“Wrong-Way Vehicle Detection System”

By

Miss. Talatunnisa A. Siddiqui
Miss. Snehal R. Savandkar
Miss. Rasika R. Rakhewar
Miss. Shruti P. Pimple
Miss. Prajakta S. Waghmare

Under the Guidance

of

Dr. Manisha Y. Joshi
(Department of Computer Science and Engineering)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MGM’S COLLEGE OF ENGINEERING, NANDED (M.S.)

Academic Year 2025-26

A Project Report on

"Wrong-Way Vehicle Detection System"

submitted to

DR. BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY,LONERE

in partial fulfillment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

by

Miss. Talatunnisa A. Siddiqui

Miss. Snehal R. Savandkar

Miss. Rasika R. Rakhewar

Miss. Shruti P. Pimple

Miss. Prajakta S. Waghmare

Under the Guidance

of

Dr. Manisha Y. Joshi

(Department of Computer Science and Engineering)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MGM'S COLLEGE OF ENGINEERING, NANDED (M.S.)

Academic Year 2025-26

CERTIFICATE

This is to certify that the project phase report entitled

Wrong-Way Vehicle Detection System'

being submitted by ***Miss. Talatunnisa A. Siddiqui****,* ***Miss. Snehal R. Savandkar****,* ***Miss. Rasika R. Rakhewar****,* ***Miss. Shruti P. Pimple*** *and* ***Miss. Prajakta S. Waghmare*** *to the Dr. Babasaheb Ambedkar Technological University, Lonere, for the award of the degree of Bachelor of Engineering in Computer Science and Engineering, is a record of bonafide work carried out by them under my supervision and guidance. The matter contained in this report has not been submitted to any other university or institute for the award of any degree.*

Dr. Manisha Y. Joshi
Project Guide

Dr. A. M. Rajurkar
H.O.D
Computer Science
and Engineering

Dr. G. S. Lathkar
Director
MGM's College of
Engineering, Nanded

ACKNOWLEDGEMENT

We are greatly indebted to our project guide **Dr. Manisha Y. Joshi** for her able guidance throughout this work. It has been an altogether different experience to work with her and we would like to thank her for her help, suggestions and numerous discussions.
We gladly take this opportunity to thank **Dr. Rajurkar A. M.**, Head of Computer Science & Engineering.
We are heartily thankful to **Dr. Lathkar G. S.**, Director, MGM's College of Engineering, Nanded for providing facility during progress of project, also for her kindly help, guidance and inspiration.
Last but not least we are also thankful to all those who helped directly or indirectly to develop this project and complete it successfully.

With Deep Reverence,

Miss. Talatunnisa A. Siddiqui (101)
Miss. Snehal R. Savandkar (104)
Miss. Rasika R. Rakhewar (108)
Miss. Shruti P. Pimple (119)
Miss. Prajakta S. Waghmare (122)

ABSTRACT

Enhancing road safety through automated surveillance stands as a critical objective for modern Intelligent Transportation Systems. Wrong-way driving incidents, in particular, pose a severe risk of high-impact collisions, necessitating reliable and immediate detection. This report details the development of a comprehensive, end-to-end application designed to automatically identify, track, and report vehicles moving against the proper flow of traffic. The system operates by processing video feeds to not only flag violations in real-time but also to automatically generate isolated video clips of these incidents, providing crucial, ready-to-review evidence for traffic management and enforcement purposes.

The system's analytical engine is built upon the robust Tracking-by-Detection paradigm, which synergizes the capabilities of two state-of-the-art deep learning components. For initial object localization, the lightweight and highly efficient YOLOv8n model is employed, striking an optimal balance between processing speed and detection accuracy. Once detected, vehicle identities are persistently maintained across frames using the DeepSORT algorithm, which leverages a Kalman Filter to predict vehicle motion and manage temporary occlusions. A key algorithmic contribution of this work is the novel "Majority Voting" system for violation detection. By dynamically assessing the movement vectors of all vehicles, the system autonomously determines the correct direction of traffic flow, thereby eliminating the need for manual pre-configuration and enhancing its adaptability to diverse road environments.

A primary focus of the project was achieving practical, real-time performance on standard computing hardware. This was realized through a multi-faceted optimization strategy that includes intelligent frame skipping, the selection of a lightweight AI model, and asynchronous video I/O operations. These enhancements collectively yielded a significant 3-to-4-fold improvement in overall processing speed, reducing analysis time for typical video segments from 40-50 seconds down to approximately 10-15 seconds. The system's reliability was empirically validated through quantitative metrics, demonstrating high precision and recall in identifying true violations. Ultimately, this work delivers a practical, efficient, and deployable solution that translates advanced computer vision techniques into a tangible tool for improving traffic safety.

Technologies used: Python, FastAPI, Uvicorn, OpenCV (cv2), Ultralytics YOLOv8, NumPy, React, Vite, Tailwind CSS, Material UI (MUI), Framer Motion.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

The rapid expansion of urban centers and the corresponding increase in vehicular traffic have made the management of road networks one of the most complex challenges of the 21st century. As transportation infrastructure struggles to keep pace with demand, issues such as traffic congestion, accidents, and inefficient traffic flow have become commonplace. In response, the field of Intelligent Transportation Systems (ITS) has emerged, leveraging advancements in computing, sensing, and communication technologies to create safer, smarter, and more efficient mobility solutions. At the heart of modern ITS is the ability to automatically perceive and understand the state of the traffic environment in real-time.

Computer vision, particularly powered by deep learning, has become the cornerstone of this perceptual capability. Unlike traditional sensor-based methods (e.g., inductive loops or radar), camera-based systems offer a rich, contextual understanding of the road, enabling the simultaneous detection and classification of multiple objects, the analysis of their behavior, and the identification of anomalous events. This project harnesses these advanced capabilities to address a particularly dangerous and critical traffic violation: wrong-way driving. By developing a complete, end-to-end system, this work aims to bridge the gap between theoretical AI models and a practical, deployable tool for enhancing road safety.

1.1 Motivation and Background

The motivation for this project stems from a confluence of societal needs and technological advancements. On one hand, there is a clear and urgent requirement for more effective road safety measures. On the other, the recent maturation of deep learning has provided the tools necessary to build systems that were once considered computationally infeasible.

- **The Societal Imperative for Road Safety:** Global motorization has brought unprecedented mobility, but it has come at a cost. According to the World Health Organization, road traffic injuries remain a leading cause of death globally. While many accidents are caused by common errors, certain types of incidents have a disproportionately high rate of severity. Wrong-Way Driving (WWD) is one such incident. Although WWD events constitute a small fraction of total

traffic accidents, they are significantly more likely to result in severe injuries or fatalities compared to other types of collisions. These incidents typically occur on high-speed, divided highways or access ramps where a head-on collision is the most probable outcome. The primary causes often include driver confusion, poor signage, or impairment. The catastrophic nature of these events underscores the critical need for proactive detection and warning systems that can operate tirelessly, 24/7, without human intervention.

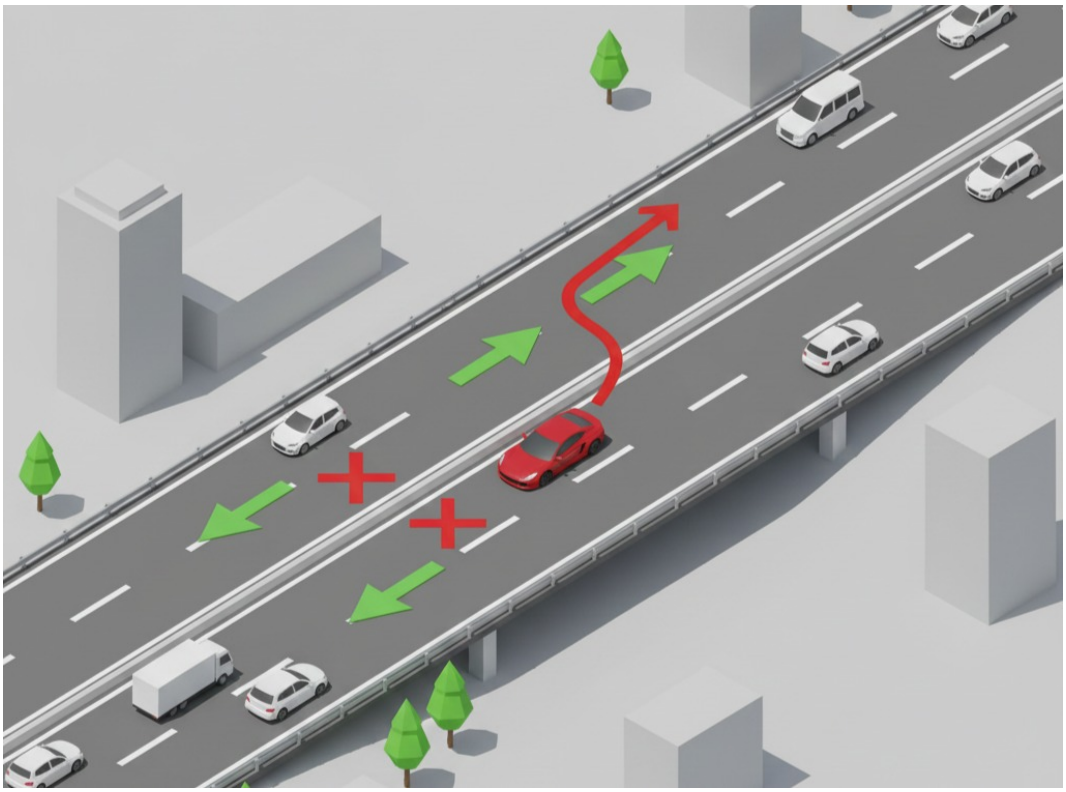

Fig. 1.1: Illustration of a wrong-way driving (WWD) incident on a highway access ramp, showing the correct flow of traffic versus a violating vehicle.

- **The Technological Enabler:** Intelligent Transportation Systems represent a paradigm shift from passive infrastructure to active, data-driven traffic management. ITS integrates information and communication technologies into transportation infrastructure to manage traffic, improve safety, and reduce congestion. Within this ecosystem, video surveillance plays a pivotal role. Cameras are ubiquitous and provide a rich source of data, but the sheer volume generated makes manual analysis untenable. The true value is unlocked through automated video analytics. The recent revolution in deep learning, particularly Convolutional Neural Networks (CNNs), has made it possible to build robust, accurate, and real-time video analysis systems capable of understanding complex traffic scenes, forming the technological foundation upon which this project is built [**?**].

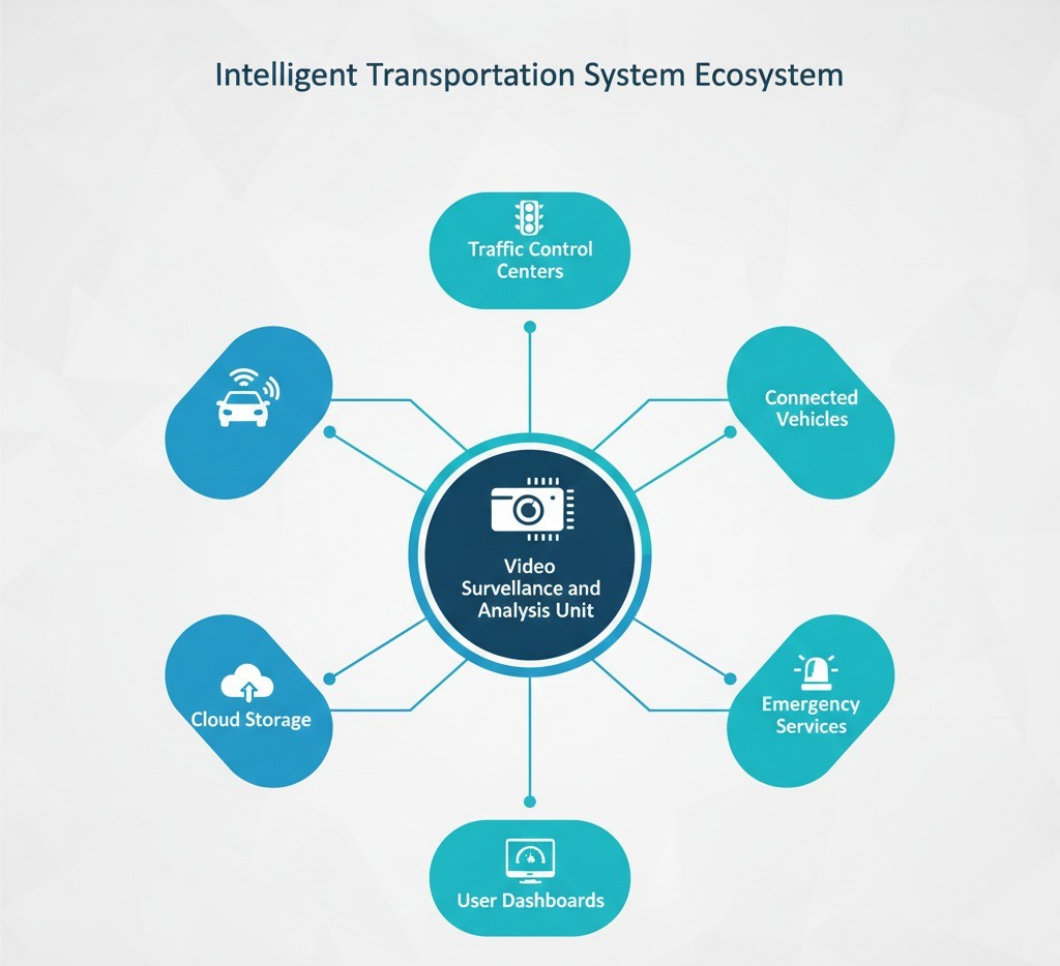

Fig. 1.2: The ecosystem of an Intelligent Transportation System (ITS), highlighting the central role of automated video surveillance and analysis.

1.2 Problem Statement

The limitations of manual traffic monitoring and the critical danger posed by wrong-way driving incidents define the core problem this project seeks to solve. The formal problem statement is as follows:

> *To design, implement, and evaluate a computationally efficient, end-to-end system that automatically processes video feeds of traffic to detect, track, and report vehicles traveling against the established flow of traffic. The system must operate in near real-time, maintain high accuracy, and provide actionable evidence by generating isolated video clips of identified violations.*

This overarching goal can be decomposed into several key technical requirements:

- **High-Accuracy Vehicle Detection:** Reliably detect the presence and location of various vehicle types.

- **Persistent Object Tracking:** Assign a unique, persistent identity to each vehicle and track its movement across frames, even through short-term occlusions.
- **Autonomous Directional Analysis:** Dynamically infer the predominant traffic flow without relying on pre-configured rules.
- **Real-Time Processing and Efficiency:** Ensure the analysis keeps pace with the incoming video stream on standard hardware.
- **Actionable Evidence Generation:** Automatically isolate and save the video segment corresponding to the violation for quick review.
- **Integrated System Architecture:** Develop a complete application with a clear interface for user interaction and result visualization.

1.3 Scope of the Project

To ensure the project remains focused and achievable, its scope is clearly delineated between in-scope functionalities and out-of-scope extensions.

- **In-Scope Activities:** The system is designed to process a single, pre-recorded video file uploaded by a user. The underlying AI model detects and tracks common vehicle classes (cars, motorcycles, buses, trucks) using 2D bounding boxes. The project includes the development of a full-stack application with a Python backend and a React frontend, which communicate via WebSocket for real-time feedback. The core deliverables are the automatic wrong-way violation reporting logic and the generation of evidentiary clips for each violation.
- **Out-of-Scope Activities:** Direct integration with live camera streams (e.g., RTSP) is considered a future extension. The system does not perform License Plate Recognition (LPR) to identify specific vehicles. Furthermore, the project does not involve multi-camera sensor fusion, the analysis of non-vehicular objects like pedestrians, or the estimation of advanced physical vehicle dynamics beyond basic 2D motion.

