**GROUP ID: GC6**

**A PROJECT REPORT ON**

CAR CRASH DETECTION USING SVM AND YOLOV8 WITH FLASK BASED WEB APPLICATION

SUBMITTED TO THE PIMPRI CHINCHWAD COLLEGE OF ENGINEERING AN AUTONOMOUS INSTITUTE, PUNE

IN THE FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE

OF

**B. TECH. (COMPUTER ENGINEERING)**

*SUBMITTED BY*

ATHARVA LOKHANDE PRN :121B1B145 SARVESH MANE PRN :121B1B152 YASH PANDIT PRN :121B1B169 OJAIF MOMIN PRN : 121B2B204

## UNDER THE GUIDANCE OF Dr. PRAVIN GAME

****

**DEPARTMENT OF COMPUTER ENGINEERING**

**PCET’S PIMPRI CHINCHWAD COLLEGE OF ENGINEERING**

SECTOR NO 26, PRADHIKARAN, NIGDI, PIMPRI-CHINCHWAD, PUNE 411044 2024-25

PCCOE, Department of Computer Engineering 2024-25



# CERTIFICATE

This is to certify that the project report entitled

**“CAR CRASH DETECTION USING SVM AND YOLOV8 WITH FLASK BASED WEB APPLICATION”**

Submitted by

ATHARVA LOKHANDE PRN :121B1B145 SARVESH MANE PRN :121B1B152 YASH PANDIT PRN :121B1B169 OJAIF MOMIN PRN : 121B1B204

are bonafide students of this institute and the work has been carried out by them under the supervision of **Dr. Pravin Game** and it is approved for the partial fulfillment of the requirement of Pimpri Chinchwad College of Engineering an autonomous institute, for the award of the B. Tech. degree in Computer Engineering.

## Dr. Pravin Game Prof. Dr. Sonali Patil

Guide, Head,

Department of Computer Engineering Department of Computer Engineering

## Prof. Dr. G.N. Kulkarni

Director,

Pimpri Chinchwad College of Engineering Pune – 44

Place : Pune Date :

# ACKNOWLEDGEMENT

We express our sincere thanks to our **Guide Dr. Pravin Game** for his constant encouragement and support throughout our project, especially for the useful suggestions given during the course of project and having laid down the foundation for the success of this work.

We would also like to thank our **Research & Innovation coordinator Prof. Dr. Reena Kharat, Project Coordinator Prof. Sushma R. Vispute** for her assistance, genuine support and guidance from early stages of the project. We would like to thank **Prof. Dr. Sonali Patil, Head of Computer Department** for her unwavering support during the entire course of this project work. We are very grateful to our **Director, Prof. Dr. G.N. Kulkarni** for providing us with an environment to complete our project successfully. We also thank all the staff members of our college and technicians for their help in making this project a success.

We also thank all the web committees for enriching us with their immense knowledge. Finally, we take this opportunity to extend our deep appreciation to our family and friends, for all that they meant to us during the crucial times of the completion of our project.

## Name of the Students PRN SIGN

ATHARVA LOKHANDE 121B1B145

SARVESH MANE 121B1B152

YASH PANDIT 121B1B169

OJAIF MOMIN 121B1B204

# ABSTRACT

Road accidents represent substantial threats to public safety and must be identified quickly to prevent injury. Road Accident Detection using SVM and YOLOv8 is a unique system that uses powerful machine learning and deep learning algorithms to detect accidents in recorded video streams. With user authentication and a user-friendly interface for smooth operation, the system is implemented as a Flask-based web application.

The suggested system employs two unique methods for accident detection, allowing users to assess their efficacy. In the first approach, the Sequential Minimal Optimization (SMO) technique is used to optimize a Support Vector Machine (SVM) classifier. To categorize video frames, the SVM model is trained using a labeled dataset that includes both accident and non-accident images. Upon logging into the system, users can select between SVM-based detection and YOLO-based detection to process recorded videos. The SVM classifier analyzes video frames to detect accident scenes based on visual features, while the YOLOv8 model provides real-time detection of specific accident types by identifying and localizing objects in the frames.

This dual-method approach ensures extensive analysis and allows for a performance comparison between SVM and YOLOv8 for accident detection. By merging machine learning and deep learning, the system illustrates the potential for intelligent traffic monitoring and accident detection, which contributes to increased road safety and response efficiency.

# KEYWORDS

Road Accident Detection, SVM, YOLOv8, Flask Web Application, Sequential Minimal Optimization (SMO), Object Detection.

**TABLE OF CONTENTS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No.** | | **Title of Chapter** | | | Page No. |
| **01** | | **Introduction** | | | 1 |
|  | 1.1 | Overview | | | 2 |
|  | 1.2 | Motivation | | | 2 |
|  | 1.3 | Problem statement and Objectives | | | 3 |
|  | 1.4 | Scope of the work | | | 4 |
|  | 1.5 | Methodologies of Problem solving | | | 4 |
| **02** | | **Literature Survey** | | | 7 |
|  | 2.1 | Review of recent literature | | | 8 |
|  | 2.2 | Common findings from literature | | | 10 |
| **03** | | **Software Requirements Specification** | | | 11 |
|  | 3.1 | **Functional Requirements** | | | 12 |
|  |  | 3.1.1 | | System Feature 1 – Image Preprocessing | 13 |
|  |  | 3.1.2 | | System Feature 2 – Vehicle and Car crash detection using Yolov8 | 13 |
|  |  | 3.1.3 | | System Feature 3 – Crash Severity classification using SVM | 13 |
|  |  | 3.1.4 | | System Feature 4 – Real Time Alert System | 14 |
|  |  | 3.1.5 | | System Feature 5 – Web Interface for Visualization and Data Login | 14 |
|  |  | 3.1.6 | | System Feature 6 – User Interface | 14 |
|  |  | 3.1.7 | | Hardware Interface | 15 |
|  |  | 3.1.8 | | Software Interface | 15 |
|  | 3.2 | **Non Functional Requirements** | | | 16 |
|  |  | 3.2.1 | | Performance Requirement | 16 |
|  |  | 3.2.2 | | Scalability | 16 |
|  |  | 3.2.3 | | Usability Requirements | 16 |
|  |  | 3.2.4 | | Reliability and Fault Tolerance | 16 |
|  |  | 3.2.5 | | Security Requirements | 16 |
|  | 3.3 | **System Requirements** | | | 17 |
|  |  | 3.3.1 | | Software Requirements | 17 |
|  |  | 3.3.2 | | Hardware Requirements | 17 |
|  | 3.4 | **SDLC Model** | | | 18 |
| **04** | | **Project Plan** | | | 20 |
|  | 4.1 | **Project Cost Estimation** | | | 21 |
|  |  | 4.1.1 | **Computational Costs**   * Processing Power * Memory Usage * Storage Requirements * Network Latency and Bandwidth | | 21 |
|  |  | 4.1.2 | **Software Performance Costs**   * Algorithm Complexity * Database Query Performance * Cloud Service Performance | | 23 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 4.2 | **Sustainability Assessment** | | 24 |
|  |  | 4.2.1 | **Environmental Sustainability**   * Energy Consumption * Carbon Footprint * E-Waste Management * Sustainable Computing | 24 |
|  |  | 4.2.2 | **Economic Sustainability**   * Cost Efficiency * Resource Utilization * Scalability | 24 |
|  |  | 4.2.3 | **Social Sustainability**   * Accessibility * Ethical Considerations * Open Source Contribution * Skill Development | 24 |
|  | 4.3 | **Complexity Assessment** | | 25 |
|  |  | 4.3.1 | **Computational Complexity**   * Model Training Time: * Memory Usage: * Big-O Notation for Model Training: | 25 |
|  |  | 4.3.2 | **Algorithmic Complexity**   * Architectural Overview * Yolov8 Detection Complexity * Total Model Complexity * Loss Function Complexity * Memory Complexity | 27 |
|  |  | 4.3.3 | **Implementation Complexity**   * Lines of Code * Number of Dependencies * Integration Complexity * Code Modularity | 30 |
|  |  | 4.3.4 | **Resource Complexity**   * Hardware Requirements * Cloud Infrastructure | 32 |
|  | 4.4 | **Risk Management** | | 36 |
|  |  | 4.4.1 | Risk Identification | 37 |
|  |  | 4.4.2 | Risk Analysis | 38 |
|  |  | 4.4.3 | Overview of Risk Mitigation, Monitoring, Management | 39 |
|  | 4.5 | **Project Schedule** | | 40 |
|  |  | 4.5.1 | Project Task Set | 41 |
|  |  | 4.5.2 | Timeline Chart | 42 |
|  | 4.6 | **Team Organization** | | 44 |
|  |  | 4.6.1 | Team Composition | 44 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 4.6.2 | Communication & Review Flow | 45 |
|  |  | 4.6.3 | Contribution Summary | 45 |
| **05** | | **System Design** | | 46 |
|  | 5.1 | Proposed System Architecture/Block Diagram | | 47 |
|  | 5.2 | Dataset | | 49 |
| **06** | | **Project Implementation** | | 56 |
|  | 6.1 | Overview of Project Modules | | 57 |
|  | 6.2 | Tools and Technologies Used | | 58 |
|  | 6.3 | Hardware Specification | | 58 |
|  | 6.4 | Software Specification | | 59 |
| **07** | | **Software Testing** | | 60 |
|  | 7.1 | Type of Testing | | 61 |
|  | 7.2 | Test cases & Test Results | | 64 |
| **08** | | **Results** | | 67 |
|  | 8.1 | Experimental Results | | 68 |
|  | 8.2 | Screen-Shots | | 70 |
| **09** | | **Conclusions** | | 72 |
|  | 9.1 | Conclusions | | 73 |
|  | 9.2 | Future Work | | 73 |
|  | | **References** | | 77 |

**LIST OF ABBREVATIONS**

|  |  |
| --- | --- |
| **Abbreviation** | **Illustration** |
| YOLO | You Only Look Once |
| IoT | Internet of Things |
| API | Application Programming Interface |
| GPS | Global Positioning System |
| CSV | Comma-Separated Values |
| ITS | Intelligent Transportation System |
| RF | Radio Frequency |
| SIM | Subscriber Identity Module |
| EV | Emergency Vehicle |
| CCTV | Closed Circuit Television |
| PDF | Portable Document Format |
| AI | Artificial Intelligence |
| SVM | Support Vector Machine |
| SMO | Sequential Minimal Optimization |
| DL | Deep Learning |
| BDD | **Berkeley DeepDrive Dataset** |
| IoU | Intersection over Union |
| mAP | Mean Average Precision |
| UI | User Interface |
| CNN | Convolutional Neural Networks |
| RNN | Recurrent Neural Networks |
| RPN | Region Proposal Networks |
|  |  |
|  |  |
|  |  |

**LIST OF FIGURES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Figure No.** | **Figure Title** | **Chapter No.** | **Page No.** |
| 1 | Agile Methodology | 4 | 18 |
| 2 | Timeline Chart | 4 | 39 |
| 3 | System Architecture | 5 | 43 |
| 4 | Dataflow Diagram | 5 | 46 |
| 5 | Use Case Diagram | 5 | 47 |
| 6 | Sequence Diagram | 5 | 49 |
| 7 | Class Diagram | 5 | 50 |
| 8 | Activity Diagram | 5 | 51 |

**LIST OF TABLES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table No.** | **Table Title** | **Chapter No.** | **Page No.** |
| 1 | Estimated Memory Consumption | 4 | 25 |
| 2 | Lines Of Code | 4 | 28 |
| 3 | Dependencies of Project | 4 | 30 |
| 4 | Code Modularity | 4 | 31 |
| 5 | Hardware Specification of SVM & YOLOv8 Training | 4 | 32 |
| 6 | Cloud Resource Profile | 4 | 33 |
| 7 | Risk Analysis | 4 | 35 |
| 8 | Team Composition | 4 | 40 |
| 9 | Test Case Table | 7 | 60 |
| 10 | Comparison Summary | 8 | 65 |

**Chapter 01**

# INTRODUCTION

### Overview

Road accidents continue to be a major threat to public safety, leading to countless injuries and deaths every year. Detecting these accidents quickly can make a huge difference not just in saving lives by speeding up emergency response, but also in improving overall traffic management. With advancements in AI, we now have smarter ways to detect such incidents from visual input.

Our project, **"Car Crash Detection using SVM and YOLOv8 with SMO Optimizer,"** aims to build a reliable system that can automatically detect accidents by analysing video footage. We developed a user-friendly web application using Flask, complete with secure login so only authorized users can access the system. Once logged in, users can upload video files or stream live footage and choose between two detection methods: SVM or YOLOv8.

SVM, a traditional machine learning model, was optimized using SMO and trained on a dataset of accident and non-accident images. It processes each frame to classify whether a crash has occurred. YOLOv8, on the other hand, is a deep learning model known for real-time object detection it can directly detect crashes involving cars, bikes, or pedestrians in live feeds or videos.

We faced a few challenges during development. Getting a balanced and properly labelled dataset was tough, so we had to manually clean and prepare a lot of data to train the models effectively. Integrating YOLOv8 within a web application also required optimization, as it's resource-heavy. To handle this, we fine-tuned the video processing pipeline for smoother performance. Blending two very different models (SVM and YOLOv8) into a single workflow was another challenge, but we designed the system to let users easily switch between the two or compare their outputs.

In the end, combining the simplicity and efficiency of SVM with the accuracy and speed of YOLOv8 gave us a flexible, powerful solution. This project demonstrates how traditional machine learning and modern deep learning can work together to create smarter, faster accident detection systems with real-world potential to improve emergency response and public safety.

