI models:**

```
python scripts/download_models.py
```

- **Setup Supabase Database (Optional but Recommended):**

```
# Create .env file with Supabase credentials # See SUPABASE_SETUP.md for detailed
instructions # Run database schema # Copy contents of database/schema.sql to Supabase
SQL Editor
```

Running the Application

- **Start backend server:**

```
cd backend uvicorn main:app --reload --port 8000
```

- **Start frontend:**

```
cd frontend npm run dev
```

- **Access dashboard:**

Open http://localhost:3000 in your browser

■ Configuration

Edit `config/config.yaml` to customize:
- Model paths
- Detection thresholds

• GPS coordinate systems
• Report templates

■ Usage

Upload Videos

• Upload base (reference) video
• Upload present (current) video
• Provide GPS metadata (optional, can be extracted from video)

Run Analysis

• Select road segments to analyze
• Click "Run Analysis"
• View results in dashboard:
• Map visualization with detected issues
• Before-after comparison frames
• Severity scores and trends

Generate Reports

• Select analysis results
• Click "Generate Report"
• Download PDF with:
• Summary statistics
• Issue heatmaps
• Detailed findings

■ Technical Details

AI Models

• **Object Detection**: YOLOv8 (custom trained on road infrastructure)
• **Segmentation**: DeepLabV3+ for pixel-level analysis
• **Change Detection**: Siamese network for temporal comparison

Detection Elements

• **Pavement Condition**: Cracks, potholes, surface degradation
• **Road Markings**: Faded lines, missing markings
• **Road Studs**: Missing or damaged reflectors
• **Signs**: Damaged, missing, or obscured signs
• **Roadside Furniture**: Barriers, guardrails, lighting
• **VRU Paths**: Pedestrian crossings, bike lanes

Evaluation Metrics

• **Detection Accuracy**: mAP@0.5 for object detection
• **Classification Accuracy**: F1-score for deterioration classification
• **Geospatial Precision**: GPS alignment accuracy
• **Processing Speed**: FPS for video analysis

■ Innovation Highlights

• **Temporal Comparison**: AI-driven before-after analysis
• **Geospatial Intelligence**: GPS-tagged detections with map visualization
• **Automated Scoring**: Severity-based issue prioritization
• **Auditor Feedback Loop**: Validation mechanism for continuous improvement
• **Real-time Processing**: Optimized pipeline for quick insights

■ Software & Technologies Utilized

Programming Languages

• **Python 3.9+**: Backend development, AI/ML processing, data analysis
• **JavaScript (ES6+)**: Frontend development, React components
• **SQL**: Database schema and queries (PostgreSQL)

Backend Framework & Libraries

• **FastAPI 0.104.1**: Modern, fast web framework for building APIs
• **Uvicorn**: ASGI server for running FastAPI applications
• **Pydantic**: Data validation and settings management
• **Python-multipart**: File upload handling

AI/ML & Computer Vision

• **PyTorch 2.1.0**: Deep learning framework
• **Torchvision 0.16.0**: Computer vision utilities
• **Ultralytics YOLOv8 8.1.0**: Object detection model
• **OpenCV 4.8.1**: Computer vision and image processing

• **NumPy 1.24.3**: Numerical computing
• **Pillow 10.1.0**: Image processing
• **scikit-learn 1.3.2**: Machine learning utilities

Data Processing & Analysis

• **Pandas 2.1.3**: Data manipulation and analysis
• **SciPy 1.11.4**: Scientific computing
• **imageio 2.31.6**: Image I/O operations
• **imageio-ffmpeg 0.4.9**: Video processing

Geospatial Libraries

• **geopy 2.4.1**: Geocoding and distance calculations
• **folium 0.15.1**: Interactive map generation
• **geopandas 0.14.1**: Geospatial data operations
• **shapely 2.0.2**: Geometric operations

Visualization

• **Matplotlib 3.8.2**: Static plotting
• **Seaborn 0.13.0**: Statistical data visualization
• **Plotly 5.18.0**: Interactive visualizations

Report Generation

• **ReportLab 4.0.7**: PDF generation
• **FPDF2 2.7.6**: PDF creation library
• **Jinja2 3.1.2**: Template engine

Database & ORM

• **Supabase 2.3.0**: PostgreSQL database with real-time capabilities
• **PostgREST 0.13.0**: REST API for PostgreSQL
• **SQLAlchemy 2.0.23**: SQL toolkit and ORM
• **Alembic 1.12.1**: Database migration tool

Frontend Framework & Libraries

• **Next.js 14.2.33**: React framework for production
• **React 18.2.0**: UI library
• **React DOM 18.2.0**: React rendering
• **Axios 1.13.2**: HTTP client for API calls

Frontend UI & Visualization

• **Leaflet 1.9.4**: Interactive maps
• **React-Leaflet 4.2.1**: React components for Leaflet
• **Recharts 2.10.3**: Chart library for React
• **Lucide React 0.294.0**: Icon library
• **Tailwind CSS 3.3.6**: Utility-first CSS framework

Frontend Development Tools

• **TypeScript 5.3.3**: Type checking (dev dependency)
• **ESLint 8.56.0**: Code linting
• **PostCSS 8.4.32**: CSS processing
• **Autoprefixer 10.4.16**: CSS vendor prefixing

Utilities & Configuration

• **python-dotenv 1.0.0**: Environment variable management
• **PyYAML 6.0.1**: YAML file parsing
• **aiofiles 23.2.1**: Async file operations
• **python-jose**: JWT token handling

Testing

• **pytest 7.4.3**: Python testing framework
• **pytest-asyncio 0.21.1**: Async testing support

Development Tools & Platforms

• **Git**: Version control
• **Node.js 16+**: JavaScript runtime
• **npm**: Node package manager
• **pip**: Python package manager
• **Virtual Environment (venv)**: Python environment isolation
• **Supabase Cloud**: Database hosting and management
• **CUDA** (Optional): GPU acceleration for AI inference

IDEs & Editors

• **VS Code / Cursor**: Code editor
• **PowerShell**: Command-line interface (Windows)

API Documentation

• **Swagger UI**: Interactive API documentation (auto-generated by FastAPI)
• **ReDoc**: Alternative API documentation

■■ Database Setup (Supabase)

The system uses Supabase (PostgreSQL) for data storage. Setup is optional - the system will fall back to file-based storage if Supabase is not configured.