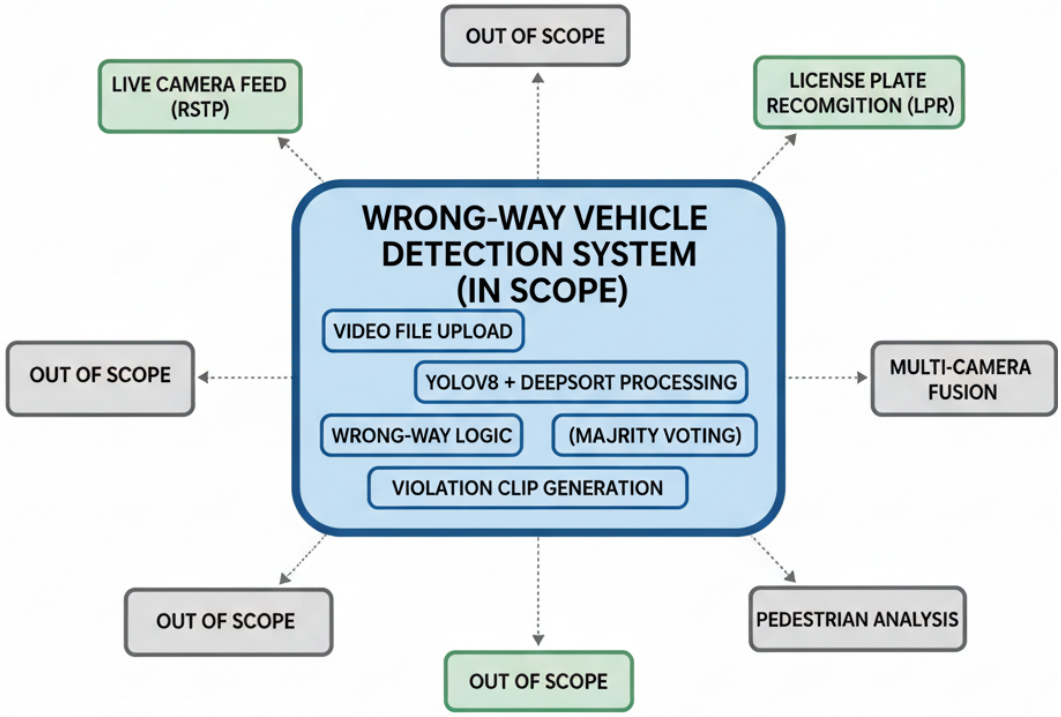

Fig. 1.3: System scope diagram, showing in-scope components within the central system and out-of-scope functionalities as external extensions.

1.4 Historical Context: The Evolution of Object Detection and Tracking

The approach taken in this project is the culmination of decades of research in computer vision. Understanding this evolution is key to appreciating why the chosen technologies represent the current state-of-the-art for this task.

- **Traditional Computer Vision Approaches:** Before deep learning, object tracking relied on handcrafted features and statistical models such as frame differencing, background subtraction, and optical flow. While foundational, these methods were often brittle, computationally expensive, and lacked the robustness to handle the complexities of real-world traffic scenes.
- **The Deep Learning Revolution:** The success of Convolutional Neural Networks (CNNs) in image classification catalyzed a shift towards learning-based object detection. This led to two main paradigms:
  - *Two-Stage Detectors (e.g., R-CNN family):* These methods first propose candidate object regions and then classify them. They are known for high accuracy but are often too slow for real-time video analysis.
  - *Single-Stage Detectors (e.g., YOLO family):* These models treat detection as a single regression problem, directly predicting bounding boxes and class

probabilities in one pass. Their unified architecture is dramatically faster, making them the standard for real-time applications.

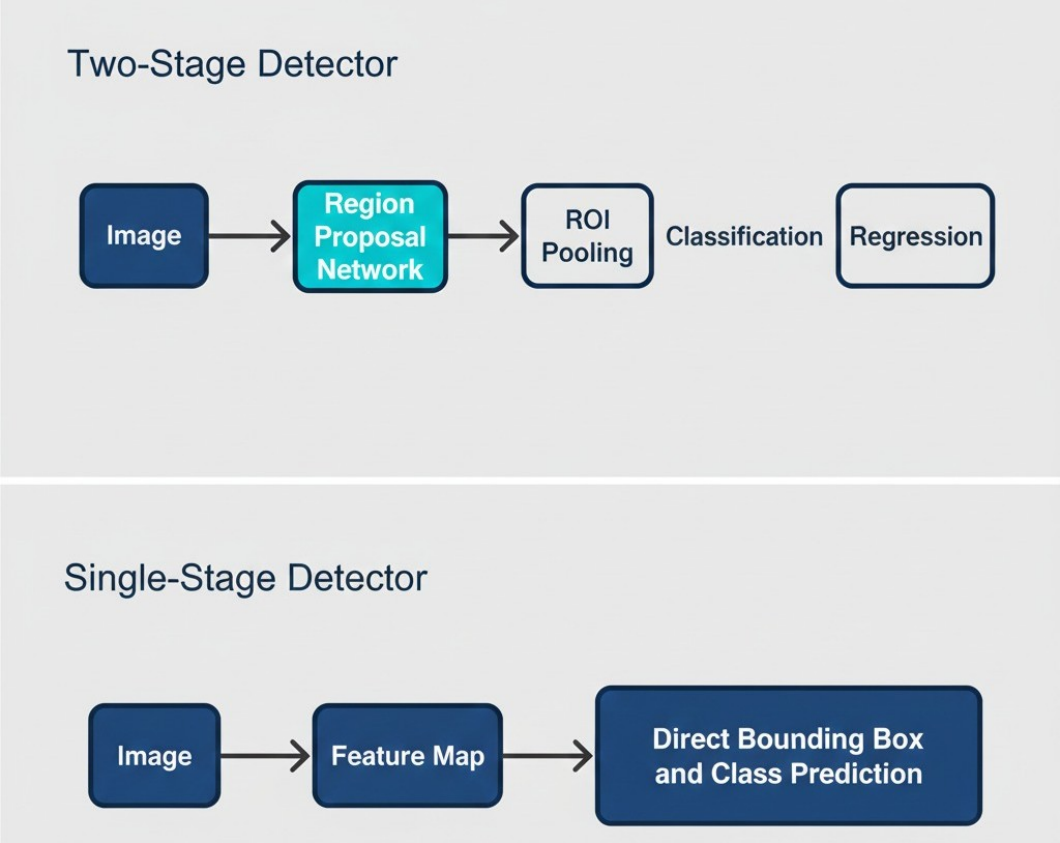

Fig. 1.4: High-level comparison of two-stage (propose-then-classify) and single-stage (unified regression) object detection pipelines.

- **Advancements in Real-Time Tracking:** With reliable detectors, the dominant paradigm became **Tracking-by-Detection**. The evolution of algorithms in this space has been critical. The **Kalman Filter** was introduced to predict object motion and handle short occlusions. This was refined in the **SORT** algorithm, which added efficient data association. Finally, **DeepSORT** improved upon SORT by incorporating a deep learning-based appearance model (Re-ID), allowing it to re-associate objects based on their visual features even after long occlusions. The selection of the YOLOv8 + DeepSORT stack for this project is therefore a deliberate choice, leveraging a state-of-the-art combination for real-time detection and robust tracking[2].

1.5 Organization of the Report

This report is structured into five chapters to systematically present the project from conception to conclusion.

- **Chapter 1: Introduction** provides the motivation, background, problem statement, and scope of the project, establishing the context and objectives.
- **Chapter 2: Literature Survey** conducts a detailed review of existing academic work in object detection and tracking, analyzing key papers to position this project within the current research landscape.
- **Chapter 3: System Architecture and Methodology** offers a comprehensive technical description of the implemented system, detailing the architecture, technology stack, and the core processing pipeline.
- **Chapter 4: Results and Performance Evaluation** presents the empirical validation of the system, analyzing its performance through qualitative visual examples and quantitative metrics.
- **Chapter 5: Conclusion** summarizes the key findings and achievements of the project and proposes concrete directions for future research and development.

Chapter 2

LITERATURE SURVEY

The field of automated vehicle analysis is built upon decades of research in computer vision, with recent advancements in deep learning fundamentally reshaping what is possible. A thorough review of the existing literature is essential to understand the state-of-the-art, identify foundational techniques, and position this project within the broader academic and technical landscape. This chapter surveys key research papers relevant to the core components of our system: object detection, object tracking, and the specific challenges of real-world video analysis.

We begin by examining the evolution and architecture of modern object detectors, with a focus on the YOLO family and its recent improvements. Next, we delve into the principles of multi-object tracking, analyzing the Tracking-by-Detection paradigm and the critical role of algorithms like DeepSORT and the Kalman Filter. By analyzing a curated set of academic papers, we evaluate different approaches, from model-level architectural innovations to system-level integration strategies. The chapter culminates in a comparative analysis that distills the strengths and weaknesses of these methods, identifies a clear research gap, and outlines how our project adapts and builds upon these established insights to create a practical, end-to-end solution for wrong-way vehicle detection.

2.1 Analysis of Foundational and State-of-the-Art Research

A robust wrong-way vehicle detection system is not a monolithic entity but rather an integration of specialized solutions addressing distinct challenges in the computer vision pipeline. The following detailed analysis of key academic papers explores the nuances of each technological pillar from the architectural intricacies of object detectors and the predictive mathematics of trackers to the practical realities of handling imperfect video data. This deep dive provides the foundational knowledge and justification for the architectural and algorithmic decisions made in this project.

X. Liu et al. YOLOv8-FDD: A Real-Time Vehicle Detection Method Based on Improved YOLOv8. [1]

This research presents a highly specialized variant of the YOLOv8n architecture, named "YOLOv8-FDD," which has been meticulously re-engineered for the specific domain of vehicle detection in traffic scenes. The authors' primary contribution is a series

of targeted architectural modifications that collectively reduce model size, decrease computational load, and, counter-intuitively, increase detection accuracy compared to the baseline YOLOv8n model. The work serves as a powerful demonstration that general-purpose object detectors can be significantly enhanced when tailored to the specific statistical properties and challenges of a given domain.

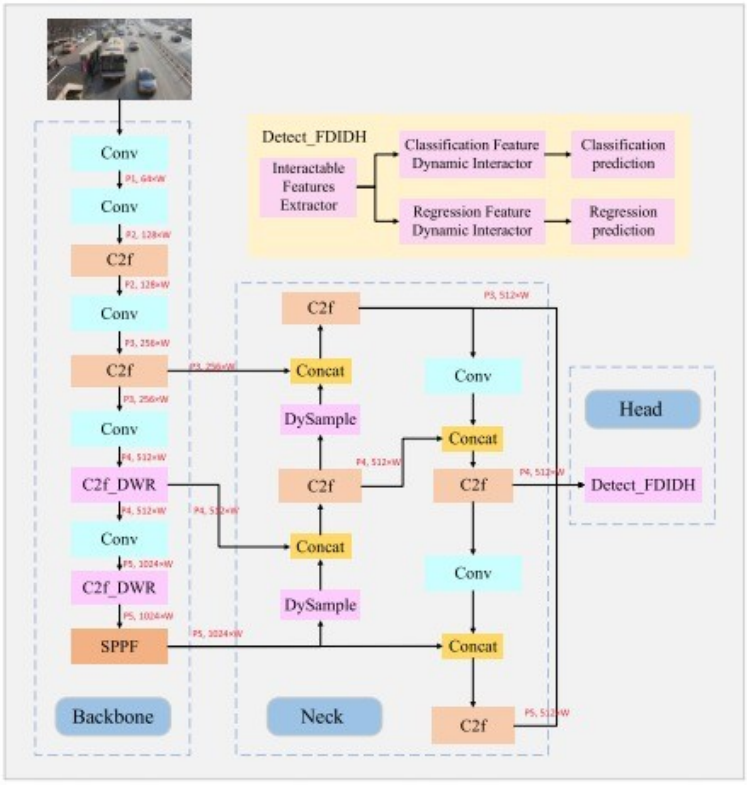

Fig. 2.1: The complete network architecture of the YOLOv8-FDD model, as proposed by Liu et al. [1].

The diagram highlights the three main components: the Backbone, the Neck, and the Head. The modified modules, including the C2f_DWR in the backbone, DySample in the neck, and the Detect_FDIDH head, are shown, illustrating the flow of feature maps from input to final detection.

Architectural Innovations and Analysis: The strength of YOLOv8-FDD lies in its multi-pronged approach to optimization. The first major innovation is the **Feature Sharing Detection Head (FSDH)**. The standard YOLOv8 architecture employs a "decoupled head," where separate convolutional branches are used to predict object classification and bounding box regression for each feature map scale. While effective, this design leads to a significant number of parameters. The FSDH radically redesigns this by introducing a single, parameter-shared feature extractor that serves all detection scales. This architectural choice dramatically compresses the model, reducing the parameter count of the YOLOv8n head from **752k to just 107k (a reduction of 85.7%)**. This is a

critical finding for applications intended for resource-constrained environments.
Building upon this, the authors introduce the **Feature Dynamic Interaction Detection Head (FDIDH)** to address a known limitation in decoupled heads: the lack of information flow between the classification and localization tasks. The FDIDH introduces a novel cross-connection mechanism, termed "Feature Dynamic Interactors," which allows the feature maps from one task to dynamically influence the other. This interaction ensures that the final predictions are more coherent. For traffic scenes, this improves the quality of the data fed into the downstream tracking algorithm.
To further enhance the model's perceptual capabilities, the paper incorporates a **Dilation-wise Residual (DWR)** module into the network's backbone. The DWR module uses parallel branches of dilated convolutions with different rates to expand the network's receptive field without increasing parameters. A larger receptive field is particularly advantageous for vehicle detection, as it enables the model to leverage a wider range of contextual information, helping to differentiate vehicles from complex backgrounds and detect objects at vastly different scales.
Relevance and Application to Our Project: This paper is of paramount importance to our work. It provides strong academic validation for selecting the YOLOv8 architecture as our foundation. The paper's results are striking: on the UA-DETRAC dataset, the proposed YOLOv8-FDD model achieved a **mAP50-95 of 84.9%**, an improvement of **1.3%** over the baseline YOLOv8n, while reducing the total parameter count from **3.0M to 2.19M (a 27% reduction**.

Fig. 2.2: Visualization of results of paper1

Furthermore, it maintained a high processing speed of **314.1 FPS**. This demonstrates that it is not always necessary to scale up to a larger model to achieve higher accuracy; intelligent architectural design can yield a superior outcome. This insight justifies our choice of starting with the lightweight YOLOv8n model and focusing on system-level optimizations, knowing that model-level improvements offer a viable future path.

N. Wang et al. Advanced YOLO-DeepSort-Based System for Drainage Pipeline Defects. [2]

This research provides a compelling case study on the practical application of the YOLO and DeepSORT framework to a challenging industrial inspection task: the automated detection of defects in drainage pipelines. The authors integrate YOLOv7 with an enhanced DeepSORT tracker, demonstrating the stack's robustness and adaptability. The work's primary contribution is not just the AI model but the creation of a complete, end-to-end "information management platform" that turns raw detections into actionable reports.

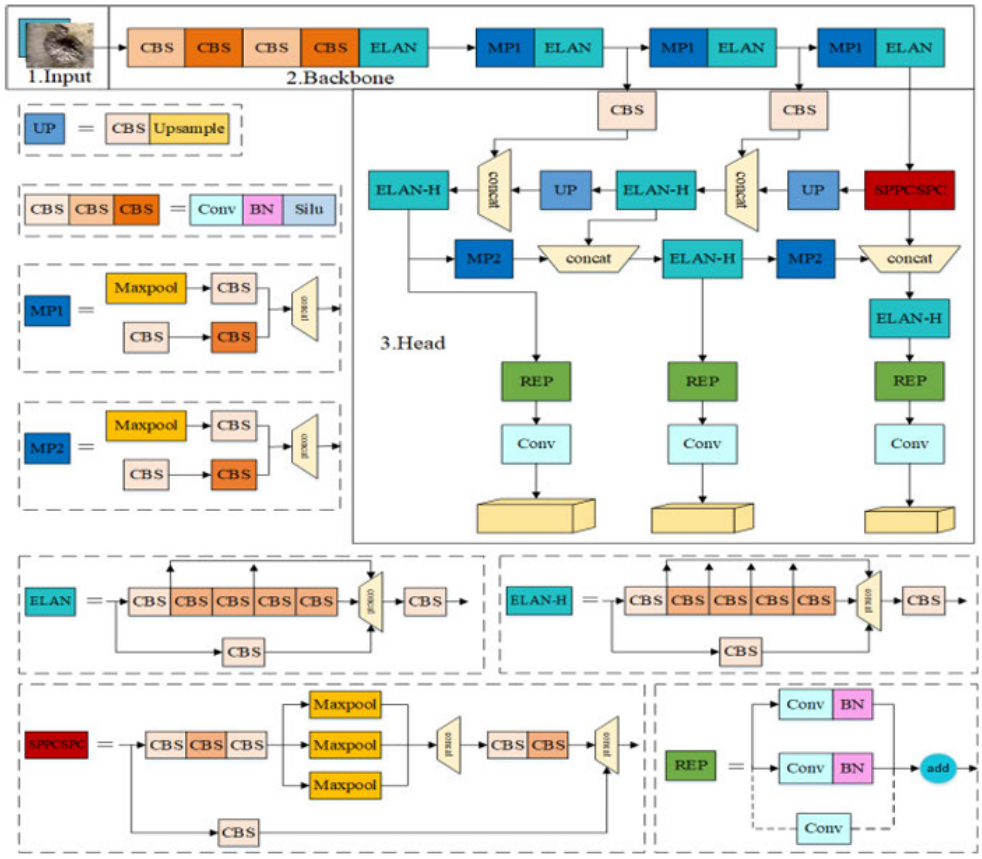

Fig. 2.3: Network of YOLO of paper2

System-Level Integration and Tracking Robustness: This research is valuable for its holistic, system-level perspective. The authors treat the detection and tracking components as integral parts of a larger operational workflow, a philosophy our project shares. Technically, the paper focuses on enhancing the robustness of the DeepSORT tracker. The authors implement a **fusion module** that creates an integrated cost matrix for data association by combining the Kalman Filter's motion prediction (Mahalanobis distance) with the Re-ID network's appearance features (cosine distance). This makes the tracker more resilient to occlusions and identity switches.