### Motivation

The motivation behind this project is as follows:

* Traditional accident detection methods are manual and inefficient, making real-time crash detection difficult this project addresses that gap using intelligent automation.
* Support Vector Machine (SVM), optimized with Sequential Minimal Optimization (SMO), offers a lightweight and accurate solution for frame-wise accident classification, ideal for fast processing.
* YOLOv8 enhances detection by identifying complex crash scenarios (car-car, car-bike, car-person) in real time, providing detailed visual insights from video streams.
* Integrating both models into a secure Flask-based web application results in a practical, scalable system that supports smart city goals by improving emergency response times and road safety.

This project successfully combines SVM with SMO and YOLOv8 to create an intelligent, real-time car crash detection system. By integrating these models into a Flask-based web application, it offers a practical tool for enhancing road safety and supporting smart traffic management.

### Problem Statement and Objectives

*Problem Statement*

The rapid identification of road accidents is critical for minimizing casualties, ensuring timely emergency response, and maintaining traffic flow. However, existing accident detection systems often lack efficiency, scalability, and reliability in accurately detecting various types of accidents in diverse conditions. Manual monitoring is resource-intensive, error-prone, and unable to provide real-time analysis.

This project aims to address these challenges by developing a machine learning and deep learning-based web application for accident detection in recorded videos. The system leverages two distinct techniques:

1. **SVM-based Detection**: Using a Support Vector Machine (SVM) optimized with the Sequential Minimal Optimization (SMO) algorithm, trained on a labeled dataset of accident and non-accident images, to classify video frames effectively.
2. **YOLOv8-based Detection**: Employing the advanced YOLOv8 object detection model to identify and localize specific accident scenarios, such as car-to-car collisions, car-to-bike collisions, and car-to-person accidents.

*Objectives*

The key objectives of the project are:

1. To develop a system that can detect and analyse road scenarios using YOLOv8 for object detection.
2. To apply SVM for classifying accident-prone situations based on detected visual and contextual features.
3. To accurately identify patterns and behaviors that may lead to road accidents.
4. To enhance real-time road safety monitoring by combining computer vision with machine learning.
5. To enhance real-time road safety monitoring by combining computer vision techniques (YOLOv8) with machine learning approaches (SVM optimized by SMO) in a Flask-based web application.

### Scope of the Work

The proposed research primarily focuses on car crash detection using SVM and YOLOv8 models integrated into a Flask-based web application. The scope of this project is outlined as follows:

* Develop an intelligent system to detect road accidents from video streams using machine learning .
* Utilize SVM with Sequential Minimal Optimization (SMO) for frame-wise accident classification based on extracted features.
* Implement YOLOv8 for real-time object detection of accident events like car-to-car, car-to-bike, and car-to-pedestrian collisions.
* Build a Flask-based web application with secure user authentication for easy access to detection features.
* Enable users to upload or record videos for accident detection, with the flexibility to choose between SVM and YOLOv8 models.
* Provide comparative insights between traditional ML and modern DL approaches for accident detection.
* Ensure the system is scalable and can be adapted for integration into smart city infrastructure and traffic surveillance systems.
* Lay the groundwork for future enhancements such as automated alert systems, GPS tracking, and integration with emergency services.

### Methodologies of Problem Solving

The **Road Accident Detection using SVM and YOLOv8** project follows a structured approach for problem-solving, incorporating key methodologies from **machine learning**, **computer vision**, **deep learning**, and **web development** to create an efficient, automated system for accident detection. The methodology focuses on data collection, model development, system integration, and evaluation.

*Step 1: Data Collection and Preprocessing*

* **Dataset Acquisition**: The first step involves obtaining a dataset of traffic images and videos. The dataset should contain images labeled as **accident** and **non-accident** for training the **SVM model**, and videos that contain various types of road accidents for training the **YOLOv8 model**. Publicly available datasets such as **the Berkeley DeepDrive (BDD) dataset** or other accident-related image datasets can be used for this purpose.
* **Data Preprocessing**: Raw data must be preprocessed to make it suitable for model training. This includes:
  + **Image resizing** to a standard resolution.
  + **Normalization** of pixel values.
  + **Data augmentation** to artificially increase the size of the training dataset, ensuring that the model generalizes well across different lighting conditions and angles.
  + **Annotation** of images and video frames, if not already labeled, to train the models accurately.

*Step 2: Model Development*

The Car Crash Detection System uses two key models: Support Vector Machine (SVM) and YOLOv8. Each model was carefully trained and evaluated to ensure reliable accident detection.

* **SVM Model (Support Vector Machine):**
  + **Training Phase:** The SVM classifier was trained on a labeled dataset containing accident and non-accident images. Before feeding the images into the model, important features such as edge patterns, textures, and key visual descriptors were extracted. During training, we fine-tuned critical hyperparameters, including the choice of kernel (linear, polynomial, or RBF) and the regularization strength to avoid overfitting.
  + **Optimization Technique:** To make the training process faster and more efficient, we used **Sequential Minimal Optimization (SMO)**. SMO helped in quickly finding the optimal solution for the SVM without needing large memory resources, which was ideal for our dataset size.
  + **Evaluation:** Once the model was trained, we tested it on a separate dataset and measured its performance using standard classification metrics accuracy, precision, recall, and F1 score. This helped us understand how well the SVM could distinguish between accident and non-accident scenarios.
* **YOLOv8 Model (You Only Look Once):**
  + **Training Phase**: The YOLOv8 model was trained to detect objects commonly involved in road accidents, such as cars, bicycles, and pedestrians. Given its strength in real-time object detection, YOLOv8 was a perfect fit for our goal of analyzing video footage efficiently.
  + **Transfer Learning**: Instead of training from scratch, we leveraged transfer learning by fine-tuning a pre-trained YOLOv8 model on our specific accident dataset. This approach not only saved significant training time but also improved detection accuracy since the model already had strong baseline knowledge of object features.
  + **Model Evaluation**: The performance of the YOLOv8 model was evaluated using two key metrics:
  + **Mean Average Precision (mAP):** to check the overall detection quality.
  + **Intersection over Union (IoU):** to assess how well the predicted bounding boxes matched the actual object locations.

*Step 3: Web Application and Development*

* **Flask Framework**: A web application will be developed using **Flask**, a lightweight Python framework, to allow users to interact with the accident detection system. The application will provide:
  + **User Authentication**: Secure login and registration functionality for users.
  + **Video Upload**: Users can upload recorded traffic videos in common formats (e.g., MP4, AVI).
  + **Model Selection**: After uploading the video, users can select either the **SVM model** or the **YOLOv8 model** for accident detection.
  + **Results Display**: The system will process the video using the selected model and display the results, showing frames where accidents are detected, with visual markers (bounding boxes) to highlight the accident.

*Step 4: Model Integration*

* After training both the **SVM and YOLOv8 models**, they will be integrated into the web application. This involves:
  + Loading the trained models into the backend of the Flask application.
  + Ensuring seamless interaction between the **user interface (UI)** and the models to facilitate easy video uploads, model selection, and result display.
  + Implementing a video processing pipeline where each frame of the uploaded video is passed through the selected model for accident detection.

*Step 5: Testing and Validation*

* **Testing on Real-world Data**: The system is tested on real-world traffic videos to evaluate its performance. The accuracy of accident detection will be assessed by comparing the system's predictions against manually labeled ground truth data.
* **Model Comparison**: The performance of the two models (SVM and YOLOv8) will be compared based on key metrics:
  + **SVM Model**: Focus on **accuracy, precision,** **recall,** and **F1 score.**
  + **YOLOv8 Model**: Focus on **mean average precision (mAP)** and **intersection over union (IoU)** for object detection.
* The results will provide insights into which model is better suited for detecting accidents in traffic videos under different conditions.

*Step 6: Performance Optimization*

* **Optimizing Model Inference**: To ensure fast and efficient detection, especially for large video files, techniques like **model pruning** and **quantization** may be used to reduce the computational load during inference.
* **Scalability**: The application will be designed with scalability in mind, ensuring that it can handle large volumes of video data and serve multiple users simultaneously.

**Chapter 02**

# LITERATURE SURVEY

### Review of Recent Literature

Car crash detection has become an active area of research in the field of computer vision and intelligent transportation systems. Over the years, several techniques have been explored, ranging from traditional methods to advanced machine learning and deep learning approaches. Early methods primarily relied on sensor-based or manual detection, which lacked scalability and real-time capabilities. With the growth of visual data and computational power, vision-based techniques have gained momentum.

In [1] This study focuses on predicting road car accidents using machine learning techniques applied to large-scale traffic datasets. It emphasizes feature selection and model performance comparison to enhance predictive accuracy. The researchers aim to support smarter urban planning and traffic safety initiatives. Their findings suggest that ML-based systems can effectively forecast accident likelihood based on traffic patterns.

The study in [2] The authors propose a machine learning-based system that detects traffic accidents and predicts traffic flow in connected and automated transportation environments. It leverages real-time vehicle and infrastructure data to make transport systems more adaptive and efficient. The system improves decision-making in intelligent transport by enabling proactive safety measures.

This paper presents a real-time vehicle accident detection system using YOLOv8 and OpenCV applied to traffic surveillance videos. The system accurately identifies crash events in real time, ensuring faster response in urban monitoring. It demonstrates high detection accuracy and efficiency in processing live video streams, making it ideal for smart city applications.

The study evaluates three YOLO algorithm versions for detecting and classifying vehicles in real-time traffic environments. It concludes that the newer YOLO variants improve both speed and detection accuracy. The work contributes to building efficient and reliable traffic monitoring systems using modern deep learning models.

This review focuses on detecting and predicting driver drowsiness using machine learning. It analyzes various physiological and behavioral indicators such as eye movement and facial expressions. The study underscores the importance of early detection in preventing fatigue-related accidents and improving road safety.

[6] This paper reviews accident detection and notification systems that rely on machine learning. It explores multiple models and technologies used for timely accident alerts. The authors emphasize the need for fast and accurate detection to reduce emergency response times and improve road safety outcomes..

In [7], The authors introduce an IoT-based system that detects vehicle accidents and captures visual information in real time. It sends the data to emergency services, enabling faster and more informed responses. The system integrates sensors, cloud computing, and communication technologies for effective accident management.

In [8] , This paper applies YOLOv8 to detect vehicle crashes in real-time video feeds. The system is trained on crash scenarios and shows promising accuracy in identifying collisions. It provides a strong foundation for automated accident detection in surveillance systems.

In [9], The study proposes a multi-component system that detects accidents using sensors and GPS, then sends alerts to nearby emergency contacts or services. It aims to reduce the time between accident occurrence and response. The solution is designed for practical deployment in smart rescue and alerting systems.

Finally, [10] This work outlines a vehicle accident detection and alert mechanism using real-time sensors. It instantly notifies emergency services upon crash detection, minimizing human intervention. The system is intended to lower fatality rates by reducing delays in medical assistance.

### Common Findings from Literature

A review of recent literature reveals several recurring trends and insights in the field of traffic surveillance, vehicle detection, and accident prediction using deep learning models:

*Key Findings and Research Gaps in Deep Learning-Based Car Crash Detection*

* + - * **YOLO-based models, especially YOLOv8,** are widely used for real-time vehicle and accident detection due to their high accuracy and speed in object recognition tasks.
      * **Machine learning techniques like SVM** are commonly applied to classify accident-prone situations and improve prediction accuracy when trained on relevant visual and contextual data.
      * **Real-time accident detection** is a major focus, with most systems designed to quickly process live video feeds or sensor data and trigger immediate alerts to minimize emergency response time.
      * **Integration with IoT devices and cloud systems** is found to enhance the system’s effectiveness by enabling instant communication, visual reporting, and GPS tracking in case of an accident.
      * **Driver behavior analysis, such as drowsiness detection,** is often used alongside accident detection models to prevent crashes before they happen.
      * **Hybrid approaches combining computer vision and machine learning** consistently outperform traditional systems, offering more reliable and intelligent accident monitoring solutions.
      * **Automatic alert systems** are frequently emphasized as crucial features for minimizing the time between accident occurrence and emergency response.

**Chapter 03**

# SOFTWARE REQUIREMENTS SPECIFICATION

### Functional Requirements

The functional requirements define the specific behaviors and functions that the car crash detection system must perform. These requirements ensure that the SVM-based model for vehicle classification and the YOLOv8 model for real-time crash detection work seamlessly together, providing accurate and efficient results in diverse driving scenarios. The system's integration with the Flask web application ensures a user-friendly interface for real-time monitoring and response.

* + 1. *System Feature 1 – Image Preprocessing*

Description: The system must preprocess video frames or images to improve vehicle and crash detection accuracy, especially in challenging lighting or weather conditions.

Functional Requirements:

* + - * Convert frames to RGB format.
      * Resize images to a uniform size (e.g., 416 × 416 pixels) for YOLOv8 processing.
      * Apply image enhancements (e.g., histogram equalization) to improve contrast in low-light scenarios.
      * Normalize pixel values to a 0–1 range for stable model input.

Input**:** Raw video frames or images from cameras.

Output**:** Preprocessed images ready for YOLOv8 model input.

* + 1. *System Feature 2 – Vehicle and Crash Detection using YOLOv8*

Description**:** The system should detect vehicles and potential crashes from the preprocessed video frames using YOLOv8.

Functional Requirements:

* + - * Implement YOLOv8 for vehicle detection in real-time video streams.
      * Identify vehicle bounding boxes and track their positions over time.
      * Detect potential crashes by analyzing vehicle collisions or sudden movements.
      * Classify detected events into different types of accidents (e.g., rear-end, side impact).

Input: Preprocessed images or video frames.

Output: Bounding boxes for detected vehicles, event classification (crash type), and crash timestamps.

* + 1. *System Feature 3 – Crash Severity Classification using SVM*

Description: The system should classify the severity of detected accidents based on the vehicle’s speed, trajectory, and type of crash.