Quick Setup:
• Create a Supabase project at https://supabase.com
• Get your `SUPABASE_URL` and `SUPABASE_KEY` from project settings
• Create `.env` file with credentials
• Run `database/schema.sql` in Supabase SQL Editor
• Test connection: `python scripts/test_database.py`

For detailed instructions, see SUPABASE_SETUP.md and database/README.md.

■ API Documentation

Once the server is running, visit:
• Swagger UI: http://localhost:8000/docs
• ReDoc: http://localhost:8000/redoc

■ Contributing

This is a hackathon project. For improvements:
• Fork the repository
• Create feature branch
• Submit pull request

■ License

MIT License - Hackathon Project

■ Team

National Road Safety Hackathon 2025 Team: CHIKKUZ
Members:
• Ananya Rai
• Sumant Kumar Giri

Built with ❤■ for Road Safety

Section 2: workingExp

Source: workingExp.md

Working Experience - Road Safety Infrastructure Analysis System

Project Overview

The **Road Safety Infrastructure Analysis System** is an AI-powered platform designed for comparative analysis of road infrastructure elements using computer vision and deep learning. The system compares base (reference) and present (current) road videos to detect deterioration, damage, and changes in road infrastructure.

System Architecture

1. Backend Architecture (FastAPI)

The backend is built using FastAPI and follows a modular service-oriented architecture:

Core Components:

- **API Routes** (`backend/api/routes/`):
- `video.py`: Handles video upload, retrieval, and management
- `analysis.py`: Manages analysis job creation and status tracking
- `reports.py`: Generates PDF reports from analysis results
- `dashboard.py`: Provides dashboard data and visualizations

- **Services** (`backend/services/`):
- `video_service.py`: Manages video storage and metadata
- `analysis_service.py`: Orchestrates the AI analysis pipeline
- `report_service.py`: Generates comprehensive PDF reports
- `dashboard_service.py`: Aggregates data for dashboard visualization
- `config_manager.py`: Centralized configuration management

Main Application (`backend/main.py`):

- FastAPI application with CORS middleware
- Modular router inclusion for clean API structure
- Health check endpoints

• Configuration-based initialization

2. AI/ML Pipeline

Detection System (`ai_models/detection/detector.py`):

• **YOLOv8 Model**: Uses Ultralytics YOLOv8 for object detection
• **Detection Elements**:
• Pavement cracks and potholes
• Faded road markings
• Missing road studs
• Damaged signs
• Roadside furniture damage
• VRU (Vulnerable Road User) path obstructions
• **Severity Classification**: Heuristic-based classification (minor, moderate, severe)
• **Confidence Thresholding**: Configurable confidence and IoU thresholds

Change Detection (`ai_models/change_detection/change_detector.py`):

• Compares detections between base and present videos
• Identifies temporal changes and deterioration
• Calculates severity scores for detected issues

3. Data Processing Pipeline

Frame Extraction (`data_processing/frame_extraction.py`):

• Extracts frames from video at configurable FPS
• Handles multiple video formats (MP4, AVI, MOV, MKV)
• Optimizes frame extraction for processing efficiency

Frame Alignment (`data_processing/alignment.py`):

• Aligns frames from base and present videos
• GPS-based alignment using metadata
• Temporal alignment for synchronized comparison
• Handles coordinate system transformations

Preprocessing (`data_processing/preprocessing.py`):

• Frame resizing and normalization
• Video stabilization
• Quality enhancement for better detection accuracy

4. Analysis Workflow

The analysis process follows these steps:

• **Video Upload**:
• User uploads base (reference) and present (current) videos
• Optional GPS metadata can be attached
• Videos are stored with unique IDs

• **Frame Extraction** (10% progress):
• Frames extracted from both videos at configured FPS
• Maximum frame limit to prevent excessive processing

• **Frame Alignment** (30% progress):
• Frames aligned using GPS coordinates and timestamps
• Creates matched pairs for comparison

• **Object Detection** (50% progress):
• YOLOv8 model processes each frame
• Detects road infrastructure elements
• Classifies severity of detected issues

• **Change Detection** (50-80% progress):
• Compares detections between base and present frames
• Identifies new issues, deterioration, or improvements
• Calculates change scores

• **Result Aggregation** (80% progress):
• Groups detections by road segments
• Aggregates by element type and severity
• Generates summary statistics

• **Result Storage** (100% progress):
• Saves results as JSON files
• Updates status tracking
• Makes results available for visualization

5. Frontend Architecture (Next.js/React)

Components:

• `VideoUpload.js`: Drag-and-drop video upload interface
• `AnalysisDashboard.js`: Real-time analysis status and results display
• `MapVisualization.js`: Interactive map with GPS-tagged detections

Features:

• Real-time progress tracking
• Interactive map visualization
• Before-after frame comparison
• Heatmap visualization of issues
• Report generation interface