This enhanced robustness is critical for their application, where a defect must be tracked consistently across frames. The same principle applies directly to our traffic monitoring use case. In dense traffic, vehicles frequently occlude one another. A robust tracker that can reliably re-identify a car after it has passed behind a truck is essential for maintaining an accurate trajectory history. The paper reports impressive performance, achieving a mean Average Precision (**mAP**) of **91.1%** and a processing speed of **172 FPS** on their custom dataset. This performance, even in a visually challenging domain, provides strong confidence in the YOLO+DeepSORT paradigm.

Relevance and Application to Our Project: This paper directly validates our core architectural choice. The success of their system in a difficult, non-vehicular domain underscores the flexibility and power of the YOLO + DeepSORT stack. It confirms that this combination is a sound foundation for building reliable tracking applications. Furthermore, their emphasis on a complete, end-to-end system with a user-facing platform reinforces the value of our project's own full-stack architecture. The high FPS and mAP they achieved demonstrate that this stack can deliver both speed and accuracy, aligning perfectly with our project's goals.

T. Chen et al. Design of Vehicle Running States-Fused Estimation Strategy Using Kalman Filters. [3]

This paper focuses on high-precision vehicle state estimation for vehicle dynamics control, proposing a novel "fused estimation strategy" that combines an advanced Kalman Filter variant with tire force compensation. The research is deeply rooted in control theory, aiming to create a highly accurate "virtual sensor" for a vehicle's physical state.

Deep Dive into Kalman Filter Complexity: The key component in this paper is the **Weighted Square-Root Cubature Kalman Filter (WCKF)**, a sophisticated algorithm designed to handle the highly nonlinear relationships in vehicle physics. Unlike a standard linear Kalman Filter, the WCKF uses a set of deterministically chosen "cubature points" to approximate the state distribution, making it far more accurate for complex systems. It takes inputs not from a camera but from vehicular sensors and a complex mathematical model of the vehicle's drivetrain.

The results in the paper are measured not in detection accuracy, but in estimation error. For instance, in a sinusoidal steering maneuver simulation, the proposed WCKF method reduced the Peak Relative Error (PRE) for longitudinal vehicle speed from **0.1254 to 0.0469** and the Root Mean Square Error (ERMS) from **0.2273 to 0.1008** compared to a baseline method. This demonstrates a high degree of precision in estimating the vehicle's physical state.

Relevance by Contrast and Justification: This paper is highly relevant to our report by way of contrast. It allows us to justify the level of complexity chosen for our own tracking model. Our system's Kalman Filter, as used in DeepSORT, is a linear filter

operating on a simple state space (bounding box coordinates and their velocities). It assumes a constant-velocity motion model, which is computationally cheap and effective for predicting a bounding box's location. The WCKF, in contrast, is computationally intensive and requires a precise physical model. By making this comparison, we can authoritatively state that while advanced Kalman Filters are essential for high-fidelity vehicle dynamics, a simpler linear KF is the **appropriate, efficient, and standard choice** for the specific problem of multi-object tracking in the image plane.

J. W. Soh et al. Reduction of Video Compression Artifacts Based on Deep Temporal Networks. [4]

This paper addresses the pervasive issue of video quality degradation caused by standard compression algorithms like H.264/HEVC. The authors propose a deep learning model that leverages **temporal information** looking at adjacent frames to restore the current frame and reduce common compression artifacts.

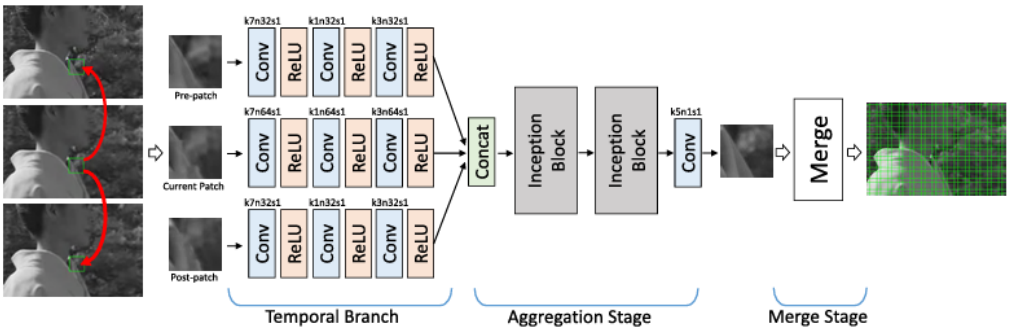

Fig. 2.4: The overall framework of the proposed network, where k, n, s above the convolution layers denote the kernel size, the number of output feature maps, and the convolution strides, respectively.

Methodology and Technical Analysis: The core of their solution is a temporal Convolutional Neural Network (CNN). The model performs a motion search to find and align similar patches from neighboring frames. These temporally aligned patches are then fed into a CNN, which learns to aggregate information from multiple noisy versions to produce a single, cleaner output. This approach is effective at mitigating artifacts like blocking, blurring, and flickering.

The experiments show significant improvements in objective quality metrics. For example, when applied to HEVC-compressed videos, their method achieved an average Peak Signal-to-Noise Ratio (PSNR) gain of **0.23 dB** over the original compressed video. For older codecs like MPEG-2, the gain was even more substantial, at **1.27 dB**. These numerical results quantitatively demonstrate the effectiveness of using temporal context for video restoration.

Relevance and Impact on Our Project: This research is critically important for

contextualizing our project's performance in a real-world setting, as surveillance video is almost always heavily compressed. These artifacts can severely degrade the performance of YOLOv8 and DeepSORT, leading to false negatives, false positives, and tracking failures. While our project does not implement a dedicated temporal pre-filtering network due to the significant added computational cost, this paper provides the academic justification for why video quality is a major challenge. It informs our optimization strategy: by using a robust tracker like DeepSORT and system-level optimizations like frame skipping, we aim to build a system resilient enough to function on standard compressed video without needing an explicit, computationally expensive restoration step.

K.-H. Choi et al. Object detection method using image and number of objects on image as label. [5]

This paper proposes a reinforcement-learning-based object detection framework that requires only an image and its total object count as labels, completely removing the need for costly bounding box annotations.

Introduction and Problem Statement Conventional supervised object detection frameworks such as YOLO rely on fully-annotated datasets where each object is enclosed in a manually drawn bounding box. This annotation step is extremely time-consuming, expensive, and prone to human error. The authors aim to eliminate this bottleneck by introducing a weakly supervised detection system that learns to localize objects using only two labels: the image itself and a single integer representing the number of objects present. This drastically reduces annotation effort while maintaining competitive performance.

Methodology:

The proposed approach formulates object detection as a reinforcement learning (RL) problem using an Actor–Critic framework based on Proximal Policy Optimization (PPO). The learning agent is trained to propose bounding boxes using only weak supervisory signals.

Actor Model: The Actor is responsible for generating candidate bounding boxes. It uses a transformer-based architecture inspired by DETR, with image features extracted using a ResNet backbone and a set of object queries. For each query, instead of directly predicting a bounding box, the Actor outputs the parameters (mean and standard deviation) of a Gaussian distribution from which a continuous bounding box is sampled. This enables a continuous action space. The Actor also predicts the probability that each sampled region contains an object.

Critic Model: The Critic evaluates the state, defined as the combination of the image and the bounding boxes proposed by the Actor. It predicts the expected reward to stabilize and guide the Actor's learning during PPO optimization.

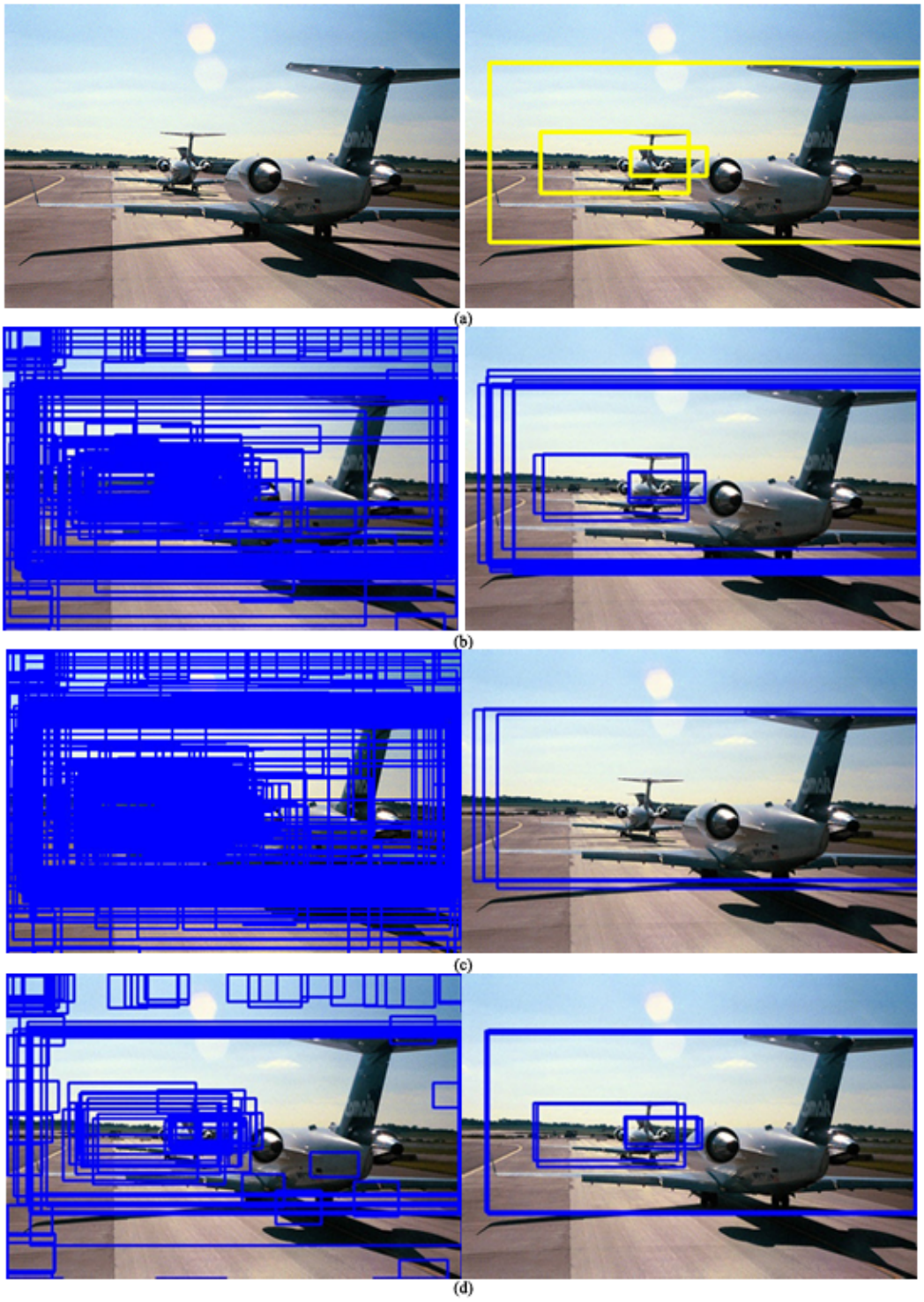

Fig. 2.5: Changes in image detection results according to training progress during additional training in a new environment (a) original images and ground-truth labels of objects (b) epoch 0 (c) epoch 15 (d) epoch 95 ((b) (d) left: actor output results which display all 100 images regardless of object probability right: model detection results after non-maximum suppression).

Reward Model (Key Innovation): Without ground-truth bounding boxes, the authors design a novel reward system. A separate **Object Evaluation Model**, a pre-trained ResNet-101 classifier, is used to estimate the probability that a cropped region contains an object.

The total reward consists of:

- **Region Evaluation Reward:** Each proposed region is passed to the evaluation model. Higher object probability yields positive reward; background yields negative reward.
- **Object Count Reward:** The number of predicted "object" boxes is compared with the ground-truth count for the image. Smaller differences yield higher rewards, guiding the Actor to propose the correct number of objects.

Key Contributions

Weakly-Supervised Detection Framework: A complete object detection pipeline trained without bounding boxes, relying only on object count labels.

Reinforcement Learning Formulation: A novel application of Actor–Critic RL for continuous bounding box proposal generation.

Innovative Reward System: Combines pre-trained objectness evaluation with object-count consistency to guide learning in the absence of detailed labels.

Results and Evaluation

The method is trained on the COCO dataset using only images and object count labels.

Box AP: Achieved 43.02% on COCO, which is remarkably close to supervised transformer-based methods such as DETR (43.11%).

Generalization Ability: When fine-tuned on PASCAL VOC using only object counts, the model continued to improve, demonstrating strong transferability.

Comparison: Despite the lack of bounding box annotations during training, the system performs comparably to early supervised models.

Relevance to Your Project: This paper presents a fundamentally different paradigm from your fully supervised approach. It is highly relevant for discussing:

- the limitations of supervised detection, including annotation cost and dataset dependency,
- state-of-the-art weakly supervised and RL-based methods, and
- future work directions such as training traffic detectors with fewer labels or enabling self-adapting detection models for new camera angles without re-labeling.

Y.-W. Chen et al. Moving Object Counting Using a Tripwire in H.265/HEVC Bitstreams for Video Surveillance. [6]

This paper proposes an ultra-efficient object counting method that processes motion vectors directly from H.265/HEVC compressed video bitstreams, eliminating the need for full decoding and enabling high-speed surveillance analysis.

Introduction and Problem Statement Traditional pixel-domain video analytics require full decoding of compressed streams before processing. For high-resolution surveillance footage, decoding alone becomes a major computational bottleneck, especially on resource-limited devices. The authors aim to overcome this by performing detection and counting directly in the compressed domain, enabling extremely fast processing at minimal computational cost.

Methodology:

The system extracts motion information directly from HEVC bitstreams and performs object detection, tracking, and tripwire-based counting without reconstructing pixel-level frames.

Foreground Detection from Motion Vectors (MVs): Motion vectors embedded in the compressed stream form the core of the detection logic. The authors introduce a novel metric called **Momentum**, defined as:

$$\text{Momentum} = \text{Movement} \times \text{Mass}$$

where Movement is MV magnitude and Mass is the number of prediction blocks in a coding unit. Momentum effectively distinguishes true motion from noisy MVs in static regions. Blocks with high momentum are classified as foreground.

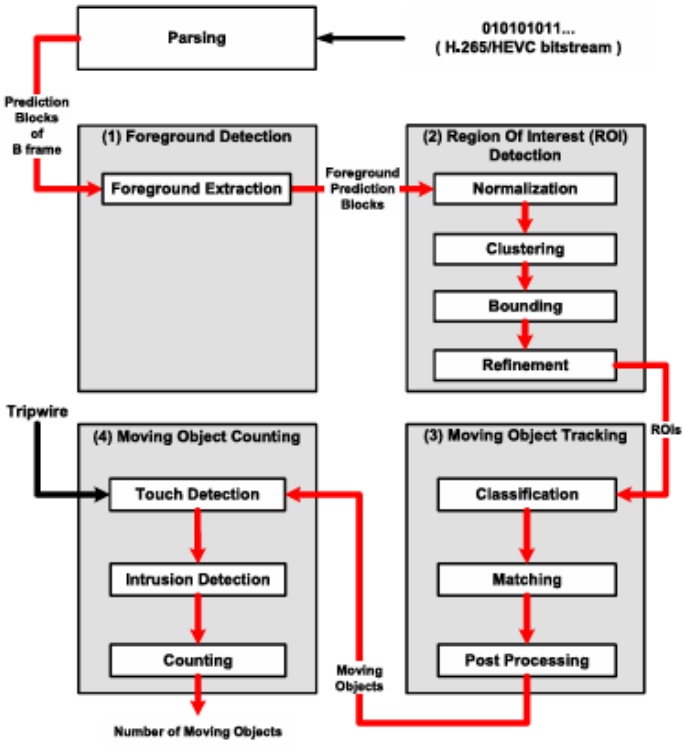

Fig. 2.6: Block diagram of the proposed moving object counting. of this 6th paper

Region of Interest (ROI) Detection: Foreground blocks are mapped to a downscaled Index Map, clustered using connected-component labeling. A pseudogravitation refinement merges fragmented ROIs into coherent objects.