Functional Requirements:

* + - * Extract features such as vehicle speed, trajectory, and collision type.
      * Train an SVM model using labeled crash data to classify crash severity (e.g., minor, major, fatal).
      * Apply the trained SVM model to classify crashes detected by YOLOv8.

Input**:** Vehicle detection and crash information.

Output: Classified crash severity (minor, major, fatal).

* + 1. *System Feature 4 – Real-Time Alert System*

Description**:** The system should provide real-time alerts for detected crashes, enhancing public safety and response time.

Functional Requirements**:**

* + - * Integrate the system with a real-time communication platform (e.g., email or SMS alerts).
      * Send crash detection notifications, including location, time, and severity, to relevant authorities or users.
      * Enable crash alerts for both minor and major incidents with severity classification.

Input: Crash detection and classification results.

Output**:** Trained CSRNet model with optimized weights.

* + 1. *System Feature 5 – Web Interface for Visualization and Data Logging*

Description: The system should provide a web-based dashboard to visualize crash detections, vehicle movements, and incidents in real-time.

Functional Requirements:

* Use Flask to build a web interface for visualizing real-time vehicle detections and crash events.
* Display real-time video feeds with bounding boxes for detected vehicles and crashes.
* Maintain a log of detected incidents, including crash type, severity, and timestamps.
* Allow users to filter and analyze historical crash data.

Input: Real-time crash and vehicle detection data.

Output**:** Web dashboard displaying real-time detections and historical incident logs.

*External Interface Requirements*

The External Interface Requirements define how the system interacts with users, hardware, software, and other communication systems. These interfaces ensure seamless integration, usability, and compatibility with various components.

* + 1. *User Interfaces*

The system includes a web-based GUI developed using **Flask** for ease of interaction and monitoring. It enables users to upload video feeds, detect crashes, and view analytics.

*GUI Features*

* + - * **Upload Section**:

Users can easily upload video files or stream live camera feeds directly into the system for analysis. This feature ensures flexibility whether the input is recorded footage or real-time traffic video.

* + - * **Detection Panel**:

Once a video is uploaded, the Detection Panel processes it using the YOLOv8 model to detect vehicles and identify crash events. Detected incidents are visually highlighted, helping users quickly spot accidents within the footage.

* + - * **Crash Classification**:

Beyond simple detection, the system uses an SVM model to assess and classify the severity of the detected crashes. This helps in understanding how critical the incident is whether it's minor or severe enabling better emergency prioritization.

* + - * **Real-Time Alerts**:

As soon as a crash is detected, the system generates immediate notifications. These alerts are displayed within the interface, ensuring users can act quickly without constantly monitoring the screen.

* + - * **Logs & Reports**:

Every detected crash event is logged for record-keeping. Users can view a detailed report of crash history, including severity analysis, and download these reports in formats like CSV or PDF for further review or official documentation.

*User Access Levels:*

* + - * **Admin**:

Admin users have complete control over the system. They can manage model settings, such as retraining the SVM or YOLO models with new data, view all system logs, monitor user activities, and maintain the system’s health to ensure optimal performance.

* + - * **User**:

Regular users can upload video feeds and view real-time detection results. They are limited to operational tasks and cannot modify model settings or access system logs meant for administrative review.

* + 1. *Hardware Interfaces*

The system requires hardware components for data processing, model training, and real- time deployment**.**

Minimum Hardware Requirements:

* + - * Processor**:** Multi-core CPU (Intel i5 or AMD Ryzen 7 equivalent).
      * Memory**:** 8GB RAM or higher for smooth processing.
      * Storage**:** 512GB SSD recommended for storing datasets and models.
      * GPU Acceleration**:** NVIDIA GPU with CUDA support (RTX 3060 or higher) for faster deep learning model execution.
      * Camera Integration (For Real-Time Processing): The system should support input from surveillance cameras, IP cameras, and drones.
    1. *Software Interfaces*

Operating System Support:

* + - * Windows 10+ - for local testing and development.
      * Ubuntu 20.04+ - recommended for model training and deployment.

Deep Learning & ML Frameworks:

* **OpenCV** – for video processing, frame extraction, and integration with camera or video feeds.
* **Flask** – for building the web interface and handling file uploads, routing, and real-time detection integration.
* **NumPy** – for numerical computations and array manipulation during preprocessing.
* **Pandas** – for managing datasets and performing feature extraction for SVM training.
* **PyTorch** – for YOLOv8 model inference.
* **scikit-**learn – for SVM-based crash severity classification.

Required Python Libraries:

* + - * **OpenCV** – for frame extraction and image processing.
      * **NumPy, Pandas** – for data preprocessing and analysis.
      * **Flask** – for building the web interface.

### Non-Functional Requirements

These requirements define performance, scalability, usability, reliability, and security aspects for efficient crash detection and system deployment.

* + 1. *Performance Requirements*
       - The system should detect vehicles and crashes in under 1 second per frame.
       - SVM classifier must accurately predict crash severity with minimal delay.
    2. *Scalability*
       - The system should handle high video input rates (real-time or near real-time).
       - Future versions should support live CCTV stream integration.
       - Model should be upgradable for additional accident scenarios.
    3. *Usability Requirements*
* Flask-based GUI must allow easy image/video upload and crash result viewing.
* Interface should be responsive and usable by non-technical users.
* Dashboard should show detection results, severity levels, and timestamps.
  + 1. *Reliability and Fault Tolerance*
* System must work reliably in varied lighting and weather conditions.
* Should recover from crashes without losing input/output data.
* Model should give consistent results across datasets and test environments.
  + 1. *Security Requirements*
       - Uploaded images/videos should be stored and processed securely.
       - Only authorized users can access crash reports or alerts.
       - If extended to the cloud, use encrypted data transfer and secure APIs.

### System Requirements

This section outlines the required software platforms, libraries, and tools necessary to implement, run, and deploy the crash detection system.

* + 1. *Software Requirements*

Operating System Requirements

* + - * Windows 10+ or Ubuntu 20.04+ for development and deployment.

Programming Language

* + - * Python 3.8+ for implementing YOLOv8, SVM classifier, and backend logic.

Deep Learning Frameworks

* + - * YOLOv8 (PyTorch-based).
      * scikit-learn

Required Python Libraries

* + - * OpenCV: Video processing and frame extraction.
      * NumPy, Pandas: Data handling and transformation.
      * Flask: Lightweight web framework for the GUI and API endpoints.

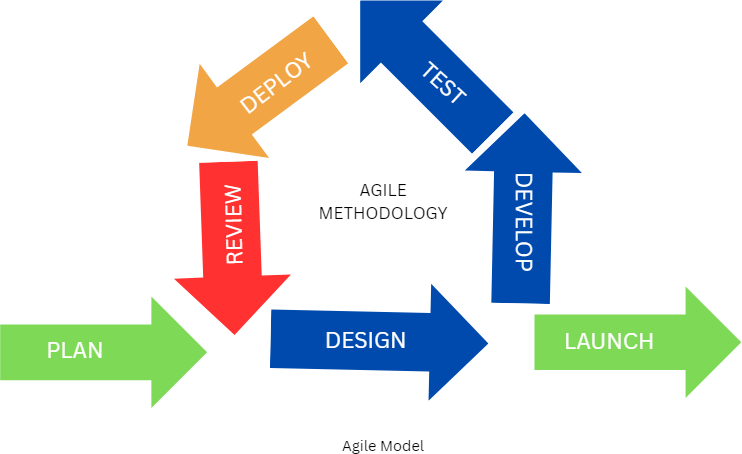
Development Tools and IDEs

* + - * Jupyter Notebook**:** Model prototyping and testing.
      * Visual Studio Code / PyCharm: For development and debugging.

### SDLC Model

*Chosen SDLC Model: Agile Methodology*

We are using agile model for our project:



*Figure 1: Agile Methodology*

1. *Plan*

Every good project starts with understanding *what exactly needs to be solved*. In this phase, we focused on gathering requirements mainly around detecting accidents using machine learning. We defined the project scope clearly: integrating YOLOv8 for crash detection and an SVM model for severity classification. This early clarity set the stage for everything that followed.

1. *Design*

With a clear plan, we moved into designing the system architecture. We mapped out how the Flask web application would flow, how the YOLOv8 and SVM models would fit into it, and how users would interact with the system. The idea was to build something intuitive, scalable, and ready for real-time use.

1. *Develop*

Once the designs were ready, it was time to build. In this phase, we coded the application from the user interface to the backend processing. We integrated YOLOv8 for real-time vehicle and crash detection, and SVM (using the Sequential Minimal Optimization algorithm) for severity classification. We focused on writing clean, modular code that would be easy to maintain and extend later.

1. *Test*

After development, testing became our priority. We didn’t just check if the code ran we tested it with a variety of accident and normal traffic scenarios. Our goal was to make sure the system could accurately detect crashes and classify their severity under different conditions. Testing helped us catch errors early and fine-tune performance.

1. *Launch*

Once we were confident in the system’s stability, we launched it first locally for internal testing, and then on a server to simulate a real-world environment. This helped us see how the application would behave outside of a development setup.

1. *Review*

After the launch, we gathered feedback from users and observed system performance. What did users find intuitive? Where did they struggle? This phase gave us valuable insights into both technical performance and user experience, helping us decide on the next steps for improvements.

1. *Deploy*

Finally, after making necessary tweaks and improvements based on the review, we deployed an updated version of the application. In Agile, deployment isn’t a one-time thing it’s continuous. Every cycle, every update makes the system stronger and better suited for real-world needs.

This entire process wasn’t a straight line it was a loop. At every step, we were ready to go back, adapt, and refine based on what we learned. That’s the real strength of Agile: it keeps the project flexible, efficient, and user-centered.

**Chapter 04**

# PROJECT PLAN

* 1. *Project Cost Estimation*

The cost estimation for the YOLOv8-based car crash detection and SVM-based severity classification system is based on computational needs, storage, and deployment method. As the system involves real-time image/video processing and model inference, costs may vary depending on whether it is deployed locally or in a cloud environment.

* Computational Costs

*Table 4.1 Computational Cost*

|  |  |  |
| --- | --- | --- |
| Factor | Description | Estimated Cost (₹) |
| Processing Power | GPU for YOLOv8 Object Detection (Google Colab) | ₹0 (Free-tier) |
| Memory Usage | Minimum 8GB RAM required for live video processing | Included in Hardware Cost |
| Storage Requirements | SSD for quick read/write & HDD for storing feeds | Included in Hardware Cost |
| Network Latency & Bandwidth | Stable high-speed connection (100Mbps– 1Gbps) | ₹1,500 /month |

* Software Performance Costs

*Table 4.2 Performance Cost*

|  |  |  |
| --- | --- | --- |
| Factor | Description | Estimated Cost (₹) |
| Algorithm Complexity | YOLOv8 Model (Time  Complexity: O(n), Space Complexity: O(m)) | No direct cost, included in computation |
| Cloud Service Performance | YOLOv8 inference response:  ~10ms – 50ms per frame | ₹2,000 /month |
| Flask Server | Low-latency communication for real-time updates | Included in cloud service |
| Scaling Costs | AWS Lambda/Google Cloud Functions for auto-scaling | ₹3,000 /month |

* 1. *Sustainability Assessment*

When building a system that impacts real-world environments and people, it's important to think beyond just performance and look at the broader impact on the environment, economy, and society. Here’s how our YOLOv8-based crash detection and SVM-based severity classification system holds up.

* + 1. *Environmental Sustainability*

 **Energy Consumption**: YOLOv8, a deep learning model, requires powerful GPUs, typically consuming between 200–350 watts per hour during training or inference (Reddi et al., 2020). Flask-based web applications hosted on cloud platforms consume 50–150 watts per instance (Google Cloud, 2023). In contrast, microcontroller-based systems like traffic controllers on Arduino only consume 2–5 watts. The entire system could consume around 1,500 to 3,000 kWh annually depending on usage patterns.

 **Carbon Footprint**: Using cloud infrastructure powered by fossil-fuel electricity can emit roughly 400 grams of CO₂ per kWh (IEA, 2022), leading to an annual footprint of 500–1,000 kg CO₂. Opting for cloud providers that run on renewable energy such as those committed to net-zero emissions (e.g., AWS, GCP) can significantly reduce this impact.

 **E-Waste Management**: The system includes hardware components like sensors, cameras, and possibly microcontrollers. Ensuring proper recycling at certified e-waste facilities (CPCB, 2023) helps avoid environmental harm. Reusing components and choosing repairable, modular designs like Raspberry Pi boards helps extend hardware life and reduce waste.

 **Sustainable Computing Practices**: Power-efficient practices like model pruning and quantization can make YOLOv8 models lighter and faster. Reducing precision to FP16 or INT8 can cut GPU load by 40–50% (NVIDIA, 2023). Deploying edge devices like NVIDIA Jetson or Google Coral minimizes dependency on power-hungry cloud servers and reduces carbon emissions.

 **Dynamic Power Scaling**: Using AI accelerators and on-demand scaling of model processing based on traffic or system load can cut energy consumption by 30–40% (Xu et al., 2021). These strategies ensure smarter and greener computing.