6. Geospatial Features

• **GPS Tagging**: All detections are tagged with GPS coordinates
• **Map Visualization**: Interactive maps showing issue locations
• **Heatmap Generation**: Visual representation of issue density
• **Road Segment Analysis**: Aggregation by geographic segments
• **Coordinate System Support**: WGS84 (EPSG:4326) with configurable systems

7. Reporting System

• **PDF Report Generation**: Comprehensive reports with:
• Summary statistics
• Issue heatmaps
• Before-after comparisons
• Detailed findings per element type
• Severity breakdowns
• **Configurable Templates**: Customizable report formats
• **Export Options**: Multiple output formats

Technical Implementation Details

Configuration Management

The system uses YAML-based configuration (`config/config.yaml`):
• Model parameters (confidence thresholds, input sizes)
• Detection class definitions with severity weights
• Video processing settings (FPS, frame limits)
• Geospatial settings (coordinate systems, map providers)
• API settings (CORS, upload limits)
• Storage paths

Data Storage

Current Implementation (File-based):
• Videos stored in `uploads/` directory
• Metadata stored in JSON files
• Analysis results in `processed/` directory
• Status tracking via JSON files

Supabase Integration (New):
• Structured database for videos, analyses, and results
• Real-time subscriptions for status updates
• Efficient querying and filtering
• Scalable storage solution

API Endpoints

Video Management:

- `POST /api/v1/video/upload`: Upload video files
- `GET /api/v1/video/{video_id}`: Get video information
- `GET /api/v1/video/list`: List all videos
- `DELETE /api/v1/video/{video_id}`: Delete video

Analysis:

- `POST /api/v1/analysis/run`: Start analysis job
- `GET /api/v1/analysis/{analysis_id}/status`: Get analysis status
- `GET /api/v1/analysis/{analysis_id}/results`: Get analysis results
- `GET /api/v1/analysis/{analysis_id}/detections`: Get all detections
- `GET /api/v1/analysis/{analysis_id}/heatmap`: Get heatmap data

Dashboard:

- `GET /api/v1/dashboard/summary`: Get dashboard summary
- `GET /api/v1/dashboard/stats`: Get statistics

Reports:

- `POST /api/v1/reports/generate`: Generate PDF report
- `GET /api/v1/reports/{report_id}`: Download report

Key Features & Innovations

1. Temporal Comparison

- AI-driven before-after analysis
- Identifies deterioration over time
- Quantifies changes with severity scores

2. Multi-Element Detection

- Simultaneous detection of 6+ infrastructure elements
- Custom-trained or adapted models for road infrastructure
- Configurable detection thresholds

3. Geospatial Intelligence

- GPS-tagged detections
- Map-based visualization
- Road segment-based aggregation
- Spatial querying capabilities

4. Automated Scoring

• Severity-based issue prioritization
• Weighted scoring system
• Configurable severity thresholds

5. Real-time Processing

• Background task processing
• Progress tracking via status updates
• Asynchronous API design
• Optimized for performance

6. Scalable Architecture

• Modular service design
• Easy to extend with new detection types
• Configurable processing parameters
• Database-backed for production scale

Performance Optimizations

• **Batch Processing**: Configurable batch sizes for model inference
• **GPU Support**: CUDA acceleration for faster processing
• **Frame Sampling**: Configurable FPS to balance accuracy and speed
• **Caching**: Intermediate results cached for faster retrieval
• **Async Operations**: Non-blocking I/O for better throughput

Technology Stack

Backend:

• **FastAPI**: Modern, fast web framework
• **Python 3.9+**: Core language
• **YOLOv8 (Ultralytics)**: Object detection
• **OpenCV**: Computer vision operations
• **PyTorch**: Deep learning framework

Frontend:

• **Next.js**: React framework

• **React**: UI library
• **Tailwind CSS**: Styling
• **Map Libraries**: Geospatial visualization

Database:

• **Supabase**: PostgreSQL database with real-time capabilities
• **SQLAlchemy**: ORM for database operations

Infrastructure:

• **Docker** (optional): Containerization
• **Cloud Storage**: Scalable file storage
• **CDN**: Content delivery for static assets

Development Workflow

• **Setup**:
• Install Python dependencies
• Setup Supabase project
• Configure environment variables
• Download AI models

• **Development**:
• Backend: `uvicorn backend.main:app --reload`
• Frontend: `npm run dev`
• Database migrations: `alembic upgrade head`

• **Testing**:
• Unit tests for services
• Integration tests for API endpoints
• Model validation tests

• **Deployment**:
• Environment configuration
• Database migration
• Model deployment
• Static asset optimization

Future Enhancements

• **Custom Model Training**: Train YOLOv8 on road infrastructure dataset
• **Real-time Video Streaming**: Process live video feeds
• **Mobile App**: Native mobile application for field data collection
• **Advanced Analytics**: Trend analysis, predictive maintenance
• **Integration**: Connect with road maintenance management systems
• **Multi-user Support**: User authentication and role-based access

• **Audit Trail**: Complete history of analyses and changes

Challenges & Solutions

Challenge 1: Frame Alignment

Problem: Aligning frames from different videos with varying GPS data
Solution: Multi-criteria alignment using GPS coordinates, timestamps, and visual features

Challenge 2: Processing Speed

Problem: Large video files require significant processing time
Solution: Frame sampling, batch processing, GPU acceleration, and background tasks

Challenge 3: Detection Accuracy

Problem: General-purpose models may not detect road-specific elements accurately
Solution: Custom class mapping, fine-tuning, and severity classification heuristics

Challenge 4: Scalability

Problem: File-based storage doesn't scale well
Solution: Supabase integration for structured, scalable data storage

Conclusion

The Road Safety Infrastructure Analysis System demonstrates a comprehensive approach to automated road infrastructure monitoring. By combining computer vision, deep learning, and geospatial analysis, the system provides actionable insights for road maintenance and safety improvements. The modular architecture ensures maintainability and extensibility for future enhancements.