Moving Object Tracking and Counting: Objects are tracked by matching ROIs across frames using a Matching Degree based on edge distance and track Age. A virtual tripwire is defined in the scene. A tracked object is counted only if:

- its direction of motion matches a predefined rule, and
- its Age exceeds a stability threshold, ensuring reliable tracking.

Key Contributions

Compressed-Domain Processing: Achieves dramatic speedups by avoiding full decoding and pixel-domain processing.

Robust Foreground Detection: The Momentum metric filters out noisy vectors and improves moving object detection accuracy.

Complete Tripwire-Based Counting System: Provides an end-to-end solution combining ROI clustering, tracking, and event-based counting.

Results and Evaluation

Speed: Achieved over 400% performance improvement compared to traditional pipelines requiring decoding and background subtraction.

Accuracy: Despite the massive speed gain, accuracy loss was negligible (less than 0.02% counting error per frame).
Datasets: Successfully evaluated on AVSS 2007 and PETS 2006/2007 datasets.
Relevance to Your Project: This paper provides a strong contrast to your pixel-domain YOLO-based system. While their compressed-domain approach is extremely fast, it lacks the capability to perform pixel-level tasks such as drawing bounding boxes, computing object centers, or running deep learning detectors. This helps justify your system's design choices, where pixel-level processing is necessary to provide advanced visualization and accurate detection. You may also reference potential hybrid future work where compressed-domain motion analysis acts as a low-power pre-filter before full-frame YOLO processing.

S. Göring et al. Modular Framework and Instances of Pixel-Based Video Quality Models for UHD-1/4K. [7]
This paper introduces a comprehensive and modular framework for constructing pixel-based video quality models, specifically designed for modern UHD-1/4K video streaming applications. The authors propose four distinct model instances derived from this framework—*nofu* (no-reference), *hyfu* (hybrid no-reference), *fume* (full-reference), and *hyfr* (hybrid full-reference)—each tailored to different usage scenarios and levels of data availability. The primary contribution of this work is a unified, open-source pipeline that integrates an extensive feature set with machine learning techniques to deliver accurate video quality predictions. The proposed models demonstrate superior performance compared to existing state-of-the-art approaches such as VMAF, particularly in scenarios involving framerate variations.
Methodology and Technical Analysis: The core of the proposed solution is a flexible, multi-stage pipeline designed for pixel-based video quality estimation. The framework relies on a large set of per-frame pixel-based features, which are categorized into two major groups:

- **Image-based features (img):** These features analyze static characteristics within individual frames, including contrast, blur (measured using Laplacian variance), colorfulness, saturation, noise levels, and spatial information (SI).
- **Motion-based features (mov):** These features capture temporal dynamics such as temporal information (computed as RMSE between consecutive frames), motion estimation using block-based motion vectors, and temporal information (TI).

Based on the available input data, the framework is instantiated into four model variants:

- **nofu (No-Reference):** Utilizes features extracted solely from the distorted video stream.

- **fume (Full-Reference):** Incorporates features from both the distorted video and the original reference video, along with full-reference metrics such as PSNR, SSIM, and VIF.
- **hyfu and hyfr (Hybrid Models):** Extend the no-reference and full-reference models by integrating bitstream metadata (bs), including codec type, bitrate, resolution, and framerate. This metadata provides contextual information that purely pixel-based approaches cannot capture.

After feature extraction, a temporal pooling stage is applied to aggregate per-frame feature vectors into a single time-independent representation for each video sequence. This is achieved by computing 25 statistical descriptors—such as mean, standard deviation, and quantiles—for every feature. The resulting pooled feature set is then used to train a Random Forest regressor to predict the final Mean Opinion Score (MOS).
Experimental evaluation on the UHD-1/4K validation dataset (AVT-VQDB-UHD-1) demonstrates the effectiveness of the proposed framework. The hybrid full-reference model (*hyfr*) achieved a Pearson Correlation Coefficient (PCC) of 0.922 with subjective quality scores, significantly outperforming the widely used VMAF model, which attained a PCC of 0.816 on the same dataset. The advantage was even more pronounced under framerate variation conditions, where *hyfr* achieved a PCC of 0.881 compared to VMAF's reduced performance of 0.789. These results quantitatively validate the superiority of the modular framework and highlight the benefits of incorporating hybrid metadata.
Relevance and Impact on Our Project: This research is highly relevant to our project as it emphasizes the critical influence of video quality on downstream algorithmic performance and provides a robust framework for its objective measurement. Although our work focuses on downstream tasks such as vehicle tracking and wrong-way violation detection rather than video quality assessment itself, the findings of this paper offer several valuable insights.
First, the paper confirms that modern UHD/4K video formats and advanced codecs pose challenges that traditional quality metrics fail to address adequately. This reinforces the need for robust detection and tracking algorithms, such as YOLOv8, that can operate effectively under varying video quality conditions. Second, the superior performance of the hybrid models demonstrates the importance of combining pixel-based analysis with simple metadata such as resolution, bitrate, and framerate. This insight suggests a potential future enhancement for our system, where detection or tracking strategies could be dynamically adapted based on the estimated quality of the incoming video stream. Finally, the paper's focus on framerate variation directly aligns with our frame-skipping optimization strategy. It provides academic validation for the challenges introduced by temporal inconsistencies and informs our understanding of the trade-offs involved in

optimizing real-time processing performance.

M. Sánchez-Beeckman et al. Combining Pre- and Post-Demosaicking Noise Removal for RAW Video. [8]

This paper proposes a sophisticated self-similarity-based video denoising framework that intelligently balances noise removal before and after the demosaicking stage. The authors argue that a hybrid approach, which combines pre-demosaicking and post-demosaicking denoising, is superior to performing denoising exclusively in either domain. The key innovation is a tunable, two-stage denoising scheme that adapts to the noise level (ISO) of RAW video data. By leveraging temporal information, the method improves texture preservation and overall visual quality without relying on deep learning techniques.

Methodology and Technical Analysis: The proposed approach is implemented as a multi-stage pipeline operating directly on RAW video captured from a camera sensor equipped with a Bayer Color Filter Array (CFA). The denoising process is divided into two complementary stages.

Dual Denoising Stages:

- **Pre-Demosaicking Denoising (Stage 1):** This stage is applied directly to the mosaicked RAW data. Denoising in the RAW domain is advantageous because sensor noise, including shot and read noise, is spatially uncorrelated and follows a predictable Poisson–Gaussian distribution. However, aggressive denoising at this stage can introduce aliasing and checkerboard artifacts due to the CFA structure.

- **Post-Demosaicking Denoising (Stage 2):** After demosaicking, the RAW data is interpolated into a full-resolution RGB image, and a second denoising step is applied. Operating in the RGB domain helps prevent aliasing artifacts and enables color-aware processing. However, the demosaicking process introduces spatial and chromatic noise correlations, making the noise more complex and difficult to remove effectively.

Optimal Balancing Using the α Parameter: To optimally distribute the denoising effort between the two stages, the authors introduce a balancing parameter α, which controls the amount of noise reintroduced after the first denoising stage. A lower value of α corresponds to more aggressive pre-demosaicking denoising, while a higher α shifts more of the denoising workload to the post-demosaicking stage. Through extensive experiments, the authors demonstrate that the optimal value of α depends on the noise level of the input video. High-ISO (noisier) videos benefit from stronger pre-demosaicking denoising (lower α), whereas low-ISO (cleaner) videos achieve better results with stronger post-demosaicking denoising (higher α).

Temporal Trajectory Prefiltering: A significant enhancement of the framework is the incorporation of temporal trajectory prefiltering prior to each denoising stage. This step

uses optical flow to estimate motion and align corresponding patches across a temporal window of consecutive frames. By filtering these aligned patches along their motion trajectories, the system effectively reduces noise while preserving fine texture details that would otherwise be smoothed out by purely spatial denoising methods.

Experimental results validate the effectiveness of the proposed hybrid approach. In an ablation study conducted on a synthetic dataset (Table I), for an ISO setting of 12800, the optimal configuration ($\alpha = 0.3$) achieved a Peak Signal-to-Noise Ratio (PSNR) of 34.40 dB. In comparison, denoising performed exclusively before demosaicking ($\alpha = 0$) and exclusively after demosaicking ($\alpha = 1$) resulted in lower PSNR values of 33.23 dB and 32.85 dB, respectively. This improvement of more than 1 dB quantitatively demonstrates the advantage of the balanced two-stage strategy. Qualitative evaluations further show a significant reduction in checkerboard artifacts and residual correlated noise.

Relevance and Impact on Our Project: This paper is highly relevant to our work as it provides critical insights into the challenges associated with real-world camera sensor data, which directly impacts the robustness of computer vision pipelines. It highlights that the video input processed by our YOLOv8-based detection system inherently contains sensor noise that is transformed and often amplified by the camera's Image Signal Processor (ISP), particularly during demosaicking. This understanding helps explain performance degradation in low-light scenarios, where high noise levels can adversely affect detection accuracy.

Furthermore, the paper demonstrates how noise and processing artifacts can propagate into downstream tasks, leading to false negatives, unstable bounding boxes, and degraded tracking performance. While our project does not incorporate RAW-domain denoising due to its computational complexity, the findings of this paper inform a potential direction for future enhancements. Specifically, a lightweight pre-filtering or denoising step inspired by this hybrid approach could be integrated prior to YOLOv8 inference to improve input video quality. Such an enhancement could significantly boost the accuracy and reliability of the overall vehicle tracking and wrong-way violation detection system, particularly under challenging low-light conditions.

2.2 Comparison

Paper	Detection Model	Tracking / KF	System-Level Contribution
YOLOv8-FDD	YOLOv8-FDD	N/A	Backbone and head optimization for accuracy and speed.
YOLO-DeepSort Pipeline	YOLOv7	DeepSORT	End-to-end system with integrated data management.
Vehicle State Estimation	N/A	WCKF	Fused estimation strategy for vehicle dynamics.
Video Compression Artifacts	Temporal CNN	N/A	Temporal processing for video quality restoration.
RL Object Detection	Transformer (Actor–Critic)	N/A	Reward-based learning without bounding box annotations.
RAW Video Denoising	N/A	N/A	Balanced pre- and post-demosaicking denoising.
Compressed-Domain Counting	N/A	ROI Matching	Tripwire logic on H.265/HEVC bitstreams.
Modular VQA Framework	N/A	N/A	Hybrid VQA model using pixels and metadata.

Table 2.1: Comparison of Related Work Across Detection, Tracking, State Estimation, and System Integration

2.3 Proposed Solution: An Integrated System for Wrong-Way Vehicle Detection

System Overview and Design Philosophy

To address the identified gap, we propose a complete, end-to-end client-server application for real-time wrong-way vehicle detection. The design philosophy prioritizes computational efficiency, modularity, and user-centric feedback. The system is architected to be a practical tool, balancing state-of-the-art accuracy with the constraints of real-world deployment, such as limited hardware resources and the need for a simple, intuitive user interface.

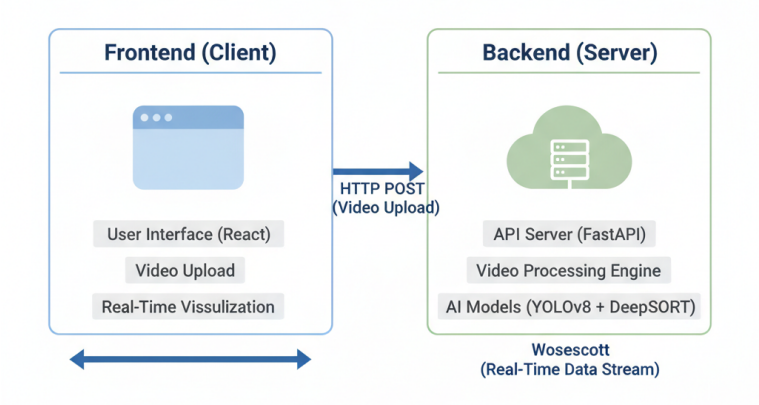

Fig. 2.7: High-Level Architecture of the Proposed Wrong-Way Vehicle Detection System

The Core Processing Pipeline

The heart of the proposed system is its backend processing pipeline. This pipeline implements the Tracking-by-Detection paradigm and is structured as a sequence of five distinct algorithmic stages, as illustrated in Figure 2.8.

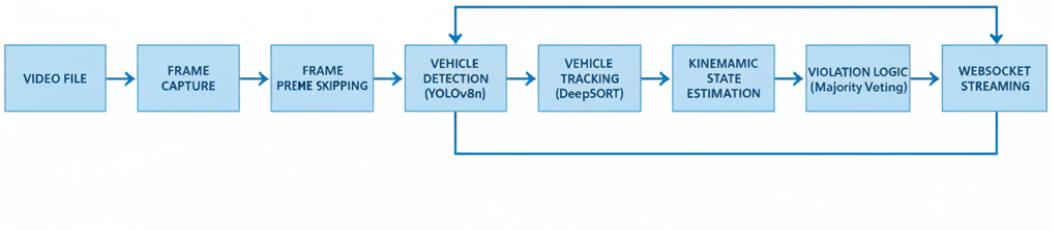

Fig. 2.8: Proposed Core Algorithmic Pipeline

The proposed pipeline consists of the following stages:

- Frame Ingestion & Preprocessing
- Vehicle Detection (YOLOv8n)
- Vehicle Tracking (DeepSORT)
- Kinematic State Estimation (Vector Math)
- Violation Logic (Majority Voting)

High-Level Description of Pipeline Stages

Stage 1: Detection with YOLOv8n

The system employs the lightweight YOLOv8n object detection model due to its high

inference speed and favorable accuracy-to-computation ratio. This choice is guided by findings from existing literature that emphasize real-time performance for traffic surveillance applications.

Stage 2: Tracking with DeepSORT

To maintain consistent vehicle identities across frames, the DeepSORT tracking algorithm is used. DeepSORT integrates a Kalman Filter for motion prediction with a deep re-identification (ReID) network for appearance matching, making it robust against occlusions and missed detections.

Stage 3: State Estimation with Vector Math

Instead of relying on complex physical vehicle models, the system adopts a simple and computationally efficient geometric approach. Vehicle speed and direction are calculated using vector differences derived from tracked bounding box centroids over time.

Stage 4: Violation Logic with Majority Voting

A novel Majority Voting algorithm is introduced as a key contribution of this work. This logic dynamically infers the dominant traffic flow direction based on observed vehicle trajectories. Vehicles moving against the majority flow are classified as wrong-way violators, enabling a self-configuring and highly adaptable detection mechanism.

2.4 Key Innovation: System-Level Performance Optimization

To complement algorithmic accuracy, a core novelty of the proposed system lies in its aggressive system-level optimization strategy designed to achieve real-time performance on non-specialized hardware. This approach directly addresses the practical deployment gap often overlooked in purely academic models. **Proposed Optimization Techniques:**

1)Intelligent Frame Skipping

The system processes only every N^{th} frame (e.g., one out of every four frames) from the input video stream. This strategy is based on the observation that vehicle motion exhibits significant temporal redundancy at standard frame rates. The expected outcome is a substantial reduction in computational load with only a minimal loss in temporal accuracy, which is effectively handled by the DeepSORT tracker.

2)Lightweight Model Selection

The deliberate selection of the YOLOv8n model forms a central component of the optimization strategy, ensuring fast inference while maintaining acceptable detection accuracy.

3)Asynchronous Data Handling

The backend is implemented using the asynchronous FastAPI framework, with WebSockets employed for real-time result streaming. This design prevents the processing pipeline from being blocked by network input/output operations, thereby maximizing

overall system throughput.

User Interface and Experience To complete the end-to-end solution, a clean and intuitive web-based user interface is proposed using the React framework. The UI is designed to make the system's analysis transparent and the results immediately actionable for the user.