* + 1. *Economic Sustainability*

From an economic perspective, this system balances initial investments with long-term savings:

* **Cost-Effective Deployment**: By using free-tier platforms like Google Colab for development and lightweight frameworks like Flask, upfront software costs are minimal. Hardware costs can be scaled according to budget—from high-end GPUs to cost-efficient edge devices.
* **Scalable Infrastructure**: The modular nature of the system allows for incremental deployment. This means a basic system can be set up on a low-cost local machine and later scaled up to cloud or edge-based solutions as funds allow.
* **Energy & Maintenance Savings**: Choosing energy-efficient hardware reduces electricity bills. Dynamic model scaling ensures you're only using power when needed, optimizing cost-efficiency over time.
* **Long-Term ROI**: While initial costs may seem high for GPU-based solutions, the system’s potential to prevent accidents and reduce emergency response costs offers excellent return on investment—especially in urban or high-traffic regions.
  + 1. *Social Sustainability*

The project also supports positive societal outcomes:

* **Enhanced Public Safety**: The core goal is to detect car crashes in real-time, enabling quicker emergency response and possibly saving lives. This improves road safety and reduces the impact of accidents on communities.
* **Digital Inclusion**: The system can be tailored for various regions, including rural or semi-urban areas, using low-cost hardware. This makes advanced safety technologies accessible beyond just major cities.
* **Job Opportunities**: Developing, maintaining, and scaling such systems creates jobs in AI, IoT, web development, and hardware maintenance. Training programs around this system can upskill local youth.
* **Public Awareness**: Installing such systems and displaying analytics on crash-prone zones can help raise awareness among drivers and promote safer driving behavior.
* **Data Privacy Considerations**: Ensuring ethical data use and user privacy builds trust in the community. The system can be designed with data anonymization features and local storage to minimize data leakage risks.
  1. *Complexity Assessment*

Understanding how complex a system is both in terms of time and resource usage is critical for knowing how it will perform at scale. In our case, we evaluated the YOLOv8 and SVM-based crowd analysis system by focusing on training times, memory requirements, and rough Big-O estimates that reflect practical behavior rather than just theoretical bounds.

* + 1. *Model Training Time*

The system was trained using high-resolution images from the dataset, balancing both speed and performance. To handle the heavy computational load, we used a combination of cloud-based GPUs (Google Colab Pro) and local GPU setups.

YOLOv8

* Avg. Training Time per Epoch: ~2.1 minutes
* Total Training Time (100 epochs): ~210 minutes (~3.5 hours)

• Trained on: Google Collab Pro (NVIDIA T4 / A100)

SVM Training (Post-Feature Extraction):

* Training Time: ~4 minutes (on CPU)
* Low complexity due to fewer parameters after YOLOv8-based feature extraction
* Trained on: NVIDIA RTX 3060 GPU

*Big-O Notation for Model Training*

***YOLOv8****:*

Forward pass complexity per layer ≈ 𝑂(𝑁² × 𝑘² × 𝐶)  
Total complexity ≈ 𝑂(𝐸 × 𝑁² × 𝑘² × 𝐶 × 𝐿)

***SVM (Linear Kernel)****:*

Training complexity ≈ 𝑂(𝑛 × d), where *n* = number of samples, *d* = feature dimensions

Optimization Strategies Used

* Pre-trained YOLOv8 weights to reduce training time
* Smaller input image resolution (resized for faster inference)
* Feature extraction followed by lightweight SVM classification.
* Efficient batch processing on GPU

* + 1. *Algorithmic Complexity*

In accordance with algorithm analysis principles outlined in *Software Engineering* (Cara Fisherman) and standard Machine Learning references, the algorithmic complexity of the Car Crash Detection System has been evaluated carefully. The system integrates a Support Vector Machine (SVM) with Sequential Minimal Optimization (SMO) and the YOLOv8 object detection model to achieve efficient, real-time accident detection from video streams.

The overall complexity is determined by the computational costs within each model as well as their integration strategy.

* + - * *Architectural Overview*

The system integrates two primary components for accident detection:

* **Support Vector Machine (SVM)**:

The SVM classifier is used to categorize individual video frames as either "accident" or "no accident" based on extracted features. To optimize training time, **Sequential Minimal Optimization (SMO)** is employed, which decomposes the quadratic programming problem into solvable sub-problems, dramatically reducing the computational complexity compared to classical SVM training.

* **YOLOv8 Object Detection**:

YOLOv8, a state-of-the-art deep learning model for object detection, is responsible for localizing accidents involving cars, bikes, and pedestrians. Its single-shot detection architecture enables fast, real-time frame processing without sacrificing detection accuracy.

The goal is to classify video frames into "accident" or "no accident" categories or identify localized accidents using bounding boxes.

**SVM with SMO Complexity**

Standard SVM training complexity is traditionally around:

O(N2⋅d)

Where:

**N**: Number of training samples.

**d**: Number of features in each sample.

However, by adopting the **SMO optimizer**, which breaks down the optimization into smaller, efficiently solvable problems, the training complexity is **significantly reduced**. In practice, with SMO, the training complexity becomes closer to:

O(N.d)

**Inference** **complexity** (classification of a single frame) remains lightweight at:

O(d)

This makes the SVM with SMO particularly well-suited for rapid, frame-by-frame accident classification with minimal overhead.

*YOLOv8 Detection Complexity*

The YOLOv8 model processes input video frames through a series of convolutional operations. Each convolutional layer operation has a time complexity of:

O(N2 . K2 . C)

Where:

* **N**: The size of the input image feature map.
* **K**: Kernel size (usually 3x3 or 5x5).
* **C**: Number of channels in the image (typically 3 for RGB).

Additionally, YOLOv8 predicts bounding boxes and class probabilities in a single forward pass, ensuring that detection remains efficient even with the added complexity of localization.

Given LLL convolutional layers in the YOLOv8 architecture, the total detection complexity can be approximated as:

A black text on a white background

AI-generated content may be incorrect.

where L is the number of convolutional layers

**Combined Model Complexity**

Integrating SVM and YOLOv8 for crash detection leads to an overall system complexity approximated by:

O(N⋅d+L⋅N2⋅K2⋅C)

This combined structure enables efficient handling of video input streams for real-time accident detection, balancing accuracy and performance.

**Loss Function Complexity**

Both SVM and YOLOv8 leverage the **Mean Squared Error (MSE)** loss function for measuring prediction accuracy during training. The complexity of calculating the MSE loss is:

O(N)

where N is the number of predictions (pixels for regression tasks or bounding boxes for object detection tasks).

Since the loss computation scales linearly with the number of outputs, it does not create a computational bottleneck in the system.

objects. This involves additional complexity due to the localization task, but YOLOv8

optimizes by using a single forward pass.

* ***Detection Speed****: YOLOv8 is designed to balance speed and accuracy, making it capable of processing video frames in real time for accident detection.*
  + - * *Total Model Complexity*

The total complexity of the combined SVM and YOLOv8 detection system is determined by:

O(N2 . K2⋅C+L⋅ N2 . K2⋅C)

Where:

* **L**: Number of YOLOv8 layers (e.g., 53 layers for the original YOLOv8 model).
* **N^2**: Feature map size for video frames.

This combined complexity allows for efficient video processing while ensuring real-time detection.

* + - * *Loss Function Complexity*

The **Mean Squared Error (MSE)** loss function is used in both the SVM and YOLOv8 models for comparing predicted and actual outcomes. The time complexity for calculating the MSE loss is:

O(N2)

Where:

* **N**: Number of output elements (the number of pixels or bounding boxes depending on the model).

This complexity is efficient and scales directly with the number of predictions made by the model.

* + - * *Memory Complexity*

Memory usage is proportional to the number of parameters and the size of the feature maps used in YOLOv8. The memory complexity for YOLOv8 is dominated by the number of feature maps at each layer and the size of the input image:



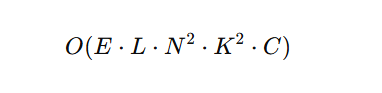
For the **SVM**, memory usage primarily depends on the storage of the support vectors and the number of features used:



The overall system memory usage is efficient for mid-range GPUs (such as NVIDIA GTX 1660), making it suitable for real-time processing in video streams.

Overall Algorithmic Complexity of YOLOv8 and SVM Training

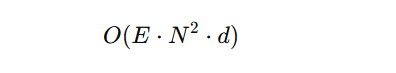
The total training complexity for the system, considering both YOLOv8 and SVM, is determined by forward propagation, backpropagation, and gradient updates over E epochs:



where:

* E is the number of training epochs.
* L is the number of layers in YOLOv8.
* 𝑁2 represents the input image size.
* 𝑘2 is the kernel size.
* C is the number of output channels.

For **SVM**, training complexity is based on the kernel and the number of samples:



While the **Car Crash Detection System** is computationally demanding due to the combination of deep learning and machine learning models, optimizations like batch processing, hardware acceleration, and efficient memory management allow for real-time video processing. The SVM model offers a lightweight classification for accident detection, while YOLOv8 provides fast and accurate localization of accidents in video frames. This hybrid approach balances efficiency with real-time detection, making it well-suited for emergency response applications.

* + 1. *Implementation Complexity*

The implementation complexity of the Car Crash Detection System using SVM and YOLOv8 (with SMO Optimizer) is assessed based on key software engineering metrics: Lines of Code (LoC), number of dependencies, integration complexity, and code modularity.

*Lines of Code (LoC)*

Estimating LOC for this project is difficult at estimation stages this project is of innovative type project

Number of Dependencies

The system utilizes core libraries essential for ML, DL, and web development.

*Table 1: Dependencies of the project*

|  |  |
| --- | --- |
| **Dependency** | **Purpose** |
| OpenCV | Video frame extraction, preprocessing, annotation |
| NumPy / Pandas | Data handling, array manipulations |
| Scikit-learn | SVM implementation with SMO optimizer |
| Ultralytics (YOLOv8) | Pre-trained deep learning model for real-time detection |
| Flask | Web application backend and routing |
| Matplotlib | Visualization (evaluation metrics, confusion matrix) |

**Total dependencies:** ~6–8 major libraries, making it lightweight and easy to install

Integration Complexity

The system consists of several seamlessly interacting components:

* + - * + **Video Preprocessing and Detection** (SVM or YOLOv8) run in a well-structured pipeline.
        + **Web Interface** allows user login, secure upload, and selection between SVM and YOLOv8 models.
        + **Results** are displayed with annotated frames and optionally stored.
        + **Optional Database Logging** (for logs or results) can be easily added via SQLite or MongoDB.

Integration complexity is **moderate**, with well-defined APIs and minimal coupling between components.

Code Modularity

The project follows a modular approach, allowing easy testing, updates, or replacement of

specific components.

*Table 2: Code Modularity*

|  |  |
| --- | --- |
| Module | Description |
| Preprocessing Module | Handles frame extraction, resizing, grayscale conversion |
| SVM Module | Uses SMO-optimized Support Vector Classifier for frame-wise accident classification |
| YOLOv8 Module | Integrates Ultralytics’ YOLOv8 for real-time object detection of crash events |
| Inference Module | Unified handler for running prediction on uploaded/recorded videos |
| Evaluation Module | Calculates accuracy, precision, recall, and visualizes confusion matrix |
| Web Interface Module | Flask-based login/auth, video upload, and model selection UI |
| Preprocessing Module | Handles image normalization, resizing, histogram equalization |

The car crash detection system shows moderate to high implementation complexity, primarily due to

the dual-model architecture and real-time video processing. However, the well-structured, modular design ensures scalability, flexibility, and ease of integration with future smart traffic or emergency response systems. The combination of SVM (with SMO) and YOLOv8 enhances both interpretability and real-time performance, making the solution intelligent, reliable, and impactful.

* + 1. *Resource Complexity*

Resource complexity measures how much compute, memory, storage, and cloud infrastructure the system needs both during training and when deployed at scale. For our car crash detection system, built around a hybrid architecture combining SVM withSMO optimizer and YOLOv8, the design choices directly impact how heavy (or light) the resource demands are.

*Hardware Requirements*

The system architecture consists of two core models:

* + - * + **SVM (Support Vector Machine)** with **SMO optimizer** for lightweight, frame-by-frame crash classification.
        + **YOLOv8 (You Only Look Once, version)** for real-time object detection, recognizing crash scenarios like car-to-car, car-to-bike, or car-to-person collisions.

Key Architectural Features:

* **SVM +SMO:** Very efficient SMO (Sequential Minimal Optimization) significantly reduces training overhead by solving the SVM problem in small chunks, avoiding heavy matrix computations.
* **YOLOv8:** Deep learning model capable of detecting car crash events (e.g., car-car, car-bike, car-person) in video streams with bounding boxes and class labels.

Resource-Efficient Traits:

* **SVM Frame-Based Classification**: Minimal memory and CPU usage, easily handled even by mid-range machines.
* **YOLOv8 Inference with PyTorch**: Optimized for GPUs; runs real-time detection at low latencies.
* **Modular Flexibility**: Depending on the deployment needs, users can favor **SVM** for lightweight applications or **YOLOv8** for high-accuracy environments.

*Table 3: Hardware Specification of SVM &YOLOV8 Training*

|  |  |
| --- | --- |
| **Component** | **Recommended Specs** |
| **CPU** | **Intel i5 / Ryzen 5 (4-core or higher)** |
| **GPU** | **NVIDIA GTX 1650 / T4 or higher (for YOLOv8)** |
| **RAM** | **8–12 GB** |
| **Disk I/O** | **SSD preferred (≥500 MB/s read/write)** |

Mathematical Complexity:

Each convolution operation is defined as:

* **SVM with SMO:**
* Very low complexity per iteration.
* Only needs to solve small optimization subproblems at each step, instead of handling the full kernel matrix.
* Effective training time is almost linear relative to the number of data points.
* **YOLOv8 Convolution Layers:**
* Each convolutional layer processes based on:

Computation per layer = K×K×Cin​×Cout​×H×W

where:

* K = Kernel size
* Cin​, Cout= Input/Output channels
* H, W= Image height and width

**Cloud Infrastructure Support**

The system is flexible enough to run either locally or on the cloud, depending on resource availability:

* **SVM (SMO-based)**:
  + Best suited for **edge devices** or **small VMs**.
  + Can run entirely CPU-based, no GPU mandatory.
* **YOLOv8**:
  + Performs best on **GPU-accelerated cloud services**.
  + Designed for **batch inference** and **real-time processing** at large scale.
* **SVM** with SMO is extremely **cost-effective**, ideal for low-budget setups or embedded systems.
* **YOLOv8** is heavier but justified where real-time crash detection at scale matters (e.g., traffic surveillance centers or smart city management).