Team: CHIKKUZ
Members: Ananya Rai, Sumant Kumar Giri
Project: National Road Safety Hackathon 2025

Section 3: QUICK_START

Source: QUICK_START.md

Quick Start Guide - How to Run the Project

Prerequisites Check

First, verify you have the required tools:

```
# Check Python version (need 3.9+) python --version # Check Node.js version (need 16+)
node --version # Check npm npm --version
```

Step-by-Step Setup

Step 1: Create Python Virtual Environment

Open PowerShell in the project directory (`C:\Users\suman\Documents\RSH`):

```
# Create virtual environment python -m venv venv # Activate virtual environment
.\venv\Scripts\Activate.ps1 # If you get an execution policy error, run this first: #
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

You should see `(venv)` at the start of your command prompt.

Step 2: Install Python Dependencies

```
# Make sure venv is activated (you should see (venv)) pip install -r requirements.txt
```

This will install all required packages (FastAPI, PyTorch, YOLOv8, etc.)

Step 3: Create Required Directories

```
# Create directories for uploads, results, etc. New-Item -ItemType Directory -Force
-Path uploads, processed, cache, reports, models
```

Step 4: Download AI Models

```
# Download YOLOv8 model (this may take a few minutes) python scripts/download_models.py
```

Step 5: Setup Supabase Database (Optional but Recommended)

The system works with file-based storage by default, but Supabase provides better scalability and features.

Quick Setup:

- **Create Supabase Project:**
- Go to https://app.supabase.com
- Click "New Project"
- Fill in project details and wait for creation

- **Get API Credentials:**
- In Supabase dashboard: **Settings** → **API**
- Copy **Project URL** (SUPABASE_URL)
- Copy **anon public key** (SUPABASE_KEY)

- **Create Database Tables:**
- In Supabase dashboard: **SQL Editor** → **New Query**
- Copy contents of `database/schema.sql`
- Paste and click **Run**

- **Configure Environment:**

```
# Create .env file in project root # Add your Supabase credentials:
SUPABASE_URL=https://your-project-id.supabase.co SUPABASE_KEY=your-anon-key-here
```

- **Test Connection:**

```
python scripts/test_database.py
```

Note: If you skip this step, the system will use file-based storage automatically. You can set up Supabase later without any code changes.

For detailed instructions, see SUPABASE_SETUP.md.

Step 6: Install Frontend Dependencies

Open a **new PowerShell window** (keep the backend one open):

```
# Navigate to project directory cd C:\Users\suman\Documents\RSH # Navigate to frontend
cd frontend # Install Node.js dependencies npm install
```

This may take a few minutes.

Running the Application

You need **two terminals** running simultaneously:

Terminal 1: Backend Server

```
# Navigate to project root cd C:\Users\suman\Documents\RSH # Activate virtual
environment (if not already activated) .\venv\Scripts\Activate.ps1 # Run backend server
(from project root) uvicorn backend.main:app --reload --port 8000
```

You should see:

```
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit) INFO: Started
reloader process INFO: Started server process INFO: Waiting for application startup.
INFO: Application startup complete.
```

Backend API is now running at: http://localhost:8000
API Documentation: http://localhost:8000/docs

Terminal 2: Frontend Server

```
# Navigate to frontend directory cd C:\Users\suman\Documents\RSH\frontend # Run
frontend development server npm run dev
```

You should see:

```
ready - started server on 0.0.0.0:3000, url: http://localhost:3000
```

Frontend is now running at: http://localhost:3000

Access the Application

• Open your web browser
• Navigate to: **http://localhost:3000**
• You should see the Road Safety Infrastructure Analysis dashboard

Alternative: Run Backend from Backend Directory

If you prefer to run from the backend directory:

```
cd backend python main.py
```

This also works because we fixed the import paths!

Testing the API

You can test the API directly:

• **Open API Docs:** http://localhost:8000/docs
• **Test Health Endpoint:** http://localhost:8000/health
• **Root Endpoint:** http://localhost:8000/

Testing Database Connection (If Supabase is Configured)

```
# Test Supabase connection python scripts/test_database.py
```

You should see:
• ■ Database connection successful (if configured correctly)
• ■■ Database not configured (system will use file-based storage)

Troubleshooting

Backend Issues

Problem: ModuleNotFoundError

```
# Make sure you're in the project root and venv is activated cd
C:\Users\suman\Documents\RSH .\venv\Scripts\Activate.ps1
```

Problem: Port 8000 already in use

```
# Find and kill the process using port 8000 netstat -ano | findstr :8000 taskkill /PID
<PID_NUMBER> /F # Or change the port in backend/main.py
```

Problem: Config file not found
• Make sure `config/config.yaml` exists
• Check you're running from project root

Problem: Database connection fails
• Check `.env` file has correct `SUPABASE_URL` and `SUPABASE_KEY`
• Verify tables exist in Supabase (run `database/schema.sql`)
• System will automatically use file-based storage as fallback

Frontend Issues

Problem: npm install fails

```
# Clear npm cache and try again npm cache clean --force npm install
```

Problem: Cannot connect to API
• Make sure backend is running on port 8000
• Check `frontend/next.config.js` has correct API URL
• Check browser console for CORS errors

Problem: Execution Policy Error (PowerShell)

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Quick Commands Reference