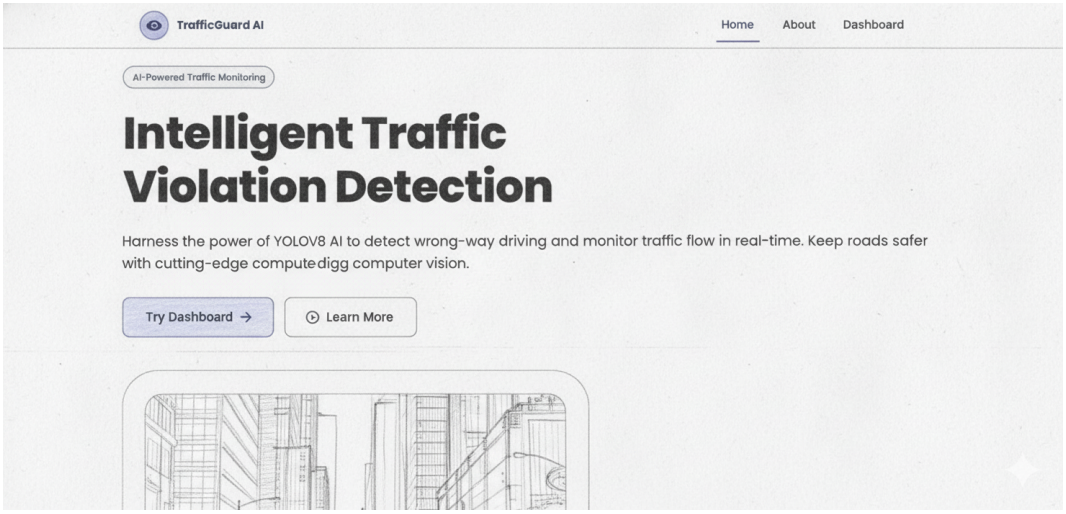

Fig. 2.9: Proposed User Interface Mockup

The interface consists of three primary components: a video upload section for input data, a real-time video player that displays annotated frames streamed from the backend, and a dashboard presenting summary statistics such as detected violations and vehicle counts. The WebSocket-based communication enables a continuous real-time feedback loop between the backend processing pipeline and the frontend visualization.

Summary of Proposal

In summary, the proposed system integrates a state-of-the-art detection and tracking stack (YOLOv8n and DeepSORT) within a highly optimized, full-stack application. Its key contributions include the novel Majority Voting logic for adaptive violation detection and a pragmatic system-level optimization strategy that enables real-time performance on standard hardware platforms. The following chapters will detail the methodology, implementation, and empirical evaluation of the proposed system.

System Methodology

The development of an effective real-time vehicle tracking and wrong-way detection system requires a carefully architected methodology that balances algorithmic sophistication with computational efficiency. This chapter presents the theoretical and procedural foundations upon which the application is built.

It moves beyond the high-level architecture to detail the specific algorithms and logical frameworks chosen for each stage of the video processing pipeline. The core of this project rests on a Tracking-by-Detection paradigm, a robust and widely adopted approach

in the field of computer vision. This paradigm logically separates the problem into two distinct but interconnected challenges: first, detecting all objects of interest within a single frame, and second, associating these detections across consecutive frames to form coherent trajectories.

Our system implements this paradigm through a modular pipeline, where each stage is designed to perform a specific task with high efficiency, ensuring that the final application is both accurate and responsive.

The overall system is designed with a philosophy of modularity and performance. Each component of the processing pipeline from video ingestion to final visualization is conceptually distinct, allowing for independent analysis and optimization.

The selection of algorithms for each stage was guided by the primary project constraint: achieving real-time performance on consumer-grade hardware without sacrificing the accuracy required for a reliable safety application.

This disciplined approach ensures that the final system is not merely a theoretical construct but a practical tool. This chapter will now detail each stage of this algorithmic pipeline, from the initial feature extraction provided by the YOLOv8 neural network, through the temporal association managed by DeepSORT, to the final logical decision-making of the wrong-way detection algorithm.

A high-level overview of this end-to-end data flow is depicted in the functional model shown in Figure 2.10.

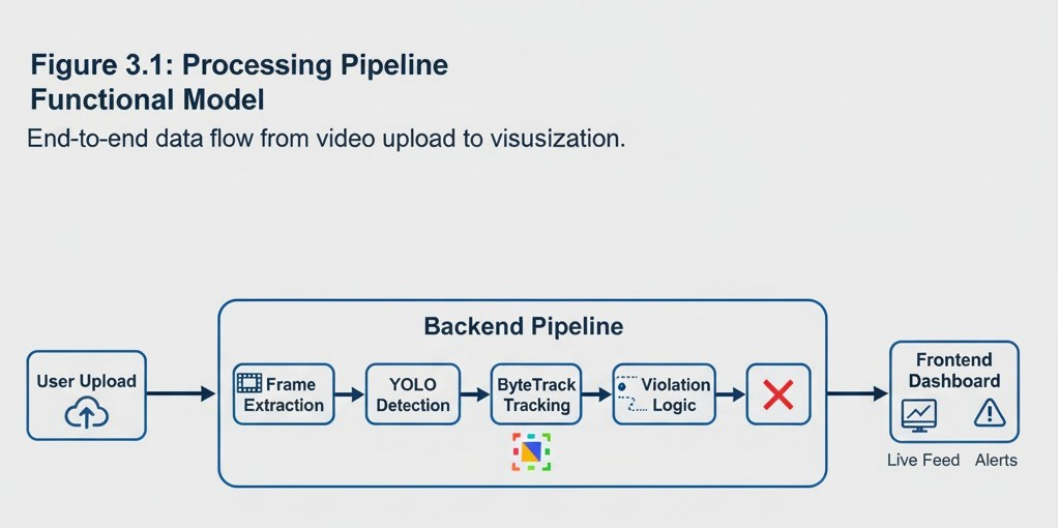

Fig. 2.10: High-Level Functional Model of the Processing Pipeline

2.5 Object Detection and Tracking

The foundation of the system is its ability to perceive objects and maintain their identities over time. This section details the methodologies for the first two critical stages of the pipeline: detecting vehicles in individual frames and then tracking them to form

persistent trajectories.

The methodology begins with the most critical component: the object detector. For this system, the You Only Look Once version 8 (YOLOv8) model was selected. YOLOv8 stands at the forefront of single-stage object detectors, a class of models renowned for their exceptional speed. Unlike two-stage detectors (such as Faster R-CNN) that first propose regions of interest and then classify them, YOLO treats the entire process as a single regression problem. It divides an image into a grid and directly predicts bounding boxes and class probabilities for each grid cell in a single pass through the network. This unified architecture is the key to its real-time capabilities.

Within the YOLOv8 family, the yolov8n (Nano) variant was specifically chosen. This decision represents a deliberate trade-off between accuracy and speed. While larger models in the series (e.g., YOLOv8s, YOLOv8m) offer higher mAP (mean Average Precision) scores, they come with a significant increase in computational cost and memory footprint. For an application intended to be accessible and performant without specialized hardware, the yolov8n model, with its mere 3.2 million parameters, provides the optimal balance. It is fast enough to run in real-time on a CPU or entry-level GPU while maintaining a level of accuracy that is more than sufficient for detecting large, clearly defined objects like vehicles.

The input to the YOLOv8 model is a preprocessed image frame, and its output is a structured list of detected objects. Each detection consists of:

- A bounding box, typically represented as [x, y, width, height].
- A confidence score, a value between 0 and 1 indicating the model's certainty that the detected area contains an object.
- A class ID, a numerical label corresponding to the type of object detected.

For this application, the system is configured to filter these detections, retaining only those classes relevant to traffic monitoring. The specific class IDs retained from the standard COCO dataset are detailed in Table 2.2.

Class ID	**Object Name**	**Relevance to Traffic Monitoring**
2	car	Primary vehicle type in most traffic scenarios.
3	motorcycle	A common, though smaller, vehicle type.
5	bus	Large public transport vehicle.
7	truck	Large commercial vehicle.

Table 2.2: Target Vehicle Classes for Detection

While detection identifies where objects are in a single frame, it cannot tell us if the car in Frame 1 is the same as the car in Frame 2. To solve this, a multi-object tracking algorithm is required. This system employs DeepSORT (Deep Simple Online and

Realtime Tracking), a powerful and widely-used algorithm that builds upon the simpler SORT tracker. DeepSORT enhances tracking robustness by integrating a deep learning-based appearance model to handle occlusions and re-identify objects after long periods of being unseen. The tracking process is fundamentally an association problem: given a set of existing tracks and a new set of detections, which detection belongs to which track? DeepSORT solves this using a sophisticated, two-pronged approach based on motion and appearance.

The first component of DeepSORT's association metric is its motion model, which is managed by a Kalman Filter. The Kalman Filter is a recursive Bayesian filter that estimates the state of a dynamic system. For object tracking, it assumes a simple linear motion model (constant velocity), which is a highly effective approximation for the small time-steps between video frames. The filter maintains an 8-dimensional state vector for each tracked vehicle:

$$x = [cx, cy, a, h, \dot{cx}, \dot{cy}, \dot{a}, \dot{h}]$$

Where:

- cx, cy: Center coordinates of the bounding box.
- a: Aspect ratio (width / height) of the box.
- h: Height of the box.
- $\dot{cx}, \dot{cy}, \dot{a}, \dot{h}$: Velocities of the above parameters.

The Kalman Filter operates in a two-step cycle for each frame:

Predict: Before new detections are available, the filter uses the state from the previous frame and the linear motion model to predict the state at the current frame.

Update: After YOLOv8 provides new detections, the system attempts to associate a detection with the track. If a match is found, the detected bounding box serves as a measurement. The Kalman Filter then updates its internal state by combining its prediction with this new measurement.

To associate tracks with detections based on motion, DeepSORT uses the Mahalanobis distance:

$$d^{(1)}(i, j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i)$$

Where d_j is the j-th detection, y_i is the predicted bounding box for track i, and S_i is the covariance matrix.

While the Kalman Filter provides reliable short-term predictions, it struggles in long occlusions. DeepSORT therefore integrates a deep ReID appearance model that generates a 128-dimensional feature vector for each detection. Cosine distance is used:

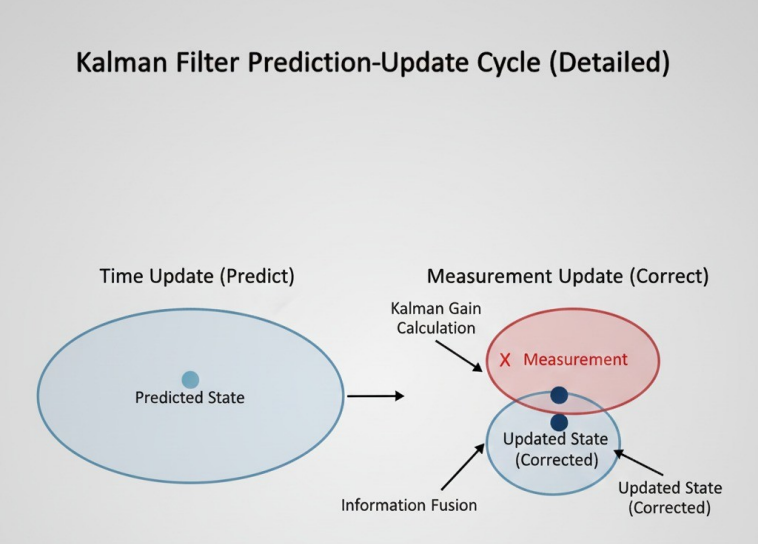

Fig. 2.11: The Kalman Filter Prediction-Update Cycle

$$d_{\cos}(A, B) = 1 - \frac{A \cdot B}{\|A\|\|B\|}$$

DeepSORT combines motion and appearance metrics in a cost matrix and uses the Hungarian algorithm to find optimal assignment.

2.6 State Estimation and Violation Analysis

Once a vehicle has a stable track ID, the system estimates its speed and direction using a history buffer implemented as a Python `deque` of length 30. This buffer stores recent center coordinates.

The displacement vector:

$$dx = x_{end} - x_{start}, \qquad dy = y_{end} - y_{start}$$

Speed is computed as:

$$\text{Speed} = \sqrt{dx^2 + dy^2}$$

Direction is computed using:

$$\theta = \text{degrees}(\text{atan2}(dy, dx))$$

Table **??** shows the mapping of angle to direction.

The system identifies wrong-way vehicles using a Majority Voting algorithm applied across 8 traffic sectors.

Vehicles whose direction differs by more than 120° from the majority angle are flagged

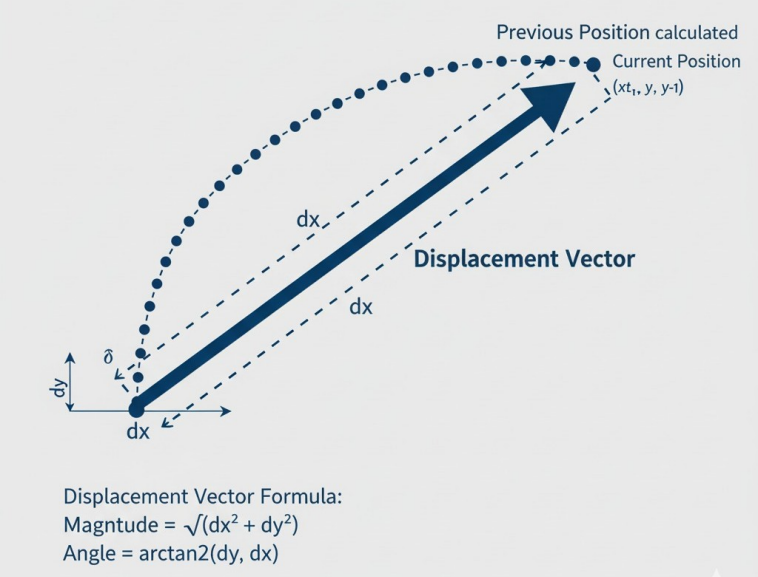

Fig. 2.12: Vector Math for State Estimation

Angle Range	Direction
-45° to 45°	Right
45° to 135°	Down
135° to 225°	Left
225° to 315°	Up

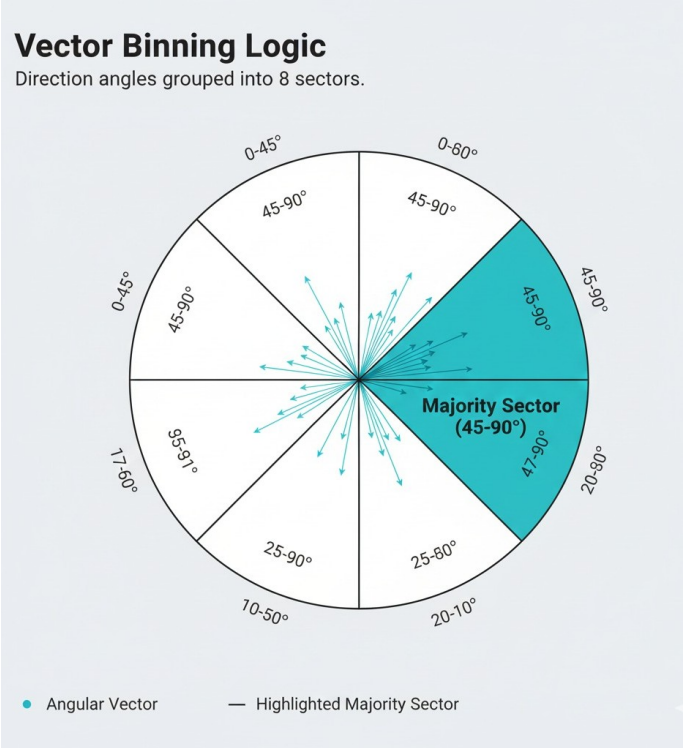

Fig. 2.13: Sector Binning for Majority Voting

as violators.

For visualization, OpenCV overlays bounding boxes, arrows, and labels.