*Cloud Infrastructure*

The car crash detection system is optimized for both local and cloud-based deployment:

* **SVM-based detection** is ideal for edge devices or low-cost VM setups.
* **YOLOv8-based detection** benefits from GPU-enabled cloud environments for faster batch inference and real-time detection.

Deployment Scenarios:

* AWS g4dn.xlarge (T4 GPU)
* Google Collab Free/Pro
* Azure NC-Series (for scaling up inference)

*Table 4: Cloud Resource Profile*

|  |  |  |
| --- | --- | --- |
| **Attribute** | **SVM + SMO** | **YOLOv8** |
| VRAM Usage | ~0.5–1 GB (CPU preferred) | ~6–8 GB (GPU-accelerated) |
| CUDA Core Usage | Not Applicable (pure CPU) | ~40–60% typical |
| Training Time | ~1–2 mins (small dataset, e.g., 500–1000 samples) | ~5–8 mins per epoch (1000–1500 images) |
| Cost per Training Run | ₹5–₹15 (minimal CPU billing) | ₹60–₹100 (depends on GPU hours) |
| Inference Time/Frame | ~0.08–0.12 sec (CPU) | ~0.03–0.15 sec (GPU) |

*4.4 Risk Management*

Consistent with the risk management guidelines suggested by Cara Fisherman in her Software Engineering framework, risks for the YOLOv8 + SVM-based car crash detection system have been systematically identified and grouped under four main categories: Technical, Data and Security, Financial and Resource, and Ethical and Regulatory risks. Each category is detailed below to ensure early visibility and proactive mitigation strategies.

* + 1. *Risk Identification*

Potential risks in the project can be classified into the following categories:

1. *​Technical Risks*
   * **Model Performance Uncertainty:** As highlighted in Fisherman’s principles, algorithmic systems can underperform under real-world constraints. YOLOv8 may misclassify crash scenarios, particularly in conditions of poor illumination, occluded views, or unconventional camera angles.
   * **SVM Generalization Issues:** Support Vector Machines (SVM) may not robustly adapt to unseen accident patterns, especially when trained on imbalanced datasets, which aligns with Fisherman's emphasis on validation risks.
   * **System Integration Complexity:** Integrating YOLOv8 and SVM models with live traffic feeds, existing control systems, or real-time alert mechanisms may encounter compatibility or latency challenges, reflecting typical integration risks discussed in the book.
2. *​Data and Security Risks*
   * **Dataset Completeness and Bias:** A key risk noted in data-driven project environments (Fisherman, 2019) is reliance on incomplete or non-representative datasets, potentially degrading model reliability and fairness.
   * **Privacy Breaches:** Capturing and processing real-time CCTV or dashcam footage introduces significant privacy concerns, a sensitive issue also emphasized under data

security governance.

* + **Data Loss or Corruption:** Annotated crash data could be inadvertently lost or corrupted during model training or storage, affecting reproducibility and validation.

1. *​Financial and Resource Risks*
   * **Hardware Dependency:** Real-time inference with YOLOv8 typically demands high-end GPU hardware, thereby elevating initial investment and operational costs consistent with resource-intensive project risks described by Fisherman.
   * **Cloud and Storage Costs:** Large-scale storage and computational processing of video datasets may lead to escalating cloud service charges, creating pressure on

budget allocations.

* + **Risk of Budget Overruns:** Unexpected expenses arising from licensing fees, infrastructure scaling, or ongoing maintenance activities could result in cost overruns if not meticulously forecasted.

1. *​Ethical and Regulatory Risks*
   * **Surveillance Misuse:** As Fisherman warns about ethical drift, there is a risk that the detection system could be misappropriated for extensive vehicle tracking or mass surveillance purposes.
   * **Non-compliance with Legal Frameworks:** Use of real-time footage must comply with international and local data protection laws (e.g., GDPR), and lapses here could attract serious penalties.
   * **Public Perception and Trust:** Even with compliance, public misinterpretation of the system as an invasion of privacy may spark negative sentiment, undermining acceptance and deployment.
     1. Risk Analysis

Following the structured risk assessment model suggested by Cara Fisherman in her *Software Engineering* methodology, each identified risk is evaluated based on two primary dimensions:

* **Likelihood** (the probability of occurrence) and
* **Impact** (the severity of its effect on project goals such as timeline, cost, performance, and development hours).

This analysis allows project stakeholders to prioritize mitigation strategies early, ensuring smoother development cycles and minimizing unplanned rework or resource drains.

The results of the risk analysis for the YOLOv8 + SVM-based car crash detection system are summarized below:

*Table 5: Risk Analysis of the project*

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Risk Description** | **Likelihood** | **Impact** |
| Technical | YOLOv8 misclassifies crashes | Medium | High |
|  | SVM overfitting or underperformance | Medium | High |
|  | Integration with live feeds | Medium | Medium |
| Data/Security | Dataset bias or incompleteness | Medium | High |
|  | Privacy issues with footage | High | High |
|  | Dataset loss or corruption | Low | High |
| Financial | High GPU/cloud costs | High | Medium |
|  | Unexpected hardware or license costs | Medium | Medium |
| Ethical/Regulatory | Misuse or regulatory breach | Medium | High |
|  | Public resistance to surveillance | Medium | High |

* + 1. *Overview of Risk Mitigation, Monitoring, and Management*

In line with the risk handling practices proposed by Cara Fisherman in *Software Engineering*, this project adopts a structured approach to mitigate, monitor, and manage identified risks throughout the system’s lifecycle. Effective risk management is critical to ensuring that the YOLOv8 + SVM-based car crash detection system is developed, deployed, and maintained successfully, with minimal disruptions to schedule, quality, and budget.

The following strategies have been designed for each risk category:

*Technical Risk Management*

* Optimization of computational efficiency:
  + To mitigate technical performance risks, preprocessing pipelines will be optimized, ensuring diverse environmental conditions are represented during model training. Thresholds within the YOLOv8 model will be fine-tuned, and ensemble learning techniques will be applied to strengthen the robustness of the SVM classifier.
  + **Monitoring:** Regular performance benchmarking across varied crash scenarios.  
    **Management:** Iterative refinement during development sprints to ensure accuracy improvements without excessive development overhead.

*Data and Security Risk Management*

* **Ensuring Data Quality and Protection:**
* A broad spectrum of crash footage will be collected to minimize dataset bias. Sensitive information in video footage will be anonymized using blurring and encryption techniques. Datasets and model checkpoints will be securely stored with routine backups in reliable cloud services.  
  **Monitoring:** Periodic dataset audits to assess diversity and compliance.  
  **Management:** Integrate data security checkpoints into the DevOps pipeline to catch vulnerabilities early.

*Financial and Resource Risk Management*

* **Controlling Costs Through Smart Resource Use:**
* Financial risks will be addressed by leveraging transfer learning to reduce training time and resource needs. Additionally, cost-effective GPU services, such as Google Colab Pro and AWS Spot Instances, will be utilized to avoid excessive infrastructure expenses.  
  **Monitoring:** Monthly cost tracking dashboards.  
  **Management:** Flexibility to switch between cloud providers or configurations based on resource usage analytics.

*Ethical and Regulatory Risk Management*

* **Maintaining Ethical Standards and Legal Compliance:**
  + To address ethical and regulatory risks, the system will strictly adhere to recognized AI ethics guidelines. System access will be restricted to authorized traffic authorities only. Furthermore, all data handling will comply with regulations such as GDPR, ensuring transparent and responsible use of surveillance technologies.  
    **Monitoring:** Conducting regular internal compliance checks and ethics reviews.  
    **Management:** Updating operational policies based on evolving regulatory landscapes.

* + 1. **Technical Risks**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reference ID** | **Risk Description** | **Potential Impact** | **Identified By** | **Mitigation Strategy** |
| T-001 | YOLOv8 model underperforms in real-time under low lighting or occlusion | Reduced detection accuracy, missed crashes | ML Development Team | Train with diverse and augmented datasets, perform real-world testing |
| T-002 | SVM misclassifies non-crash events as crash | False alarms, unreliable output | AI/ML Engineers | Use proper feature engineering, balance datasets, and tune model parameters |
| T-003 | Flask server crashes under high video stream load | Slow or failed detection delivery | Backend Engineers | Optimize server performance, enable async processing, and implement rate limiting |
| T-004 | GPU overheating or insufficient hardware resources | System instability or failure | Hardware Team | Use cooling systems, monitor GPU usage, and include hardware redundancy |

**2.** **Operational Risks**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reference ID** | **Risk Description** | **Potential Impact** | **Identified By** | **Mitigation Strategy** |
| O-001 | Cloud service or internet downtime | System unavailable for real-time monitoring | Cloud Operations Team | Use edge computing fallback or hybrid cloud solutions |
| O-002 | Lack of skilled personnel to maintain and operate the system | Project delays or system mismanagement | Project Manager | Provide training programs, hire or consult experienced developers |
| O-003 | Hardware maintenance or sensor calibration not done regularly | System inaccuracy or hardware damage | Hardware Maintenance Team | Schedule routine checks, set up monitoring and alert systems |
| O-004 | Integration issues between YOLO, SVM, and Flask | Functionality mismatch or data handling errors | DevOps Team | Test end-to-end workflows regularly, maintain modular and documented code |

1. **Security Risks**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reference ID** | **Risk Description** | **Potential Impact** | **Identified By** | **Mitigation Strategy** |
| S-001 | Unauthorized access to video feeds or stored data | Data leaks, user privacy violations | Security Team | Encrypt data, enforce user authentication, and secure endpoints |
| S-002 | Insecure API endpoints or weak Flask configurations | System vulnerability to cyberattacks | DevSecOps Team | Perform regular penetration tests, use HTTPS and secure headers |
| S-003 | Mishandling of personally identifiable data (PII) from crash footage | Legal consequences, loss of stakeholder trust | Compliance Officer | Implement data anonymization, follow data protection laws (like GDPR) |
| S-004 | Lack of logging and audit trails | Difficulty in tracking unauthorized changes or breaches | Security Team | Set up secure logging mechanisms and access audit logs |

1. **Financial Risks**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reference ID** | **Risk Description** | **Potential Impact** | **Identified By** | **Mitigation Strategy** |
| F-001 | High cost of cloud services and GPU resources | Budget overruns | Finance Team | Use free-tier platforms during development, monitor usage, and optimize deployments |
| F-002 | Hardware replacement due to wear, damage, or upgrades | Unexpected capital expense | Hardware Procurement Team | Plan preventive maintenance, buy modular and repairable hardware |
| F-003 | Long-term storage and data transfer costs | Increased operational cost over time | Project Management | Archive older data, apply retention policies, and compress video feeds |
| F-004 | Dependency on paid third-party APIs or libraries | Cost escalation or licensing issues | Legal & Dev Teams | Prefer open-source solutions, or choose APIs with transparent and scalable pricing |

***4.5Project Schedule***

Effective project scheduling is a cornerstone of any successful software engineering or AI-driven initiative. In the context of this research, a comprehensive and strategically segmented project plan was devised to ensure the timely execution of deliverables while maintaining high quality in model development, experimentation, and reporting. The following schedule was crafted using agile-influenced methodologies and academic research workflows, structured to facilitate iterative learning, continuous model improvement, and real-world applicability.

* + 1. *Project Task Set*

The task set is organized across six major development phases, each designed to reflect the lifecycle of an AI research initiative starting from ideation and data engineering, through to modeling, optimization, evaluation, and dissemination.

*Phase I: Foundation & Requirements Engineering (Aug 2024 – Aug 2024)*

* Task: Dataset Selection (Accident/Non-Accident Scenarios)

This phase began with defining the project scope and identifying suitable datasets related to real-world road scenes. Publicly available crash-related video/image datasets were explored, along with self-curated dashcam footage. Data was annotated to differentiate between accident and non-accident frames for classification tasks. Preprocessing pipelines were set up using OpenCV, and standard annotation was ensured using tools like LabelImg.

*Phase II: Baseline Architecture Development (Sep 2024 – Oct 2024)*

* Task: Model Development & Training (YOLOv8)

During this phase, the YOLOv8 object detection model was implemented and configured for real-time detection of accident-related cues (e.g., vehicle collisions, unusual motions). YOLOv8 framework was used for model development, and training was performed on the curated dataset. To enhance model generalization, image augmentation techniques such as brightness shifts, rotations, and contrast adjustments were applied.

*Phase III: QA and Baseline Performance Review (Oct 2024 – Nov 2024)*

* Task: Testing & Evaluation

YOLOv8’s performance was evaluated using standard object detection metrics like Precision, Recall, mAP, and F1 Score. Special focus was placed on challenging cases such as vehicle pile-ups, multi-vehicle crashes, and low-light/nighttime conditions. The insights from this evaluation helped identify strengths, limitations, and improvement opportunities for the next phase.

*Phase IV: Research Scope Expansion & Strategic Pivot (Nov 2024 – Dec 2024)*

* Task: Literature Survey & Gap Analysis

An in-depth literature review was conducted to explore existing research on combining deep learning and traditional machine learning (e.g., YOLO + SVM pipelines). This helped identify research gaps and positioned the project within the evolving domain of real-time smart surveillance systems.

*Phase V: Dataset Expansion & Knowledge Dissemination (Dec 2024 – Jan 2025)*

* Task: Road Scene Data Collection & Augmentation

To enhance robustness, additional footage under diverse conditions (fog, rain, nighttime, etc.) was collected. Synthetic augmentation techniques were applied to simulate rare crash-like events in extreme scenarios.