```
# Activate virtual environment .\venv\Scripts\Activate.ps1 # Run backend (from project
root) uvicorn backend.main:app --reload --port 8000 # Run backend (from backend
directory) cd backend python main.py # Run frontend cd frontend npm run dev # Deactivate
virtual environment (when done) deactivate
```

Next Steps

• ■ Backend and frontend are running
• ■ (Optional) Supabase database configured
• Open http://localhost:3000 in your browser
• Upload test videos (base and present)
• Run analysis
• View results in the dashboard

Storage Options:
• **With Supabase:** All data stored in database, scalable and efficient
• **Without Supabase:** Data stored in local files (uploads/, processed/), works out-of-the-box

Full Command Sequence (Copy-Paste)

```
# Terminal 1: Backend cd C:\Users\suman\Documents\RSH python -m venv venv
.\venv\Scripts\Activate.ps1 pip install -r requirements.txt python
scripts/download_models.py # Optional: Test database connection (if Supabase is
configured) python scripts/test_database.py uvicorn backend.main:app --reload --port
8000 # Terminal 2: Frontend (in new PowerShell window) cd
C:\Users\suman\Documents\RSH\frontend npm install npm run dev
```

Then open http://localhost:3000 in your browser!

Note: If you haven't set up Supabase, the system will automatically use file-based storage. No additional configuration needed!

Download sample videos (optional)

```
# From project root python scripts/download_sample_videos.py
```

This creates sample files in `uploads/samples/`:
• `base_road_sample.mp4`
• `present_road_sample.mp4`
• `base_gps.json` and `present_gps.json` (optional)

Use these via the web uploader as Base and Present to test the pipeline.

Section 4: SETUP

Source: SETUP.md

Setup Guide

Prerequisites

- Python 3.9 or higher
- Node.js 16 or higher
- pip (Python package manager)
- npm or yarn (Node package manager)

Backend Setup

- **Create virtual environment:**

```
python -m venv venv
```

- **Activate virtual environment:**
- Windows: `venv\Scripts\activate`
- Linux/Mac: `source venv/bin/activate`
- **Install Python dependencies:**

```
pip install -r requirements.txt
```

- **Download AI models:**

```
python scripts/download_models.py
```

- **Create necessary directories:**

```
mkdir -p uploads processed cache reports models
```

- **Start backend server:**

```
cd backend uvicorn main:app --reload --port 8000
```

The API will be available at `http://localhost:8000`
API documentation: `http://localhost:8000/docs`

Frontend Setup

- **Navigate to frontend directory:**

```
cd frontend
```

• **Install dependencies:**

```
npm install
```

• **Create environment file (optional):**

```
cp ../.env.example .env.local # Edit .env.local if needed
```

• **Start development server:**

```
npm run dev
```

The frontend will be available at `http://localhost:3000`

Running the Full Application

• Start the backend server (in one terminal):

```
cd backend uvicorn main:app --reload --port 8000
```

• Start the frontend server (in another terminal):

```
cd frontend npm run dev
```

• Open your browser and navigate to `http://localhost:3000`

Project Structure

```
RSH/ ■■■ backend/ # FastAPI backend ■ ■■■ api/ # API routes ■ ■■■ services/ #
Business logic ■ ■■■ main.py # FastAPI app entry ■■■ ai_models/ # AI model scripts ■
■■■ detection/ # Object detection ■ ■■■ change_detection/ # Change detection ■■■
data_processing/ # Video processing ■ ■■■ frame_extraction.py ■ ■■■ alignment.py ■■■
frontend/ # Next.js frontend ■ ■■■ components/ # React components ■ ■■■ pages/ #
Next.js pages ■ ■■■ styles/ # CSS files ■■■ config/ # Configuration files ■■■
scripts/ # Utility scripts ■■■ requirements.txt # Python dependencies
```

Troubleshooting

Backend Issues

• **Import errors**: Make sure you're running from the project root or backend directory
• **Model not found**: Run `python scripts/download_models.py` to download models
• **Port already in use**: Change port in `backend/main.py` or kill the process using port 8000

Frontend Issues

• **Module not found**: Run `npm install` again
• **API connection error**: Ensure backend is running on port 8000
• **Leaflet map not showing**: Check browser console for CORS or API errors

Development Notes

• The backend uses FastAPI with automatic API documentation at `/docs`
• The frontend uses Next.js with React
• Videos are stored in `uploads/` directory
• Analysis results are stored in `processed/` directory
• Reports are generated in `reports/` directory

Next Steps

• Train custom YOLOv8 model on road infrastructure dataset
• Implement database for persistent storage
• Add user authentication
• Deploy to cloud platform (AWS, GCP, Azure)
• Add more visualization features

Section 5: SUPABASE_SETUP

Source: SUPABASE_SETUP.md

Supabase Database Setup

This guide will help you set up Supabase database for the Road Safety Infrastructure Analysis System.

Quick Setup

1. Create Supabase Project

• Go to https://app.supabase.com and sign up/login
• Click "New Project"
• Fill in:
• **Name**: Road Safety Analysis
• **Database Password**: (choose a strong password)
• **Region**: (select closest to you)
• Wait 2-3 minutes for project creation

2. Get API Credentials

• In Supabase dashboard, go to **Settings** → **API**
• Copy:
• **Project URL** → This is your `SUPABASE_URL`
• **anon public key** → This is your `SUPABASE_KEY`

3. Create Database Tables

• In Supabase dashboard, go to **SQL Editor**
• Click **New Query**
• Copy the entire contents of `database/schema.sql`
• Paste and click **Run** (or press Ctrl+Enter)
• Verify tables are created in **Table Editor**

4. Configure Environment

Create a `.env` file in the project root:

```
# Supabase Configuration SUPABASE_URL=https://your-project-id.supabase.co
SUPABASE_KEY=your-anon-key-here # Optional API_HOST=0.0.0.0 API_PORT=8000
```

5. Install Dependencies

```
pip install -r requirements.txt
```

6. Test Connection

The system will automatically use Supabase if credentials are configured. If not, it falls back to file-based storage.

Database Schema

The database includes 5 tables:

• **videos**: Stores uploaded video metadata
• **analyses**: Tracks analysis jobs and status
• **detections**: Stores detected road infrastructure issues
• **reports**: Stores generated PDF reports
• **road_segments**: Defines road segments for analysis

See `database/schema.sql` for full schema details.

Features

✓ **Automatic Fallback**: If Supabase is not configured, the system uses file-based storage
✓ **Real-time Updates**: Status updates are stored in database
✓ **Efficient Queries**: Indexed tables for fast lookups
✓ **Scalable**: Handles large volumes of data

Troubleshooting

Connection Failed?
• Check `SUPABASE_URL` and `SUPABASE_KEY` in `.env`
• Verify tables exist in Supabase dashboard
• Check network/firewall settings

Tables Not Found?
• Run `database/schema.sql` in SQL Editor
• Check for errors in SQL execution

Data Not Saving?
• Check Supabase logs in dashboard

• Verify RLS policies if enabled
• Check application logs for errors

Next Steps

• Configure Row Level Security (RLS) for production
• Set up authentication if needed
• Configure backups
• Monitor usage in Supabase dashboard

For more details, see `database/README.md`.

Section 6: README

Source: database/README.md

Database Setup Guide

This guide explains how to set up the Supabase database for the Road Safety Infrastructure Analysis System.

Prerequisites

• A Supabase account (sign up at https://supabase.com)
• A new Supabase project created

Setup Steps

1. Create Supabase Project

• Go to https://app.supabase.com
• Click "New Project"
• Fill in project details:
• Name: Road Safety Analysis (or your preferred name)
• Database Password: Choose a strong password
• Region: Select closest region
• Wait for project to be created (takes a few minutes)

2. Get API Credentials

• In your Supabase project dashboard, go to **Settings** → **API**
• Copy the following values:
• **Project URL** (SUPABASE_URL)
• **anon/public key** (SUPABASE_KEY)
• Save these for later use

3. Create Database Tables

• In your Supabase project, go to **SQL Editor**
• Click "New Query"

• Copy and paste the contents of `database/schema.sql`
• Click "Run" to execute the SQL
• Verify tables are created by checking **Table Editor**

4. Configure Environment Variables

• Copy `.env.example` to `.env`:

```
cp .env.example .env
```

• Edit `.env` and add your Supabase credentials:

```
SUPABASE_URL=https://your-project-id.supabase.co SUPABASE_KEY=your-anon-key-here
```

5. Install Python Dependencies

Make sure you have the Supabase Python client installed:

```
pip install -r requirements.txt
```

6. Verify Connection

Run a test script to verify the connection:

```
from backend.services.database_service import get_database_service db =
get_database_service() if db: print("■ Database connection successful!") else: print("■
Database connection failed. Check your credentials.")
```

Database Schema

The database consists of 5 main tables:

1. `videos`

Stores uploaded video files and metadata
• `id`: Unique identifier (UUID)
• `filename`: Original filename
• `video_type`: 'base' or 'present'
• `file_path`: Path to video file
• `file_size`: File size in bytes
• `gps_data`: Optional GPS metadata (JSONB)
• `upload_date`: Upload timestamp

2. `analyses`

Stores analysis jobs and their status
• `id`: Unique identifier (UUID)

• `base_video_id`: Reference to base video
• `present_video_id`: Reference to present video
• `status`: 'pending', 'processing', 'completed', 'failed'
• `progress`: Progress percentage (0-100)
• `error`: Error message if failed
• `summary`: Analysis summary (JSONB)

3. `detections`

Stores detected road infrastructure issues
• `id`: Unique identifier (UUID)
• `analysis_id`: Reference to analysis
• `element_type`: Type of detected element
• `bbox`: Bounding box coordinates [x1, y1, x2, y2]
• `confidence`: Detection confidence (0.0-1.0)
• `severity`: 'minor', 'moderate', 'severe'
• `severity_score`: Numeric severity score (0.0-10.0)
• `gps_coords`: GPS coordinates [lat, lon]
• `frame_number`: Frame number in video

4. `reports`

Stores generated PDF reports
• `id`: Unique identifier (UUID)
• `analysis_id`: Reference to analysis
• `file_path`: Path to report file
• `file_size`: File size in bytes
• `generated_at`: Generation timestamp

5. `road_segments`

Stores road segment definitions
• `id`: Unique identifier (UUID)
• `name`: Segment name
• `gps_coords`: Polygon coordinates [[lat, lon], ...]
• `created_at`, `updated_at`: Timestamps

Indexes

The schema includes indexes for:
• Video type and creation date
• Analysis status and video references
• Detection analysis ID, element type, severity, and GPS coordinates
• Report analysis ID and generation date

Row Level Security (RLS)

RLS is disabled by default. To enable it:

• Uncomment RLS lines in `schema.sql`
• Create appropriate policies based on your authentication needs
• Configure authentication in your application

Migration from File-based Storage

If you're migrating from file-based storage:

• The system will automatically use Supabase if credentials are configured
• Existing file-based data won't be automatically migrated
• You can create a migration script to import existing data

Troubleshooting

Connection Errors

• **"Invalid API key"**: Check that SUPABASE_KEY is correct
• **"Invalid URL"**: Verify SUPABASE_URL format
• **"Connection timeout"**: Check network/firewall settings

Table Errors

• **"Table does not exist"**: Run `schema.sql` in SQL Editor
• **"Permission denied"**: Check RLS policies if enabled

Data Type Errors

• **"Invalid enum value"**: Check that enum values match schema
• **"Array length mismatch"**: Verify bbox and gps_coords formats

Best Practices

• **Backup**: Regularly backup your database
• **Indexes**: Monitor query performance and add indexes as needed
• **RLS**: Enable RLS for production deployments
• **Monitoring**: Use Supabase dashboard to monitor usage

• **Connection Pooling**: Configure connection pooling for production

Support

For issues:

• Check Supabase documentation: https://supabase.com/docs
• Review error messages in application logs
• Check Supabase project logs in dashboard

Section 7: CHANGES_SUMMARY

Source: CHANGES_SUMMARY.md

Changes Summary

Created Files

1. `workingExp.md`

Comprehensive documentation summarizing how the Road Safety Infrastructure Analysis System works, including:
• System architecture overview
• Backend, AI/ML pipeline, and frontend details
• Analysis workflow
• Technical implementation details
• Performance optimizations
• Technology stack

2. Supabase Database Integration

Database Models (`backend/models/database.py`)

• Pydantic models for all database entities
• Enums for video types, analysis status, severity, and element types
• Type-safe database operations

Database Service (`backend/services/database_service.py`)

• Complete Supabase client integration
• CRUD operations for:
• Videos
• Analyses
• Detections
• Reports
• Road Segments
• Automatic connection management
• Error handling with fallback support

Database Schema (`database/schema.sql`)

• Complete PostgreSQL schema for Supabase
• 5 main tables with proper relationships
• Indexes for performance optimization
• Triggers for automatic timestamp updates
• Row Level Security (RLS) ready (commented out)

Documentation

• `database/README.md`: Comprehensive database setup guide
• `SUPABASE_SETUP.md`: Quick setup guide
• `scripts/test_database.py`: Connection test script

3. Updated Services

`backend/services/video_service.py`

• Integrated Supabase support
• Automatic fallback to file-based storage
• Database-first approach with file fallback

`backend/services/analysis_service.py`

• Supabase integration for analysis tracking
• Database storage for detections
• Status updates in database
• Bulk detection insertion

4. Updated Dependencies

`requirements.txt`

• Added `supabase==2.3.0`
• Added `postgrest==0.13.0`

5. Configuration

Environment Variables

• `.env.example` template (blocked, but documented)
• Environment variable documentation in setup guides

Key Features

Automatic Fallback

The system automatically falls back to file-based storage if:
• Supabase credentials are not configured
• Database connection fails
• Tables don't exist

This ensures the system works out-of-the-box without requiring database setup.

Database-First Approach

When Supabase is configured:
• All data is stored in database
• Real-time status updates
• Efficient querying and filtering
• Scalable architecture

Type Safety

• Pydantic models ensure type safety
• Enum types prevent invalid values
• Validation at database level

Migration Path

From File-Based to Database

• **Setup Supabase**: Follow `SUPABASE_SETUP.md`
• **Create Tables**: Run `database/schema.sql`
• **Configure Environment**: Add credentials to `.env`
• **Test Connection**: Run `python scripts/test_database.py`
• **System Automatically Uses Database**: No code changes needed

Existing Data

Existing file-based data is not automatically migrated. To migrate:
• Export data from JSON files
• Create migration script (future enhancement)
• Import into Supabase

Testing

Test Database Connection