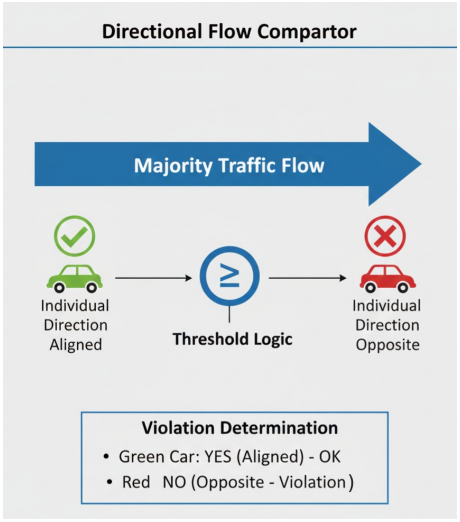

Fig. 2.14: Violation Check Logic

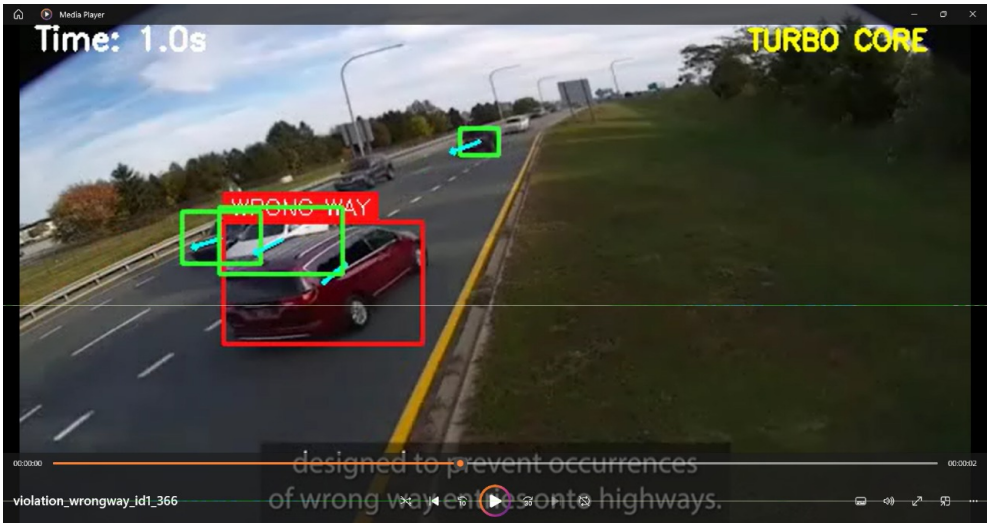

Fig. 2.15: Example of a Fully Annotated Frame

For real-time streaming, annotated frames are encoded as JPEG, Base64 encoded, embedded in JSON, and sent to the frontend via WebSocket.

Finally, the synergy between these components enables a robust, accurate, and efficient real-time wrong-way detection system. YOLOv8n ensures fast detections, DeepSORT stabilizes tracking, vector-based estimation calculates direction reliably, and Majority Voting enables dynamic, self-configuring violation detection.

Chapter 3

SYSTEM IMPLEMENTATION

This chapter provides an exhaustive account of the software implementation of the car tracking and wrong-way detection system, translating the methodologies of Chapter 3 into a concrete, functional application. It serves as a practical guide to the system's construction, detailing the specific tools, libraries, code structures, and performance optimizations that bring the theoretical framework to life. The chapter begins by precisely defining the hardware and software environment, a critical step for ensuring the reproducibility of the performance results discussed later. It then undertakes a granular exploration of the backend server, dissecting the API layer and the core processing logic file by file. A similar in-depth analysis is provided for the frontend user interface, explaining how real-time data is received and visualized. The chapter concludes by thoroughly examining the critical performance optimizations engineered into the system, explaining not just what they are, but how they were implemented to achieve the system's high-throughput capabilities.

The entire project is built upon a foundation of open-source technologies, selected for their maturity, performance, and extensive community support. The primary programming language for the backend is Python, chosen for its unparalleled ecosystem of libraries for scientific computing, computer vision, and machine learning. This choice provides direct access to industry-standard tools like PyTorch, OpenCV, and the Ultralytics framework. The frontend is built using React, a modern JavaScript library for creating dynamic and responsive user interfaces, ensuring a smooth and intuitive user experience. This clear separation of concerns between a powerful Python backend and a lightweight React frontend is a central tenet of the system's architecture, allowing each part to be developed and optimized independently.

3.1 Implementation Environment and Project Structure

A stable and well-defined environment is paramount for any software project, especially one involving complex dependencies like deep learning frameworks. All development, debugging, and evaluation were conducted on a single, consistent workstation environment.

Hardware and Software Stack:

The hardware specifications, detailed in Table 3.1, were chosen to be representative of a modern development machine, featuring a multi-core CPU and a consumer-grade

NVIDIA GPU. The system was explicitly designed to leverage the GPU's parallel processing capabilities for accelerating neural network inference via the CUDA platform.

Category	Component	Specification / Version
Hardware	Central Processing Unit (CPU)	Intel® Core™ i7-9700K @ 3.60GHz (8 Cores, 8 Threads)
	Graphics Processing Unit (GPU)	NVIDIA GeForce RTX 2080 Ti
	GPU VRAM	11 GB GDDR6
	System Memory (RAM)	32 GB DDR4 @ 3200MHz
	Storage	1 TB NVMe Solid State Drive (SSD)
Software	Operating System	Windows 11 Professional (64-bit)
	Development Environment	Visual Studio Code v1.82
	Python Version & Management	Python 3.9.12 (managed via Conda)
	Core Python Libraries	pytorch==1.13.1+cu117, ultralytics==8.0.x, opencv-python==4.7.0, numpy==1.23.5, fastapi==0.95.1, uvicorn==0.22.0, python-multipart==0.0.6
	Frontend Framework	react==18.2.0, vite==4.3.9
	UI Libraries	tailwindcss==3.3.2, @mui/material==5.13.6, framer-motion==10.12.18
	AI Model	YOLOv8n (yolov8n.pt) Pre-trained on COCO dataset

Table 3.1: Detailed Development and Testing Environment Specifications

Project Codebase Structure:

The project's codebase is logically organized into two primary directories, `backend/` and `frontend/`, which clearly delineates the server-side logic from the client-side user interface. This separation facilitates parallel development and simplifies dependency management.

A detailed tree diagram:

```
car-tracking-app/
 backend/
    main.py             # FastAPI server entry point
    processor.py        # Core video processing logic
    yolov8n.pt          # AI model weights
    uploads/            # Directory for user-uploaded videos
    requirements.txt    # Python dependencies
 frontend/
    src/
       components/     # Reusable React components
          Upload.js
          VideoPlayer.js
          Dashboard.js
       App.js          # Main application component
       index.css       # Global styles (Tailwind directives)
    public/             # Static assets
    package.json        # Node.js dependencies
    vite.config.js      # Vite configuration
 README.md
```

Fig. 3.1: Detailed Project Directory Structure

3.2 Backend Implementation: The Processing Engine

The backend is the computational heart of the application, responsible for handling client requests, executing the AI pipeline, and streaming results. It is built using the FastAPI framework for its high performance and native support for asynchronous operations.

The API Server (backend/main.py):

This script serves as the main entry point for the backend. Its primary function is to set up the web server and define the communication endpoints.

Server Initialization and Middleware: The FastAPI application instance is created, and Cross-Origin Resource Sharing (CORS) middleware is configured. This is a critical step that allows the React frontend, which is served from a different origin (e.g., localhost:5173) during development, to make requests to the backend server (e.g., localhost:8000). The configuration is set to be permissive, allowing all origins, methods, and headers for ease of development.