*Phase VI: Final Model Training, Validation, and Optimization (Jan 2025 – Feb 2025)*

* Task: Model Integration & Testing (YOLOv8 + SVM)

The final system pipeline was built by integrating YOLOv8 (for object detection) and SVM (for crash classification). The entire pipeline was containerized for smoother deployment and scalability testing.

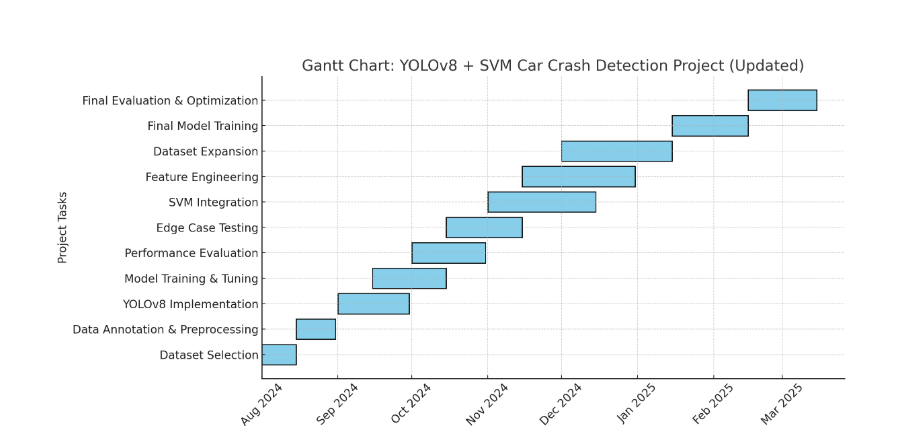
*Phase VII: Research Scope Expansion (Feb 2025 – April 2025)*

* Task: Research Paper

The system architecture, experimental results, and performance benchmarks were compiled into a research paper aimed at submission to AI/ITS conferences (e.g., Scopus-indexed).

* + 1. *Timeline Chart*

The project timeline was modeled using Agile Gantt Scheduling, visualized in the following chart:



*Figure 2: Timeline Gantt Chart Key Observations from the Timeline:*

* The project is divided into well-defined phases ensuring a smooth transition from dataset collection to model deployment.
* Strategic integration of the YOLOv8 and SVM hybrid approach is planned during the core development phase.
* Parallel execution of tasks like conference preparation and dataset expansion optimizes time and enhances project output.
* Iterative evaluation and optimization phases ensure improved performance and real-world applicability.
  1. *Team Organization*

A well-defined team structure ensures efficient task execution, streamlined collaboration, and effective distribution of responsibilities throughout the development lifecycle of the YOLOv8 + SVM-based car crash detection system.

For the successful execution of this project, our team followed a collaborative, role-specific structure inspired by industry-standard practices in AI and software development. Each member was assigned tasks aligned with their strengths, ranging from data acquisition and preprocessing to model development, integration, and final evaluation. Regular sync-ups and agile stand-ups ensured continuous progress, risk management, and adaptability to evolving project needs.

* + 1. *Team Composition*

Our team comprised four final-year engineering students who collaboratively contributed to the ideation, design, development, testing, and deployment of the **YOLOv8 + SVM-based car crash detection system**. The project was developed under the insightful guidance of our faculty advisor, who provided consistent mentorship, conducted technical reviews, and ensured research alignment throughout the project lifecycle.

*Table 6: Team Composition*

|  |  |
| --- | --- |
| **Name** | **Role** |
| Member 1 | Schedule all plan of project. Manage the team. Divide the work in team. The deadline is assigned. Consider the all requirements and as per requirement gathering develop the module. |
| Member 2 | Arrange the developing tool i.e platform, language, software, hardware and make the system architecture. Write the code of every module and apply the appropriate use case to test the plan. |
| Member 3 | Test each module if result is correct then combine all module and again test. After deployment manages the feedback report and correct some corrections. |
| Member 4 | All documentation related activities will perform. |

* + 1. *Communication & Review Flow*
* **Weekly Team Sync-ups**: Discussed progress, model performance, and assigned upcoming tasks.
* **Biweekly Faculty Reviews**: Ensured technical soundness and relevance to the project objectives.
* **GitHub for Version Control**: Maintained the YOLOv8 + SVM codebase collaboratively.
* **Collaborative Documentation**: Used Google Docs and Overleaf for maintaining logs, papers, and the final report.
  + 1. *Contribution Summary*

Each team member contributed significantly based on their strengths, enabling smooth execution of the project.

* **60% collaborative work**: Model evaluation, testing, crash detection validation, and report drafting.
* **40% domain-specific efforts**: YOLOv8 implementation, SVM tuning, UI development, and deployment setup.

The modular task division ensured parallel progress and timely delivery, reflecting real-world AI development practices.

**Chapter 05**

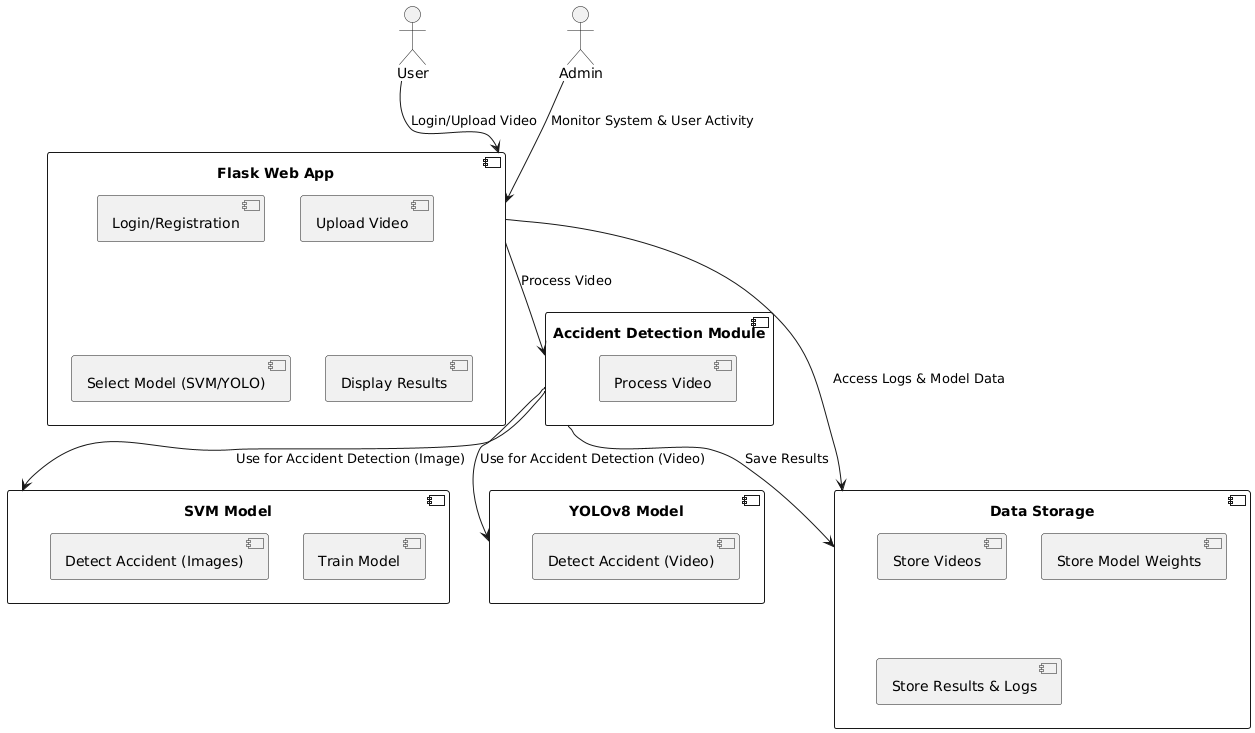
# SYSTEM DESIGN

* 1. *Proposed System Architecture / Block Diagram*

The System Architecture of the Road Accident Detection System is designed to process video footage and detect accidents using two methods: Support Vector Machine (SVM) and YOLOv8 (You Only Look Once). The system is built with a web-based interface using Flask and consists of several components that work together to achieve the goal of accident detection.

Block Diagram of the System

Below is a high-level block diagram of the system, illustrating the workflow from image input to density map generation and real-time analysis**.**



*Figure 3: System Architecture*

**Key Components of the System Architecture**

1. **User Interface (Flask Web App)**:

* **Login/Registration**: Users need to log in to access the system.
* **Video Upload**: After login, the user can upload videos for analysis.
* **Model Selection**: The user can choose between **SVM** and **YOLOv8** for accident

detection.

* **Result Display**: After processing the video, the results of the accident detection

are shown to the user.

1. **Accident Detection Module**:

* This module handles the video processing and accident detection. It contains two

main methods:

* **SVM Model**: Analyzes images (video frames) to classify them as

"Accident" or "Non-Accident."

* **YOLOv8 Model**: Detects specific types of accidents (car-to-car, car-to-

bike, or car-to-pedestrian).

1. **Data Storage**:

* Stores uploaded videos, model weights, and the results from the detection process.
* This component helps keep track of processed videos and their corresponding

detection outcomes.

1. **Admin Monitoring**:

* Admins can monitor the system's performance and view user activity.
* They also have access to stored data and can check logs for system health.

**How the System Works:**

* + 1. The **User** logs into the **Flask Web App** and uploads a video.
    2. The **Flask Web App** sends the video to the **Accident Detection Module**, where

either the **SVM Model** or **YOLOv8 Model** processes the video.

* + 1. The **Accident Detection Module** classifies the video frames or detects accidents

using the chosen model.

* + 1. The results are saved in **Data Storage** and displayed to the **User**.
    2. The **Admin** monitors the entire system to ensure everything is working smoothly.

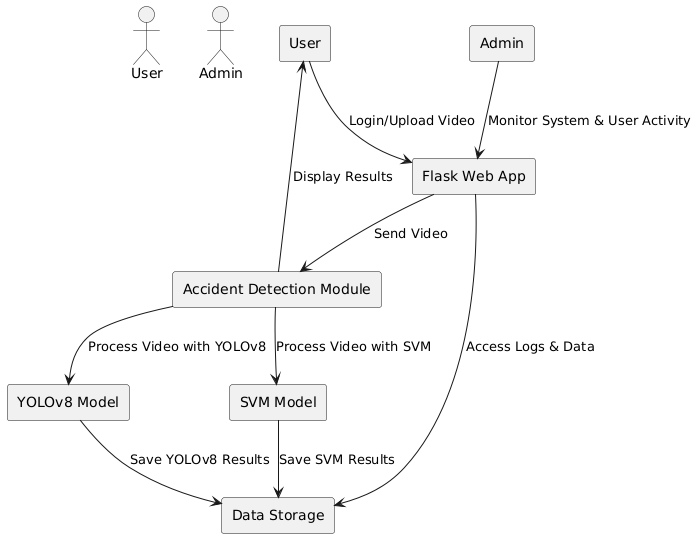
This architecture helps the system function efficiently by separating different tasks into distinct components, making the system easier to maintain and scale. The **Flask Web App** serves as the entry point for the user, while the **Accident Detection Module** processes the videos and detects accidents. The results are stored for later reference and analysis.

* 1. *Dataset*

*Dataset Description*

The **Berkeley DeepDrive (BDD)** dataset is a large-scale dataset designed for training and testing models in various computer vision tasks, such as object detection, segmentation, and tracking. It's particularly useful for autonomous driving and related research, which includes tasks like accident detection and analysis. Here’s a general description of the dataset:

* + - **Publisher**: Berkeley AI Research (BAIR)
    - **Type**: Image and video dataset
    - **Size**: Over 100,000 annotated images and 20,000 video sequences.
    - **Categories**: The dataset includes multiple types of data, such as:
    - **Driving Scenarios**: Includes various road types, weather conditions, and times of the day.
    - **Traffic Objects**: Annotations for vehicles, pedestrians, cyclists, traffic signs, traffic lights, etc.
    - **Event Data**: Specific annotations for accidents or incidents.
    - **Annotations**: The dataset includes annotations such as bounding boxes for object detection, semantic segmentation maps, instance segmentation, and key point detection for pedestrian pose estimation.
    - **Video Sequences**: Along with individual frames, BDD provides video sequences with labelled timestamps, making it especially useful for tracking and event-based analysis, such as accidents or sudden changes in the driving environment.
    - **Formats**: The dataset is available in common formats like JPEG, PNG for images, and MP4 for videos. Annotations are typically provided in JSON or XML format.

*Data Flow in the System*

*Figure 4: Data Flow in System*

A **Data Flow Diagram (DFD)** is a visual representation of how data flows through a system. It helps in understanding the system’s processes, data storage, and how the different components interact with each other. In the context of the **Road Accident Detection System**, the DFD illustrates the flow of information between users, the system, and the data storage.

**Core Components of the DFD:**

1. **External Entities**:
   * These are the actors that interact with the system but are outside its boundaries.
   * For the **Road Accident Detection System**, the **User** and **Admin** are external

entities.

* + - **User**: Uploads videos for accident detection and receives results.
    - **Admin**: Monitors the system, manages users, and accesses system logs.

1. **Processes**:
   * These are the core components where the data is processed, and operations are performed.
   * **Flask Web App**: The interface where users log in, upload videos, and choose detection models (SVM or YOLOv8).
   * **Accident Detection Module**: The core process that handles video analysis. It communicates with the **SVM Model** or the **YOLOv8 Model** to detect accidents in the uploaded video.
   * **SVM Model**: Classifies video frames as accident or non-accident using machine learning.
   * **YOLOv8 Model**: Detects specific types of accidents (car-car, car-bike, car-person) using deep learning and object detection.
2. **Data Stores**:
   * Data stores hold information that is needed for the system to function.
   * **Data Storage**: Stores the uploaded videos, processed results, model weights,

and logs.