```
python scripts/test_database.py
```

Verify Tables

• Go to Supabase dashboard
• Check Table Editor
• Verify all 5 tables exist

Next Steps

• **Enable RLS**: Configure Row Level Security for production
• **Add Authentication**: Integrate Supabase Auth
• **Data Migration**: Create script to migrate existing data
• **Monitoring**: Set up database monitoring
• **Backups**: Configure automated backups

Files Modified

• `requirements.txt`: Added Supabase dependencies
• `backend/services/video_service.py`: Added database support
• `backend/services/analysis_service.py`: Added database support
• `README.md`: Added Supabase setup section

Files Created

• `workingExp.md`: Project working documentation
• `backend/models/__init__.py`: Models package init
• `backend/models/database.py`: Database models
• `backend/services/database_service.py`: Database service
• `database/schema.sql`: Database schema
• `database/README.md`: Database documentation
• `SUPABASE_SETUP.md`: Quick setup guide
• `scripts/test_database.py`: Connection test script

Compatibility

• ■ Backward compatible with file-based storage
• ■ Works without Supabase configuration
• ■ No breaking changes to existing API
• ■ All existing functionality preserved

Section 8: DIAGNOSTICS

Source: DIAGNOSTICS.md

Project Diagnostics Report

■ Files Present

Backend Structure

- ■ `backend/main.py` - FastAPI application entry point
- ■ `backend/api/routes/` - All route files (video, analysis, reports, dashboard)
- ■ `backend/services/` - All service files
- ■ `backend/api/__init__.py` - Package init file
- ■ `backend/services/__init__.py` - Package init file

AI Models

- ■ `ai_models/detection/detector.py` - Object detection
- ■ `ai_models/change_detection/change_detector.py` - Change detection
- ■ All `__init__.py` files present

Data Processing

- ■ `data_processing/frame_extraction.py` - Frame extraction
- ■ `data_processing/alignment.py` - Frame alignment
- ■■ `data_processing/preprocessing.py` - **MISSING** (mentioned in README)

Frontend

- ■ All React components present
- ■ Next.js configuration files
- ■ Package.json with all dependencies
- ■■ `tsconfig.json` or `jsconfig.json` - **MISSING** (needed for Next.js)
- ■■ `.env.local` or `.env` - **MISSING** (only .env.example exists)

■ Issues Found

1. CRITICAL: Python Import Path Issues

Problem:
• Backend files use absolute imports like `from services.config_manager` and `from data_processing.frame_extraction`
• These imports won't work when running from `backend/` directory
• Modules outside `backend/` can't be imported without proper PYTHONPATH

Files Affected:
• `backend/main.py` - imports `api.routes` and `services.config_manager`
• `backend/services/analysis_service.py` - imports `data_processing` and `ai_models`
• `ai_models/detection/detector.py` - imports `services.config_manager`
• `data_processing/frame_extraction.py` - imports `services.config_manager`

2. Missing: preprocessing.py

Problem:
• README mentions `data_processing/preprocessing.py` but file doesn't exist

3. Missing: Frontend Configuration Files

Problem:
• Next.js typically needs `jsconfig.json` or `tsconfig.json` for proper module resolution

■ Fixes Applied

See fixes below for resolved issues.

Section 9: FIXES_APPLIED

Source: FIXES_APPLIED.md

Fixes Applied - Project Diagnostics

Summary

Comprehensive diagnostics completed and all critical issues have been fixed.

■ Fixes Applied

1. CRITICAL: Fixed Python Import Path Issues

Problem: Absolute imports wouldn't work when running from different directories.

Solution: Added `sys.path` modifications to all Python files:
- ■ `backend/main.py` - Added project root to sys.path
- ■ `backend/services/*.py` - All service files fixed
- ■ `backend/api/routes/*.py` - All route files fixed
- ■ `data_processing/*.py` - All data processing files fixed
- ■ `ai_models/**/*.py` - All AI model files fixed

Changes:
- Changed imports from `from services.config_manager` to `from backend.services.config_manager`
- Added `sys.path.insert(0, project_root)` at the top of each file
- Ensures imports work regardless of where the script is run from

2. Created Missing Files

Created:
- ■ `data_processing/preprocessing.py` - Image preprocessing utilities
- Includes resize, normalize, stabilize functions
- Matches README documentation

- ■ `frontend/jsconfig.json` - Next.js configuration
- Proper path resolution for imports
- Better IDE support

- ■ `__init__.py` - Root package init file

• Makes project root a Python package

• ■ `setup.py` - Package setup file
• Allows installation as a package
• Proper dependency management

3. Fixed Import Statements

Updated imports in:
• All backend service files
• All API route files
• All data processing files
• All AI model files

Pattern used:

```
import sys from pathlib import Path # Add project root to Python path project_root =
Path(__file__).parent.parent.parent if str(project_root) not in sys.path:
sys.path.insert(0, str(project_root)) from backend.services.config_manager import
ConfigManager
```

4. Fixed Duplicate Imports

Removed duplicate imports in:
• `backend/services/dashboard_service.py`
• `backend/api/routes/video.py`
• `backend/api/routes/analysis.py`
• `backend/api/routes/reports.py`
• `backend/api/routes/dashboard.py`

■ Verification

Linting

• ■ No linter errors in backend
• ■ No linter errors in ai_models
• ■ No linter errors in data_processing

File Structure

• ■ All __init__.py files present
• ■ All referenced files exist
• ■ Configuration files present

■ Files Modified

Backend (13 files)

- `backend/main.py`
- `backend/services/analysis_service.py`
- `backend/services/video_service.py`
- `backend/services/report_service.py`
- `backend/services/dashboard_service.py`
- `backend/api/routes/video.py`
- `backend/api/routes/analysis.py`
- `backend/api/routes/reports.py`
- `backend/api/routes/dashboard.py`

Data Processing (2 files)

- `data_processing/frame_extraction.py`
- `data_processing/alignment.py`

AI Models (2 files)

- `ai_models/detection/detector.py`
- `ai_models/change_detection/change_detector.py`

■ Files Created

- `data_processing/preprocessing.py` - Image preprocessing utilities
- `frontend/jsconfig.json` - Next.js configuration
- `__init__.py` - Root package init
- `setup.py` - Package setup
- `DIAGNOSTICS.md` - Diagnostics report
- `FIXES_APPLIED.md` - This file

■ How to Run Now

Backend (from project root)

```
# Option 1: Run from project root (recommended) cd RSH uvicorn backend.main:app --reload
--port 8000 # Option 2: Run from backend directory (now works!) cd backend python
main.py
```

Frontend

```
cd frontend npm install npm run dev
```

■ Status: All Critical Issues Resolved

- ■ Import paths fixed
- ■ Missing files created
- ■ No linting errors
- ■ Project structure complete
- ■ Ready for development

■ Notes

- **Running Backend:** The backend can now be run from either the project root or the backend directory. The sys.path modifications ensure imports work correctly.

- **Package Installation:** You can now install the project as a package using `pip install -e .` from the project root.

- **Next Steps:**
- Install dependencies: `pip install -r requirements.txt`
- Download models: `python scripts/download_models.py`
- Start development!

Section 10: README

Source: samples/README.md

Samples

This folder documents sample assets for quick demos.

We do not commit video binaries. Use the download script to fetch small public sample videos.

Download

From project root:

```
# Windows PowerShell python scripts/download_sample_videos.py
```

This will create:
- `uploads/samples/base_road_sample.mp4`
- `uploads/samples/present_road_sample.mp4`
- `uploads/samples/base_gps.json`
- `uploads/samples/present_gps.json`

Use in App

- Upload `base_road_sample.mp4` as Base
- Upload `present_road_sample.mp4` as Present
- Optionally paste the contents of `base_gps.json` / `present_gps.json` into the GPS field (if you extend the UI to accept it), or rely on temporal alignment.