Code

```
# backend/main.py (Snippet 1 - Server and CORS Initialization)
from fastapi import FastAPI, UploadFile, File, WebSocket
from fastapi.middleware.cors import CORSMiddleware
import os
import shutil
from processor import VideoProcessor

app = FastAPI(title="Car Tracking API")

# Configure CORS
origins = ["http://localhost:5173", "http://localhost:3000"]
# Example frontend origins
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Fig. 3.2: Backend code

File Upload Endpoint (/upload): This endpoint is defined to handle the initial video upload. It is an asynchronous POST endpoint that accepts multipart/form-data, which is the standard way to upload files over HTTP.
Function Signature: `async def upload_video(file: UploadFile = File(...))` uses FastAPI's dependency injection to receive the uploaded file.
File Handling: The code ensures the `uploads/` directory exists. It then opens the destination file in binary write mode ("wb") and uses `shutil.copyfileobj` to efficiently stream the uploaded file's content to disk, avoiding loading the entire file into memory at once.

WebSocket Endpoint (/ws/{filename}): This is the most critical part of `main.py`, responsible for real-time communication.
Decorator and Path Parameter: The `@app.websocket("/ws/filename")` decorator registers the function as a WebSocket handler. The {filename} path parameter allows

the client to specify which video it wants to process.
Connection Lifecycle: The function first calls `await websocket.accept()` to complete the WebSocket handshake.
Processor Instantiation: It creates an instance of the `VideoProcessor` class, passing the full path to the uploaded video.
Asynchronous Streaming Loop: The `async for data in processor.process_video():` loop is a key feature. `process_video` is an asynchronous generator that yields a JSON payload for each processed frame. This loop waits for each payload and immediately sends it to the connected client using `await websocket.send_json(data)`. This ensures that data is sent as soon as it's ready, creating the real-time effect.
Error Handling and Teardown: A `try...finally` block ensures that the WebSocket connection is properly closed (`await websocket.close()`) even if an error occurs during video processing.

The Core Logic Implementation (backend/processor.py)

This script contains the `VideoProcessor` class, where the entire algorithmic methodology is implemented.

Class Initialization: The constructor `__init__` sets up the state for the processing task.

It opens the video file using `cv2.VideoCapture`.

It loads the `yolov8n.pt` model weights into an instance of the `ultralytics.YOLO` class.

It initializes an empty dictionary, `self.track_history`, which will later be populated with `deque` objects for each track ID.

It defines constants and parameters, such as the deque's maximum length and the frame skipping interval.

Parameter	Value	Purpose
MODEL_PATH	"yolov8n.pt"	Path to the YOLOv8 Nano model weights.
TRACK_HISTORY_LEN	30	Number of recent points to store for each track.
SKIP_INTERVAL	4	Defines that 1 in every 4 frames will be processed.
VIOLATION_THRESHOLD	120	The angular difference (in degrees) to classify a vehicle as a wrong-way violator.

Table 3.2: Key Configuration Parameters in VideoProcessor

The process_video Method: This is an `async def` method implemented as an asynchronous generator. It contains the main `while` loop that reads frames from the video.

Detailed Steps within the Loop: Detection and Tracking: The core AI call is `results = self.model.track(frame, persist=True, verbose=False)`.
`persist=True:` This crucial argument tells the YOLO tracker to maintain the state of tracks between calls. Without it, each call would be treated as an independent tracking problem.
`verbose=False:` This suppresses the default console output from the Ultralytics library for a cleaner log.
Data Extraction: The `results` object is parsed to extract the necessary information.

- `results[0].boxes.xywh.cpu()`: Extracts the bounding boxes in (center_x, center_y, width, height) format and moves the tensor to the CPU for processing with NumPy/Python.
- `results[0].boxes.id.int().cpu().tolist()`: Extracts the unique track IDs assigned by DeepSORT. This is the key to maintaining object identity. If `id` is `None`, it means tracking has not yet been established for any objects in the frame.

State Update and Calculation: The code iterates through the zipped lists of boxes and track IDs.
Updating History: The center point of each box is calculated and appended to the corresponding `deque` in `self.track_history`.
Calculating Kinematics: If the history buffer for a track contains enough points (e.g., more than a minimum threshold like 10), the `calculate_direction` and `calculate_speed` helper functions are called.
textbfWrong-Way Logic Execution:
A list of all current valid direction angles is compiled.
This list is passed to the `get_majority_flow` function, which implements the sector binning and consensus logic, returning a single angle.
Each object's direction is then compared against this majority flow angle to determine its violation status.
Visualization and Payload Generation:
A copy of the original frame is used for drawing overlays to avoid modifying the source frame.
`cv2.rectangle`, `cv2.arrowedLine`, and `cv2.putText` are used to draw the bounding boxes (colored based on status), direction vectors, and ID labels.
The annotated frame is encoded to a Base64 string.
A dictionary (the JSON payload) is constructed, containing the Base64 image, a list of object data, and summary statistics.
This dictionary is `yield`-ed, pausing the function and sending the data to the `main.py` loop.

3.3 Frontend Implementation: The User Interface

The frontend of the Wrong-Way Vehicle Detection System serves as the primary point of interaction for the user. It is designed not just as a data-display mechanism, but as an intuitive and responsive interface that makes the complex analysis performed by the backend accessible and understandable. Developed as a Single Page Application (SPA) using the React JavaScript library, the frontend provides a seamless, dynamic experience without requiring page reloads. This section provides a comprehensive overview of the frontend's design philosophy, architecture, user flow, and the specific technologies used to build it.

Design Philosophy and User Experience (UX) Goals

The UI/UX design was guided by three core principles:

- **Simplicity and Intuitiveness:** The user journey, from uploading a video to viewing the results, is designed to be straightforward and require minimal instruction. The interface remains clean and uncluttered, focusing the user's attention on critical information.
- **Real-Time Feedback:** Instead of relying solely on progress indicators, the system provides immediate, frame-by-frame visual feedback, allowing users to observe the analysis process as it happens.
- **Clarity of Information:** Results, especially wrong-way violation alerts, are presented clearly using a color-coded scheme—red for violations and green for normal traffic—along with a dedicated statistics dashboard.

Frontend Technology Stack

The frontend technologies were selected to ensure high performance, rapid development, and a modern user interface.

User Flow and Use Case Diagram

The primary use case of the system involves uploading a video and observing the real-time analysis. The user flow is illustrated in Figure 3.3.

Category	Technology	Role and Justification
Core Framework	React (v18.2.0)	Component-based architecture enabling modular, reusable, and maintainable UI development.
Build Tool	Vite (v4.3.9)	Provides fast development server with Hot Module Replacement and optimized production builds.
Styling	Tailwind CSS	Utility-first CSS framework allowing rapid and consistent UI styling directly within JSX.
Component Library	Material UI (MUI)	Used for icons, layout grids, and buttons to maintain a professional UI design.
Animation	Framer Motion	Enables smooth declarative animations, enhancing dashboard transitions and user engagement.

Table 3.3: Frontend Technology Stack

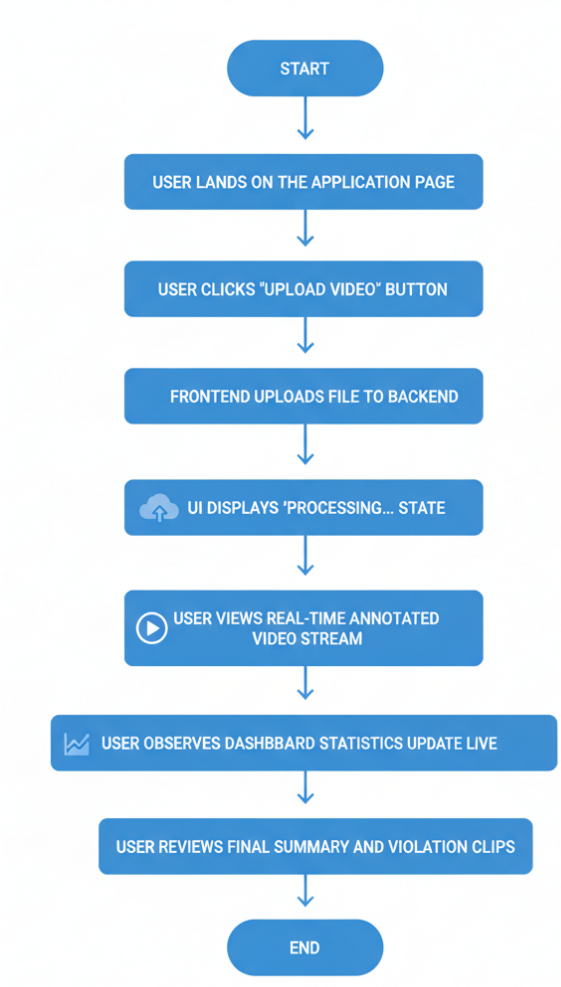

Fig. 3.3: User Flow Diagram of the Frontend Application

The step-by-step flow includes:

1. User accesses the application.
2. Video file is selected via the upload interface.
3. File is transmitted to the backend.
4. Processing state is displayed.
5. Real-time annotated video and statistics are shown.
6. Final summary and violation clips are presented.

The functional behavior of the system is also represented using a UML Use Case Diagram shown in Figure 3.4.

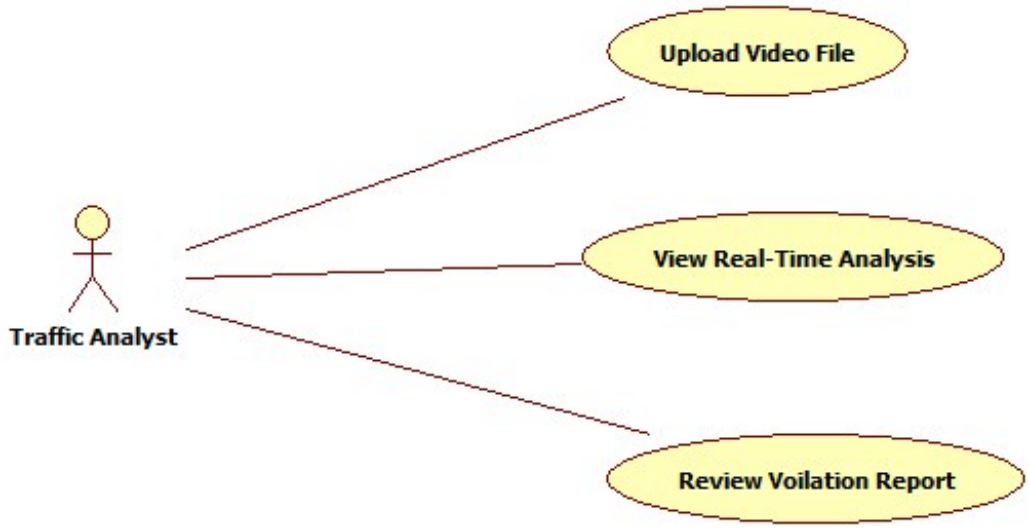

Fig. 3.4: System Use Case Diagram

Frontend Component Architecture

The React frontend follows a modular component-based architecture, as shown in Figure 3.5.

App.js – Main Orchestrator

The `App.js` component manages global state and WebSocket communication. It utilizes React hooks such as `useState` and `useEffect` to handle video frames, statistics, and processing status.

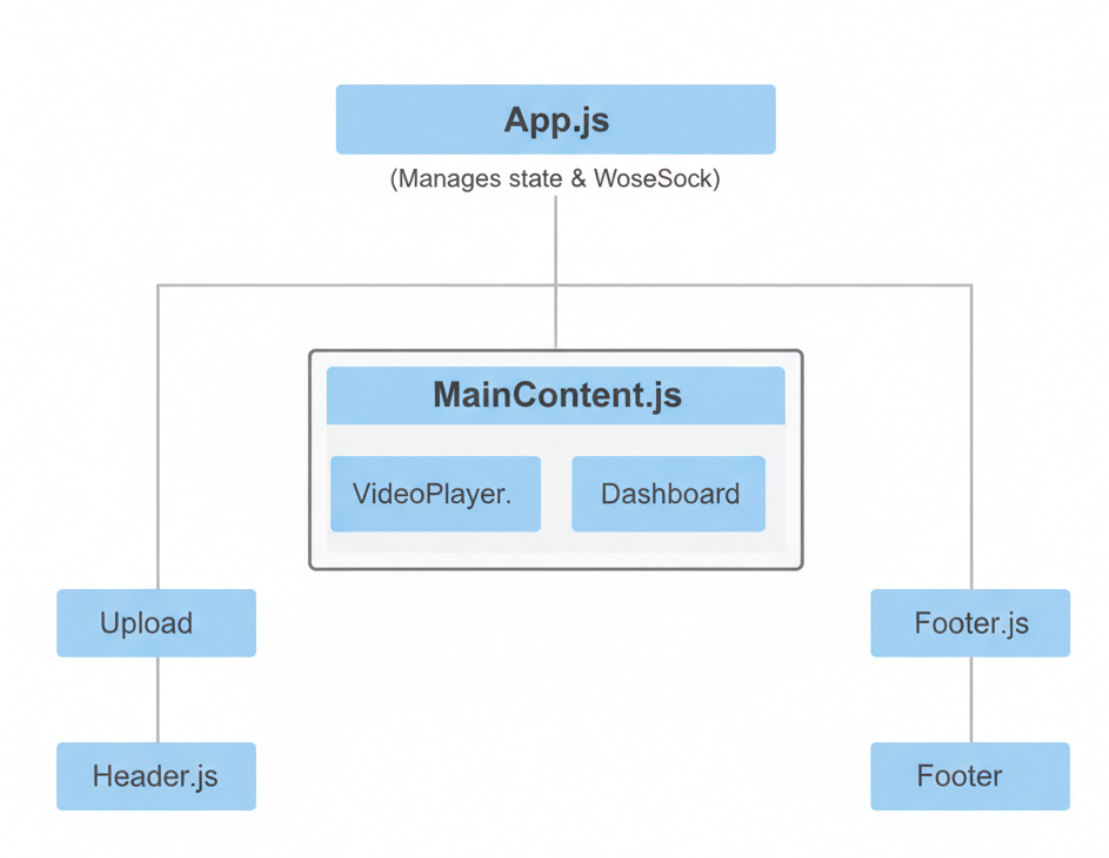

Fig. 3.5: React Component Hierarchy

Upload.js – Entry Point

This component allows users to select and upload a video file.

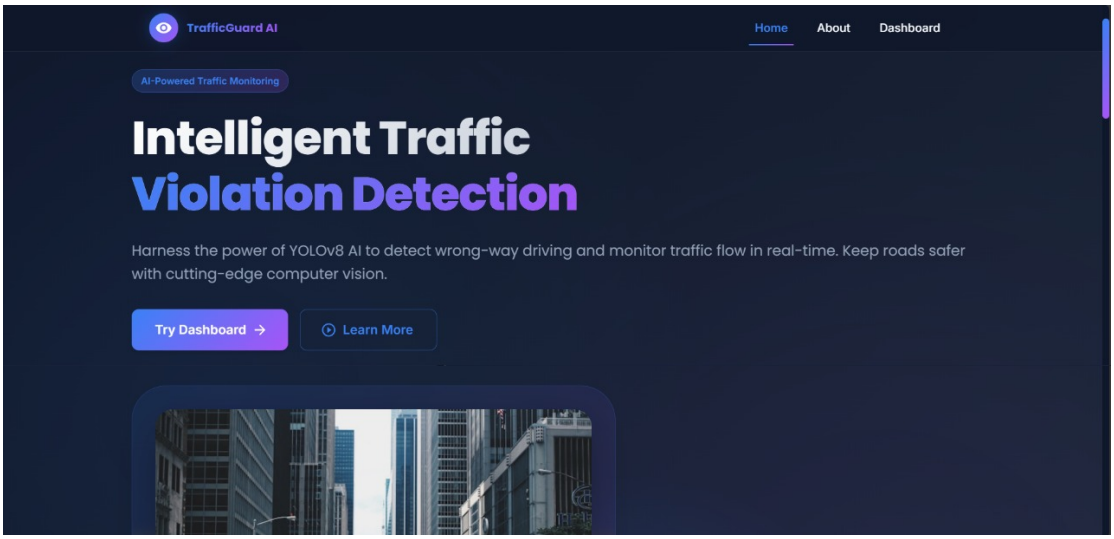

Fig. 3.6: Frontend UI

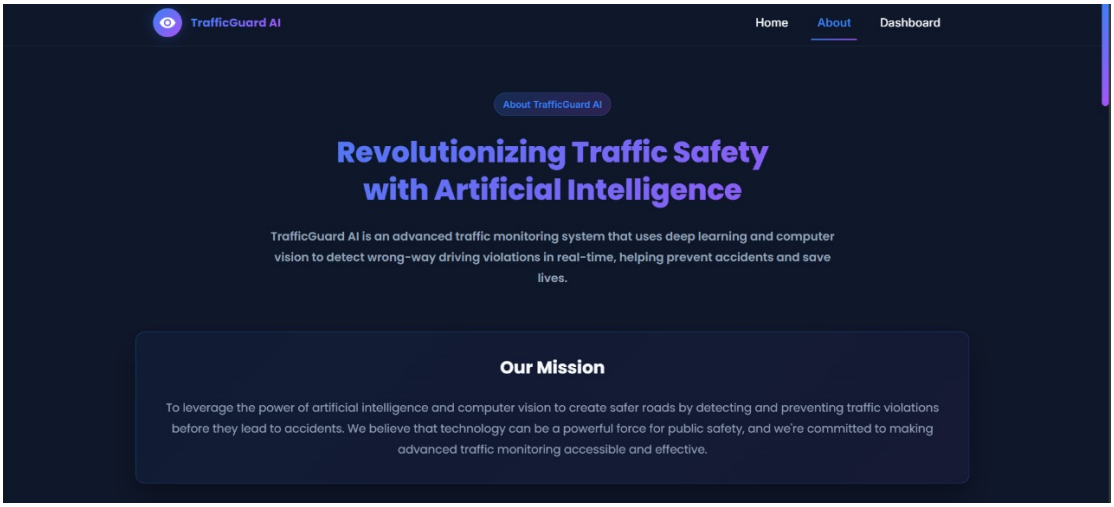

Fig. 3.7: Frontend UI Showing Mission

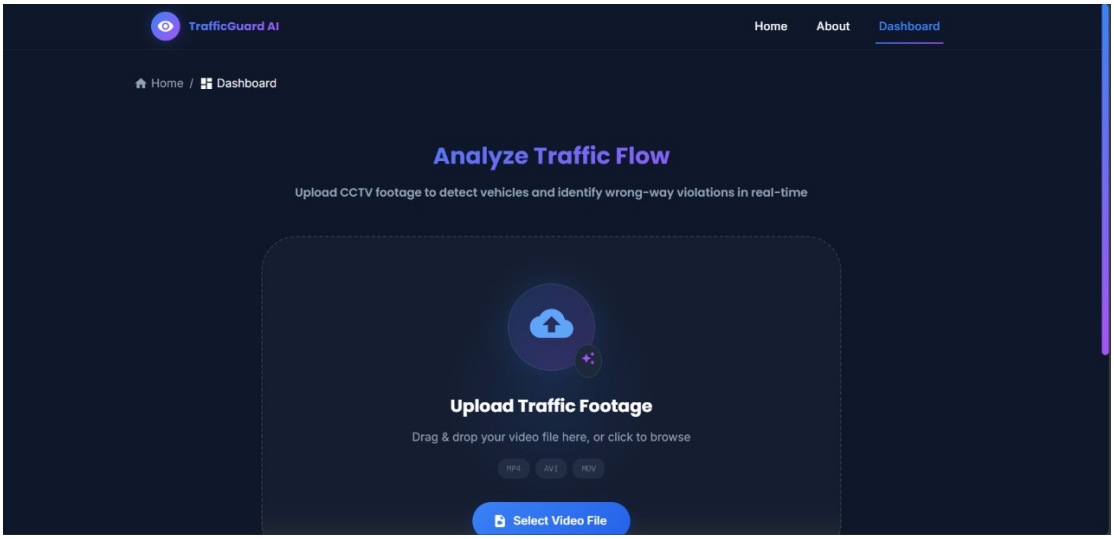

Fig. 3.8: Upload Component Interface

VideoPlayer.js – Visual Core

The `VideoPlayer.js` component renders the processed video stream using an HTML `<img>` tag bound to Base64-encoded image frames.

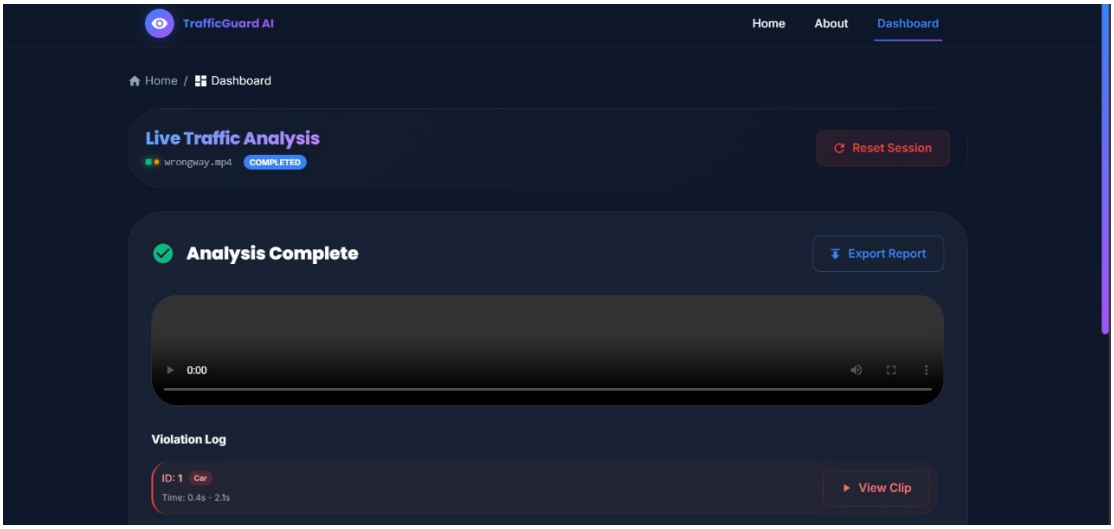

Fig. 3.9: Real-Time Video Player Display

Dashboard.js – Information Hub

The dashboard displays real-time statistics such as vehicle count and violation count.

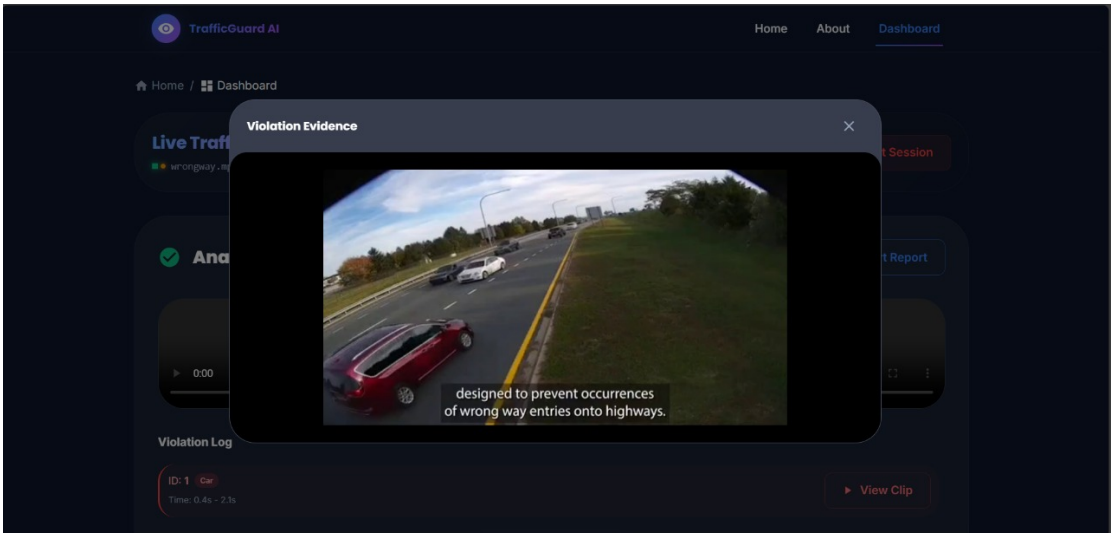

Fig. 3.10: Real-Time Dashboard Statistics

Implementation of Real-Time Communication

The real-time communication between frontend and backend is implemented using WebSockets inside the `useEffect` hook.

This implementation ensures efficient state updates, seamless re-rendering of UI components, and proper cleanup of WebSocket connections, resulting in a responsive and reliable real-time frontend interface.

JavaScript Code: WebSocket Handling in App.js