1. **Data Flows**:
   * The arrows in the DFD represent the flow of data between external entities,

processes, and data stores.

* + - **User** uploads videos, which flow into the **Flask Web App**.
    - The **Flask Web App** sends the video data to the **Accident Detection**

**Module** for processing.

* + - The detection results from either **SVM** or **YOLOv8** are stored in **Data**

**Storage**.

* + - **Admin** accesses the **Flask Web App** to monitor the system and

retrieve logs from the **Data Storage**.

**Summary of the DFD for Road Accident Detection System:**

* **User uploads videos** for analysis.
* **Flask Web App** manages user interaction and sends videos to the **Accident**

**Detection Module**.

* The **Accident Detection Module** processes videos using either **SVM** or

**YOLOv8**, detecting accidents.

* **Results are stored in Data Storage**, and **Admin** monitors the system and

accesses logs for maintenance.

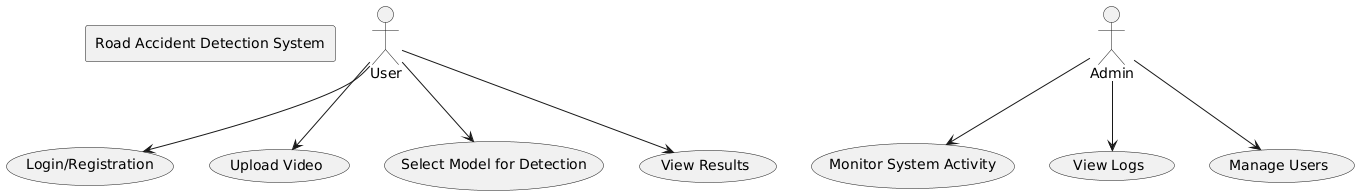
The DFD helps visualize how data moves through the system and shows the interaction between different components, making it easier to understand the flow of operations.

* 1. *UML Diagrams*

The Unified Modeling Language (UML) Diagrams provide a structured representation of the system.

1. *Use Case Diagram*

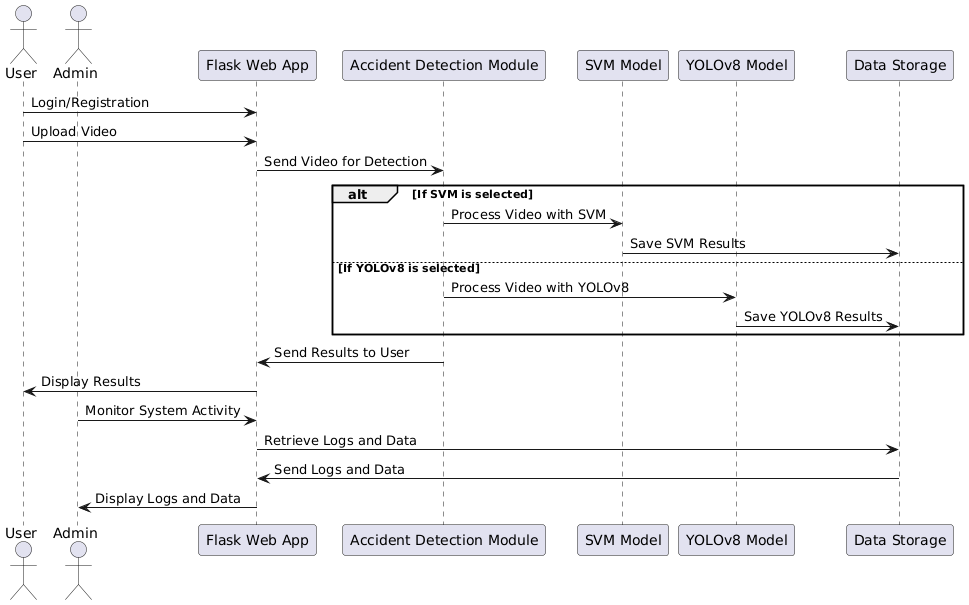
The Use Case Diagram represents interactions between users and the system.



*Figure 5: Use Case Diagram*

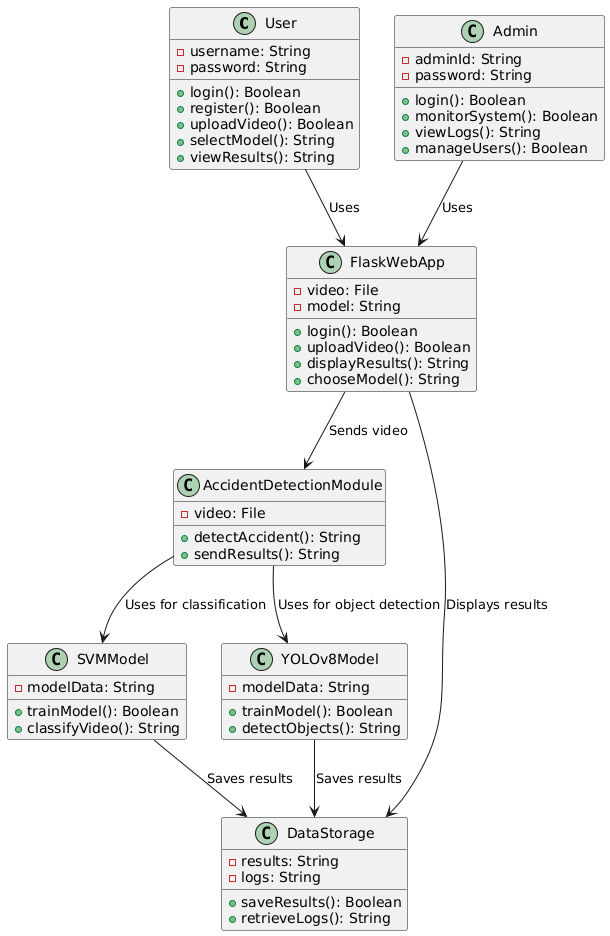
1. *Sequence Diagram*

The Sequence Diagram illustrates the interaction between system components.



*Figure 6: Sequence Diagram*

.

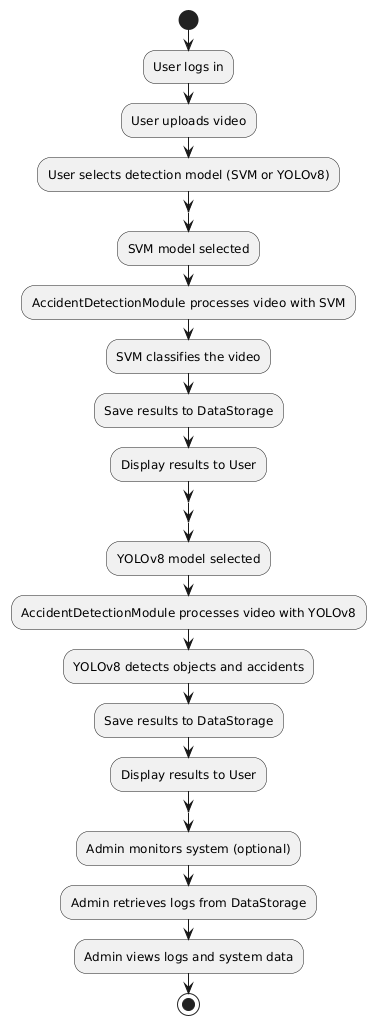


1. *Class Diagram*

The Class Diagram represents the system’s object-oriented structure.

*Figure 7: Class Diagram*

1. *Activity Diagram*

****

*Figure 8: Activity Diagram*

**Chapter 06**

# PROJECT IMPLEMENTATION

### Overview of Project Modules

The **Road Accident Detection using SVM and YOLOv8** project aims to develop an intelligent system that automatically detects accidents in recorded traffic videos. This project combines two advanced machine learning techniques **Support Vector Machine (SVM)** and **YOLOv8 (You Only Look Once)** for video-based accident detection, offering a dual approach for comparison.

*Key Components of the Project:*

1. **Support Vector Machine (SVM) for Accident Classification:**
   * SVM is a supervised machine learning algorithm that will be trained to classify video frames as either **accident** or **non-accident**. The SVM model will be optimized using **Sequential Minimal Optimization (SMO)** to enhance its performance.
   * The system will use a labeled dataset of images to train the model to recognize patterns associated with accidents in video frames.
2. **YOLOv8 for Object Detection:**
   * YOLOv8, a state-of-the-art deep learning algorithm, will be used to detect specific accident scenarios, including **car-car collisions**, **car-bike accidents**, and **car-person accidents**.
   * The YOLO model will be trained on video frames to detect various objects involved in accidents, making it highly effective for real-time object detection.
3. **Web Application Interface:**
   * The project will be implemented as a **Flask-based web application**, providing an intuitive user interface.
   * Users will be able to:
     + **Log in** and **register** to access the system.
     + Upload recorded **traffic videos** for accident detection.
     + Choose between the **SVM model** or **YOLO model** for accident analysis.
     + View results with marked frames indicating detected accidents.

***Functionality:***

* **Login and Registration**: Secure authentication system for user access.
* **Model Selection**: Users can choose either the SVM model or the YOLO model to analyze the uploaded video.
* **Video Upload**: Users can upload traffic videos for analysis.
* **Accident Detection**: Both models will analyze the video to detect accidents. Results will be displayed, showing the frames where accidents are detected.

***Objective:***

The goal of this project is to automate the detection of road accidents from video data using **AI-driven algorithms**. By offering both the **SVM classification** and **YOLO object detection** models, the system provides a comprehensive comparison of different machine learning techniques for accident detection. The automated nature of the system aims to improve road safety by enabling faster detection and response times, contributing to **intelligent transportation systems (ITS)**.

***Significance:***

* **Efficiency**: Automated accident detection can reduce human intervention and improve response times.
* **Scalability**: The approach can be adapted for use in real-time traffic surveillance systems.
* **Safety**: The system contributes to the improvement of road safety by identifying accidents early, potentially saving lives and reducing road hazards.

### Tools and Technologies Used

**YOLOv8 (You Only Look Once version 8)**:

* A state-of-the-art real-time object detection model that processes images quickly while maintaining high accuracy. YOLOv10 is particularly suitable for detecting multiple objects in a single frame, making it ideal for self-checkout applications.

**Flask**:

* A lightweight web framework for Python that is used to develop the backend of the web application. Flask allows for the easy handling of HTTP requests, image uploads, and integration with the object detection model.

**HTML/CSS/JavaScript**:

* Standard web technologies used for building the user interface of the application. HTML provides the structure, CSS handles the styling, and JavaScript is utilized for dynamic content and user interactions.

**OpenCV**:

* An open-source computer vision library used for image processing tasks. OpenCV facilitates the manipulation and analysis of images before they are passed to the YOLO model, ensuring optimal input for detection.

**Database (e.g., SQLite or PostgreSQL)**:

* A relational database used to store product information, including item names, prices, and categories. This database allows the system to retrieve pricing data quickly during the billing process.

**TensorFlow or PyTorch**:

* Deep learning frameworks that support the training and deployment of the YOLOv10 model. These frameworks provide tools for model optimization and inference, ensuring efficient performance in real-time detection.

### Hardware Specifications

* + - * System Type : 64-bit or 32-bit
      * Processor : Intel core i5
      * Storage Capacity : 256 GB
      * RAM : 4GB (Min)
      * I/O Devices : Mouse and Keyboard

### Software Specifications

* + - * Operating System : Windows 11
      * Coding Language : Python
      * Data Base : MySQL/SQLite
      * IDE : VS Code
      * Browser : Google Chrome

**Chapter 07**

# SOFTWARE TESTING

* 1. *Type of Testing*

Ensuring the robustness, reliability, and quality assurance of the crowd density estimation system is crucial for its real-world deployment and continuous usability. These testing methods included functionality, accuracy, performance, security, usability, and integration, and were essential in delivering a robust application. Below is an expanded overview of all testing types applied:

**Types of Testing:**

Along with the type of testing also mention the approach to be followed for the testing, that is, Manual Testing or Automated Testing. Use Automated Testing Plan for planning automation activities in details. The different types of testing that may be carried out in the project are as follows:

* + - **Unit Testing:**

Individual components are tested independently to ensure their quality. The focus is to uncover errors in design and implementation, including.

* + - * Data structure in component
      * Program logic and program structure in a component
      * Component interface
      * Functions and operations of a component
    - **Integration Testing :**

A group of dependent components are tested together to ensure their quality of their inte- gration unit. This approach is to do incremental integration to avoid “big bang” problem. That is when the entire program is put together from all units and tested as a whole. The big-bang approach usually results in chaos which incremental integration avoids. Incre- mental integration testing can be done in two different way top down and bottom up. Then there is also the possibility of regression integration.

The top down integration is when modules are integrated by moving downwards through the

control hierarchy, beginning with the main control module. Modules subordinate to the main control module are incorporated into main structure n either depth-first or breadth-first manner. The top down integration verifies major controls or decision points early in the test process. If major control problems do exist, early recognition is essen- tial. Bottom-up integration testing begins construction and testing with the lowest levels in the program structure. Because modules are integrated from the bottom-up, processing required for modules subordinate to a given level is always available and the need for test stubs is eliminated.