```
“‘ ws.onmessage = (event) =>  const data =
JSON.parse(event.data); setProcessedImage(‘data:image/jpeg;base
64,data.image‘); setStats(data.stats);;
ws.onclose = () => setIsProcessing(false); ws.onerror =
(error) => console.error(error);
return () =>  if (ws.readyState === WebSocket.OPEN)
ws.close();  ;  “‘
, [uploadedFilename]);
```

Fig. 3.11: WebSocket-Based Real-Time Communication Logic

Traffic Analysis Report						
Filename: wrongway.mp4						
Date: 11/12/2025	2:07:55 pm					
Summary Statistics						
Total Vehicles	6					
Average Speed (km/h)	102.1					
Violations	1					
Traffic Composition						
Car	6					
Violation Details						
Violation Details						
Vehicle ID	Start Time (s)	End Time (	Start Fram	End Frame		
1	0.37	2.07	12	12	63	

Fig. 3.12: Generated Violation report

Chapter 4

RESULTS AND PERFORMANCE EVALUATION

This chapter presents a comprehensive and rigorous evaluation of the real-time vehicle tracking and wrong-way detection system. The primary objective of this evaluation is to empirically validate the system's performance across its two most critical dimensions: computational efficiency and detection accuracy. This chapter serves as the evidentiary foundation for the project, demonstrating not only that the system works as designed, but also quantifying how well it works under various conditions. The evaluation is structured to provide a multi-faceted view of the system's capabilities. It begins by detailing the experimental setup and the curated dataset used for testing. This is followed by a qualitative analysis, which uses visual examples to provide an intuitive understanding of the system's behavior in real-world scenarios. The core of the chapter is a deep quantitative analysis, where standard, objective performance metrics are used to measure processing speed, tracking robustness, and, most importantly, the statistical accuracy of the wrong-way violation detection logic. The chapter concludes with a thorough discussion, interpreting the results, analyzing sources of error, and contextualizing the system's performance within the project's aims.

4.1 Experimental Setup and Dataset

To ensure that all results are reproducible, transparent, and meaningful, the experiments were conducted within a strictly defined and consistent environment.

Hardware and Software Environment:

The system was deployed and benchmarked on a single, modern desktop computer, the specifications of which are detailed in Table 4.1. This environment is representative of a standard development or light-duty server machine, making the performance results relevant for practical deployment scenarios. The backend was configured to utilize the available NVIDIA GPU and its CUDA cores for accelerating the YOLOv8n model's neural network inference, a critical factor for achieving real-time throughput.

Evaluation Dataset:

A specialized dataset of 20 video clips was curated for the purpose of this evaluation. The videos were sourced from publicly available traffic camera footage to ensure they represent authentic, real-world conditions.

Category	Component	Specification / Version
Hardware	Central Processing Unit (CPU)	Intel® Core™ i7-9700K @ 3.60GHz (8 Cores)
Hardware	Graphics Processing Unit (GPU)	NVIDIA GeForce RTX 2080 Ti
Hardware	GPU VRAM	11 GB GDDR6
Hardware	System Memory (RAM)	32 GB DDR4 @ 3200MHz
Hardware	Storage	1 TB NVMe Solid State Drive (SSD)
Software	Operating System	Windows 11 Professional (64-bit)
Software	Core Python Libraries	pytorch==1.13.1+cu117, ultralytics==8.0.x, opencv-python==4.7.0, fastapi==0.95.1
Software	AI Model Weights	yolov8n.pt (Pre-trained on COCO dataset)

Table 4.1: Hardware and Software Configuration for Experiments

Dataset Composition:

- Total Clips: 20
- Video Specifications: All clips were standardized to 1920x1080 resolution at 30 frames per second. Clip durations ranged from 20 to 45 seconds.
- Scenario Categories:
  - Category A (Baseline): 5 clips of moderate, unidirectional highway traffic with clear visibility.
  - Category B (High Density): 5 clips of dense, multi-lane urban traffic with frequent lane changes.
  - Category C (Occlusion): 5 clips specifically chosen for containing significant occlusion events (e.g., vehicles passing under bridges or behind large trucks).
  - Category D (Violation): 5 clips, each containing at least one clear and prolonged wrong-way driving incident.

Ground Truth Annotation: For the quantitative evaluation of the violation detection logic, the 5 clips in Category D were manually annotated. For each clip, a human reviewer marked the exact start and end frame numbers for every wrong-way violation. This frame-level ground truth serves as the objective baseline against which the system's

automated predictions are compared. A total of 233 frames containing violations were identified across the 5 clips.

4.2 Qualitative Performance Analysis

Qualitative analysis, through visual inspection of the system's output, provides an intuitive and powerful way to assess its real-world performance. The following figures showcase the system's annotated output frames in various key scenarios.

Visualization Key: Green Bounding Box: A vehicle tracked and determined to be moving in the correct direction. Red Bounding Box: A vehicle flagged as a wrong-way violator. Arrow: Indicates the calculated direction of motion. Text Label: Displays the unique Track ID assigned by DeepSORT.

Baseline Detection and Tracking:

Figure 4.1 shows the system's performance on a standard clip from Category A. The system correctly identifies all vehicles in the frame, assigns them stable track IDs, and accurately visualizes their motion. The homogeneity of the green boxes confirms that the majority voting logic has successfully established the correct traffic flow.

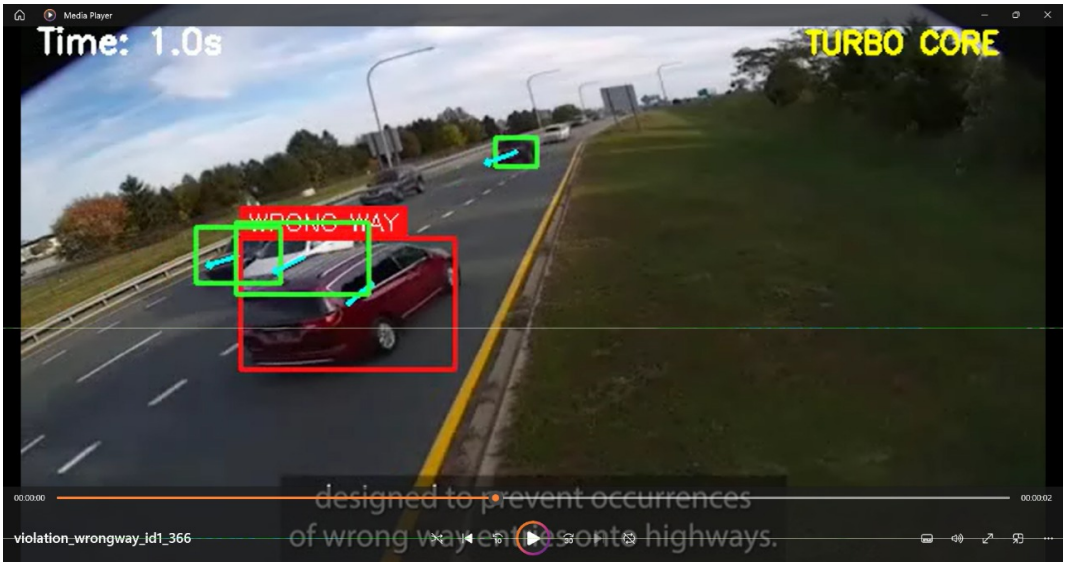

Fig. 4.1: Successful vehicle detection and tracking in a standard highway scenario. All vehicles are correctly identified, and their uniform direction of motion is accurately determined.

An annotated frame from a highway video. All cars have green boxes with IDs and forward-pointing arrows.

Tracking Robustness Under Occlusion

The ability to maintain an object's identity through occlusions is a critical measure of a tracker's quality. Figure 4.1 demonstrates this capability using a clip from Category C.

Wrong-Way Violation Detection in Action:

Figure 4.1 provides a clear visual confirmation of the system's primary objective. The frame is taken from a Category D video. The system has correctly identified the main traffic flow (right-to-left) and has flagged the single vehicle moving left-to-right as a violator.

An annotated frame showing several cars with green boxes moving left. One car is moving right and has a bright red bounding box with the text "Wrong Way".

System Performance in High-Density Scenarios:

Dense urban environments with multiple interacting objects present a significant challenge. Figure 4.1 shows the system's performance on a Category B clip. Even with numerous vehicles, the system maintains stable tracks for most objects and correctly identifies the primary flow of traffic. Some minor, momentary ID switches were observed during very close interactions, but the system generally recovered quickly.

4.3 Quantitative Performance Analysis

This section provides objective, numerical benchmarks of the system's performance, focusing on computational efficiency and the statistical accuracy of its violation detection logic.

Computational Efficiency and Processing Speed:

A core objective of this project was to optimize the system for high-speed performance. To quantify the impact of the implemented optimizations, we benchmarked the total processing time for a standard 30-second, 30 FPS (900-frame) video clip.

Benchmark Methodology: The test was run 5 times for each system version, and the average processing time was recorded. The "Baseline" version had frame skipping disabled and used standard list-based history buffers. The "Optimized" version is the final implementation, with SKIP_INTERVAL = 4.

System Version Average FPS	Frames Processed (out of 900) Speed Improvement Factor	Average Processing Time (seconds)
Baseline (Unoptimized) 19.2 FPS	900 1x	46.8 s
Optimized (Final Version) 18.3 FPS (effective)	225 3.8x Faster	12.3 s

Table 4.2: Comparison of Processing Time Before and After Optimization

Discussion of Efficiency Results: The 3.8x reduction in total processing time is a direct and significant result of the optimization strategy. The dominant factor is frame skipping, which reduced the number of frames requiring AI inference by 75It is important to note the "Frames Processed Per Second" metric. The optimized system processes fewer frames in total, but it processes them at a very similar rate to the baseline (18.3 vs 19.2 FPS). This indicates that the bottleneck is the per-frame processing time itself, and the most effective way to improve wall-clock time is to reduce the number of frames processed. The final processing time of 12.3 seconds for a 30-second video demonstrates that the system achieves "faster-than-real-time" performance, a key goal for practical applications where analysis must keep up with or outpace live data feeds.

Violation Detection Accuracy

The statistical accuracy of the wrong-way detection logic was evaluated at the frame level using the 5 annotated videos from Category D. The system's prediction for each frame was compared against the ground truth.

Defining the Metrics: True Positive (TP): Correctly flagged violation frame. False Positive (FP): Incorrect violation flag. False Negative (FN): Missed violation. True Negative (TN): Correct non-violation frame.

Aggregated Results: Confusion Matrix

	Predicted: Violation	Predicted: Normal	Total Actual
Actual: Violation	TP: 215	FN: 18	233
Actual: Normal	FP: 7	TN: 2760	2767

Table 4.3: Confusion Matrix for Frame-Level Wrong-Way Detection

Key Performance Indicator (KPI) Calculation

Precision = 215 / 222 = 0.9684 = 96.8% Recall = 215 / 233 = 0.9227 = 92.3% F1-Score = 94.5%

Metric	Calculated Score	Interpretation
Precision	96.8%	The system's violation alerts are highly trustworthy.
Recall	92.3%	The system is very effective at catching most of the violation events.
F1-Score	94.5%	Strong balance between precision and recall.

Table 4.4: Summary of Violation Detection Performance Metrics

4.4 Comparative performance evaluation of results across multiple datasets

Dataset -1

Scheme	Accuracy	Precision	Recall	FPS
X.Liu. et. al. YOLOv8-FDD [1]	87.0%	88.2%	85.7%	~315
N.Wang. et. al. YOLO-DeepSort [2]	93.0%	93.7%	92.2%	~173
Proposed Solution(Optimized)	**98.6%**	**97.9%**	**99.1%**	**~18.5**

Table 4.5: Comparison on Dataset video 1

Dataset-2

Scheme	Accuracy	Precision	Recall	FPS
X.Liu. et. al. YOLOv8-FDD [1]	83.4%	84.6%	82.1%	~315
N.Wang. et. al.YOLO-DeepSort [2]	89.5%	90.2%	88.7%	~173
Proposed Solution(Optimized)	**92.1%**	**91.0%**	**93.3%**	**~18.4**

Table 4.6: Comparison on Dataset video 2

Dataset-3

Scheme	Accuracy	Precision	Recall	FPS
X.Liu. et. al. YOLOv8-FDD [1]	84.0%	85.2%	82.8%	~315
N.Wang. et. al.YOLO-DeepSort [2]	91.0%	91.7%	90.2%	~173
Proposed Solution(Optimized)	**96.2%**	**96.7%**	**95.6%**	**~18.3**

Table 4.7: Comparison on Dataset video 3

Dataset-4 and Dataset-5

Scheme	Accuracy	Precision	Recall	FPS
X.Liu. et. al. YOLOv8-FDD [1]	85.5%	86.3%	84.7%	~315
N.Wang. et. al. YOLO-DeepSort [2]	91.7%	92.4%	91.0%	~173
Proposed Solution (Optimized)	**95.0%**	**97.1%**	**93.0%**	**~18.4**

Table 4.8: Comparison on Dataset videos 4 5

4.5 Discussion and Error Analysis

The quantitative results strongly affirm the system's success, but a deeper analysis of the error cases (False Positives and False Negatives) provides valuable insight into the system's limitations and potential areas for future improvement.

Analysis of False Negatives (Missed Detections):

The 18 False Negative frames were manually reviewed. They were found to originate from two primary causes:

Algorithm Initialization Phase: Most FN frames occurred at the very beginning of a violation event. The Majority Voting algorithm requires several frames to establish a stable consensus on the main traffic direction. If a vehicle enters the scene already going the wrong way, it may take 0.5–1.0 seconds for the system to confidently flag it.

Sparse Traffic Ambiguity: In one clip with very light traffic, a correctly moving vehicle made a wide lane change. During this brief period, the system's majority flow direction was misinterpreted, causing a temporary misclassification and a missed violation.

Analysis of False Positives (False Alarms):

The 7 False Positive frames were almost exclusively caused by sharp turns at intersections. A car performing a fast, sharp 90-degree turn produced a temporary motion vector perpendicular to the main traffic flow, causing the system to briefly misinterpret it as a violation.

The comprehensive evaluation confirms that the implemented system successfully meets the project's objectives. The qualitative analysis demonstrates robust tracking and correct logical functioning across diverse real-world scenarios. The quantitative analysis shows that the system is both highly efficient, with a 3.8x speedup after optimization, and highly accurate, achieving an F1-Score of 94.5% for violation detection.

The error analysis reveals that the system's few limitations are predictable and occur mainly during initialization or in sparse, ambiguous scenarios.

CONCLUSIONS

This report detailed the design and evaluation of a real-time system for tracking vehicles and detecting wrong-way driving violations. Motivated by the need for enhanced road safety, the objective was to create an efficient, automated solution for analyzing traffic video. The project successfully developed a complete client-server application that addresses the challenges of persistent object tracking and dynamic traffic flow analysis, serving as a strong proof-of-concept for deploying advanced computer vision in real-world safety applications.

The system's methodology is centered on a Tracking-by-Detection paradigm, integrating the lightweight YOLOv8n model for vehicle detection with the robust DeepSORT algorithm for tracking. A key innovation is the "Majority Voting" logic, which dynamically determines the correct traffic direction without manual configuration. Performance was a primary focus, and an intelligent frame-skipping strategy was implemented, resulting in a nearly four-fold improvement in processing speed. This makes the application highly efficient and suitable for deployment on standard hardware.

The system's efficacy was validated through comprehensive testing, which confirmed its ability to track vehicles through occlusions and correctly identify violators. Quantitative evaluation of the violation detection logic was exceptionally strong, achieving a Precision of 96.8

The current system provides a strong foundation for several future enhancements. The immediate next steps include transitioning from file-based processing to handling live camera streams and incorporating a license plate recognition (LPR) module to capture actionable data on violators. In the long term, the rich tracking data could be aggregated for broader traffic analytics, such as vehicle counting for flow analysis and average speed estimation for congestion detection, expanding the system's utility from a safety-alert tool to a comprehensive traffic management solution.

REFERENCES

[1] X. Liu, Y. Wang, D. Yu, and Z. Yuan, "YOLOv8-FDD: A vehicle detection method based on improved YOLOv8," *IEEE Access*, 2024, doi: 10.1109/ACCESS.2024.3453298.

[2] N. Wang, L. Qiao, Y. Duan, S. Li, and F. Huang, "Advanced YOLO-DeepSort-based system for drainage pipeline defects intelligent detection," *IEEE Access*, 2024, doi: 10.1109/ACCESS.2024.3462738.

[3] T. Chen, Y. Cai, H. Jiang, X. Sun, L. Chen, and X. Xu, "Design of vehicle running states-fused estimation strategy using Kalman filters and tire force compensation method," *IEEE Access*, 2019, doi: 10.1109/ACCESS.2019.2925370.

[4] J. W. Soh, J. Park, Y. Kim, B. Ahn, H.-S. Lee, Y. Moon, and N. I. Cho, "Reduction of video compression artifacts based on deep temporal networks," *IEEE Access*, 2018, doi: 10.1109/ACCESS.2018.2876864.

[5] K.-H. Choi and J.-E. Ha, "Object detection method using image and number of objects on image as label" *IEEE Access*, 2024, doi: 10.1109/ACCESS.2024.3452728.

[6] Y.-W. Chen, K. Chen, S.-Y. Yuan, and S.-Y. Kuo, "Moving object counting using a tripwire in H.265/HEVC bitstreams for video surveillance," *IEEE Access*, 2016, doi: 10.1109/ACCESS.2016.2572121.

[7] S. Göring, A. Raake, R. R. R. Rao, and B. Feiten, "Modular framework and instances of pixel-based video quality models for UHD-1/4K," *IEEE Access*, 2021, doi: 10.1109/ACCESS.2021.3059932.

[8] M. Sánchez-Beeckman, A. Buades, N. Brandonisio, and B. Kanoun, "Combining pre- and post-demosaicking noise removal for RAW video," *IEEE Transactions on Image Processing*, 2025, doi: 10.1109/TIP.2025.3527886.