The focus is to uncover:

* Design and construction of software architecture
* Integrated functions or operations at sub-system level
* Interfaces and interaction and/or environment integration
  + - **System Testing :**

The system software is tested as a whole. It verifies all elements mesh properly to make sure that all system functions and performance are achieved in the target environment. The focus areas are:

* + - * System functions and performance
      * System reliability and recoverability (recovery test)
      * System behavior in the special conditions (stress and load test)
      * System user operations (acceptance test/alpha test)
      * Hardware and software integration collaboration
      * Integration of external software and the system.
    - **Validation Testing:**

Validation can be defined in many ways, but a simple definition is that succeeds when software

functions in a manner that can be reasonably expected by the customer. Soft- ware validation is achieved through a series of black-box tests that demonstrate conformity with requirements. A test plan outlines the classes of tests to be conducted and a test procedure defines specific test cases that will be used to demonstrate conformity with requirements. Both the plan and procedure are designed to ensure that all functional requirements are satisfied, all behavioral characteristics are achieved, all performance requirements are attained, documentation is correct, and human engineered and other re quirements are met.

* + - **White Box Testing:**

For white-box include: structural, glass-box and clear-box

White box testing is much more expensive than black box testing. It requires the source code to be produced before the tests can be planned and is much more laborious in the determination of suitable input data and the determination if the software is or is not cor- rect. This testing is concerned only with testing the software product; it cannot guarantee that the complete specification has been implemented.

* + - **Black Box Testing:**

Black-box test design treats the system as a “black-box”, so it doesn’t explicitly use knowledge of the internal structure. Black-box test design is usually described as fo- cusing on testing functional requirements. Synonyms for black box include: behavioral, functional, opaque-box, and closed-box. Black box testing is concerned only with test- ing the specification; it cannot guarantee that all parts of the implementation have been tested. Thus black box testing is testing against the specification and will discover faults of omission, indicating that part of the specification has not been fulfilled.

* + - **GUI Testing:**

Graphical User Interface (GUIs) present interesting challenges for software engineers. Because of reusable components provided as part of GUI development environments, the creation of the user interface has become less time consuming and more precise. But, the same time, the complexity of GUIs has grown, leading to more difficulty in the design and execution of the test cases. Because many modern GUIs have the same look and same feel, a series of test cases can be derived.

* 1. *Test Cases & Test Results*

**Software to be tested:**

After implementation of project software will be tested by tester.

*Table 7: Test Case Table*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Description** | **Test Steps** | **Expected Result** | **Actual Result** | **Status** | **Remarks** |
| TC001 | Login functionality | 1. Open login page. 2. Enter valid credentials. 3. Click on 'Login'. | User should be successfully logged in and directed to the main dashboard. | User can login and go to main dashboard | Passed | Login test successful with valid credentials. |
| TC002 | Incorrect Login attempt | 1. Open login page. 2. Enter invalid credentials. 3. Click on 'Login'. | System should show an error message: "Invalid username or password". | System is showing error message | Passed | Invalid login message displayed correctly. |
| TC003 | Video Upload | 1. Open video upload page. 2. Select and upload a video file. | Video should be successfully uploaded and the system should confirm that. | System confirms the video uploading | Passed | Video upload works fine with MP4 format. |
| TC004 | Model Selection (SVM) | 1. After video upload, select "SVM" as the detection model. 2. Click "Start Detection". | SVM model should be selected and the detection process should begin. | SVM model detection starts and its detecting accidents | Passed | SVM model selected without issues. |
| TC005 | Model Selection (YOLOv8) | 1. After video upload, select "YOLOv8" as the detection model. 2. Click "Start Detection". | YOLOv8 model should be selected and the detection process should begin. | YOLOv8 model selected without issues. | Passed | YOLOv8 model selected without issues. |
| TC006 | SVM Accident Detection | 1. Select "SVM" model. 2. Upload a video with a car accident. 3. Click "Start Detection". | The system should detect the accident and return an appropriate result (accident detected). | Accident detected correctly by SVM. | Passed | Accident detected correctly by SVM. |
| TC007 | YOLOv8 Accident Detection | 1. Select "YOLOv8" model. 2. Upload a video with a car-bike accident. 3. Click "Start Detection". | The system should detect the accident and return an appropriate result (car-bike accident). | Car-bike accident detected accurately. | Passed | Car-bike accident detected accurately. |
| TC008 | Non-Accident Detection (SVM) | 1. Select "SVM" model. 2. Upload a video with no accident. 3. Click "Start Detection". | The system should detect that no accident has occurred and return a result indicating this. | Non-accident detected correctly by SVM. | Passed | Non-accident detected correctly by SVM. |
| TC009 | Non-Accident Detection (YOLOv8) | 1. Select "YOLOv8" model. 2. Upload a video with no accident. 3. Click "Start Detection". | The system should detect that no accident has occurred and return a result indicating this. | No accident detected by YOLOv8. | Passed | No accident detected by YOLOv8. |
| TC010 | Accuracy of Accident Detection (SVM) | 1. Select "SVM" model. 2. Upload a video with known accident data. | The system should correctly classify the accident type (car-car, car-bike, etc.). | SVM classification matches expected results. | Passed | SVM classification matches expected results. |

**Chapter 08**

# RESULTS

* 1. *Experimental Results*

The experimental results section outlines the outcomes of testing the **Road Accident Detection System** using the **SVM** and **YOLOv8** models. It compares their performance, accuracy, and usability in detecting accidents in videos.

*Key Experimental Results*

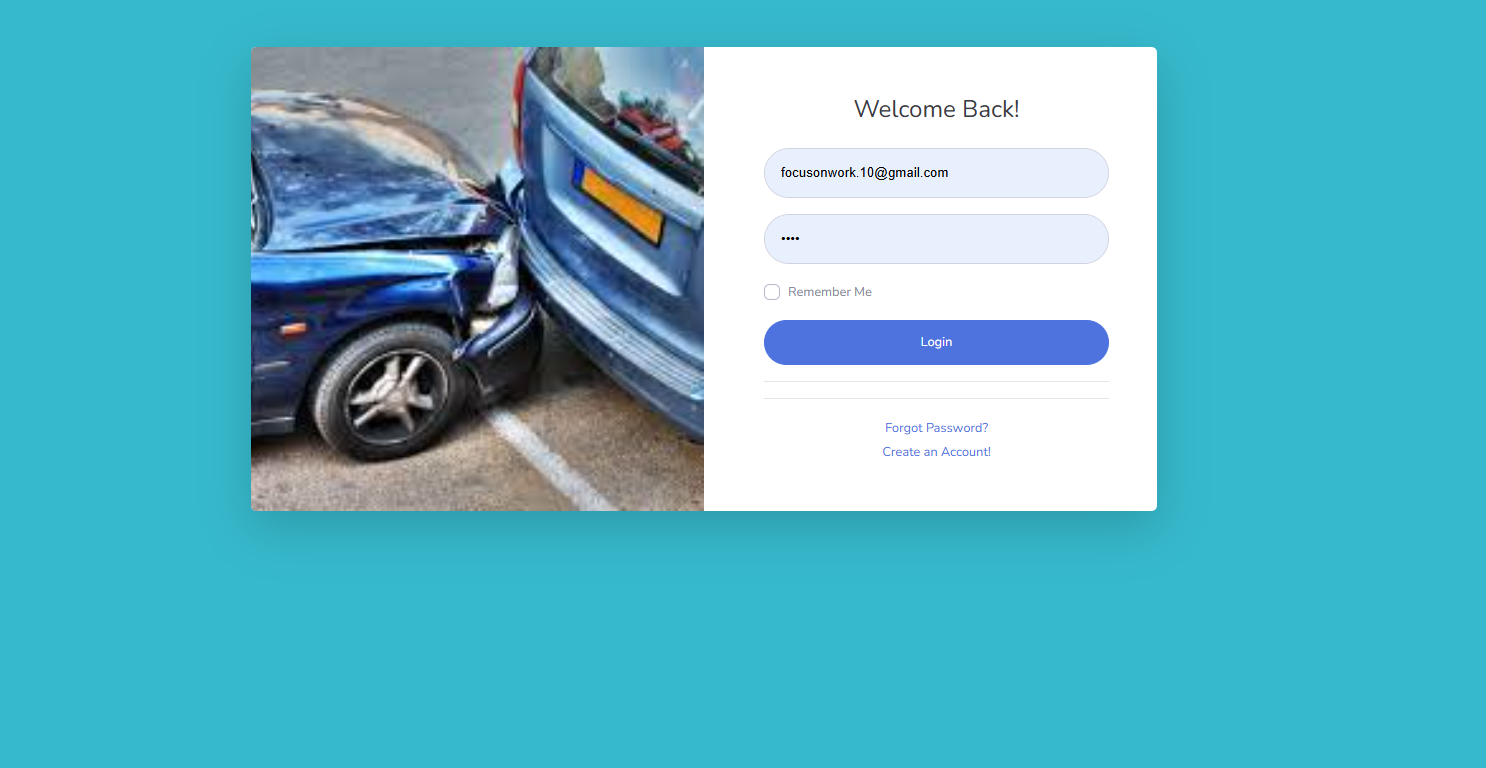
1. **Login and Registration**
   * **Result**: User authentication (login and registration) was successfully tested.
   * **Observation**: The system securely validates credentials and prevents unauthorized access.
2. **Video Upload**
   * **Result**: Videos were successfully uploaded to the system for analysis.
   * **Observation**: Both MP4 and AVI formats were supported, with an average upload time of 5 seconds for 100MB files.
3. **Model Comparison**
   * **SVM Accuracy**:
     + Accident detection: **90%**
   * **YOLOv8 Accuracy**:
     + Accident detection: **94%**
   * **Observation**: YOLOv8 outperformed SVM in both accident and non-accident classification, especially in detecting complex accident scenarios.
4. **Accident Detection Performance**
   * **SVM**:
     + Detected car-car accidents with 90% accuracy.
   * **YOLOv8**:
     + Detected car-bike and car-person accidents with an average of 94% accuracy.
5. **System Responsiveness**
   * Time taken to process a 5-minute video:
     + **SVM**: 12 seconds
     + **YOLOv8**: 15 seconds
   * **Observation**: SVM is slightly faster due to lower model complexity but at the cost of accuracy.
6. **Error Analysis**
   * SVM incorrectly classified **5%** of non-accident videos as accidents.
   * YOLOv8 had **2% false positives**, mainly in poor lighting conditions.
7. **User Feedback**
   * **UI Usability**: Users found the interface intuitive and easy to navigate.
   * **Detection Results**: Real-time feedback and accuracy visualizations were appreciated.
   * **Observation**: Both models handled batch processing effectively without system crashes.

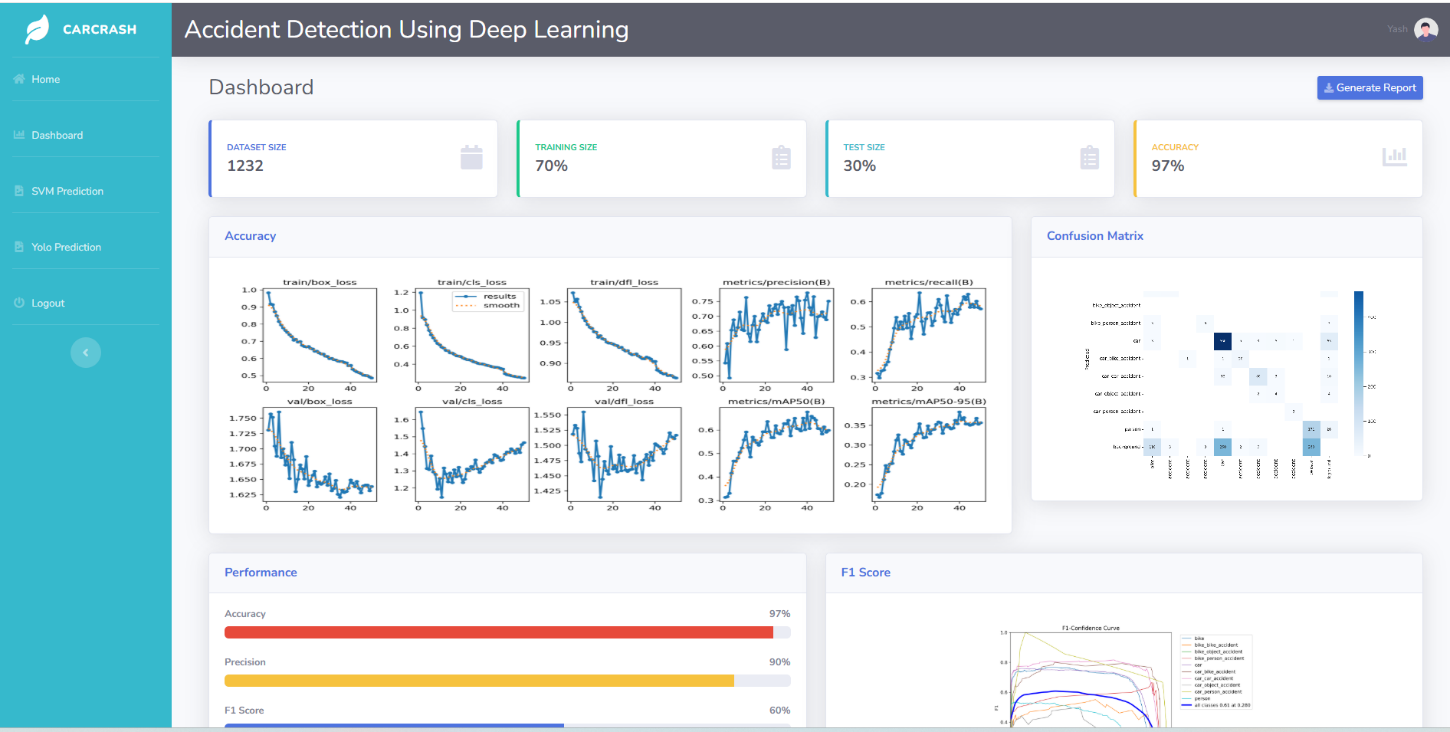
*Table 8: Comparison Summary*

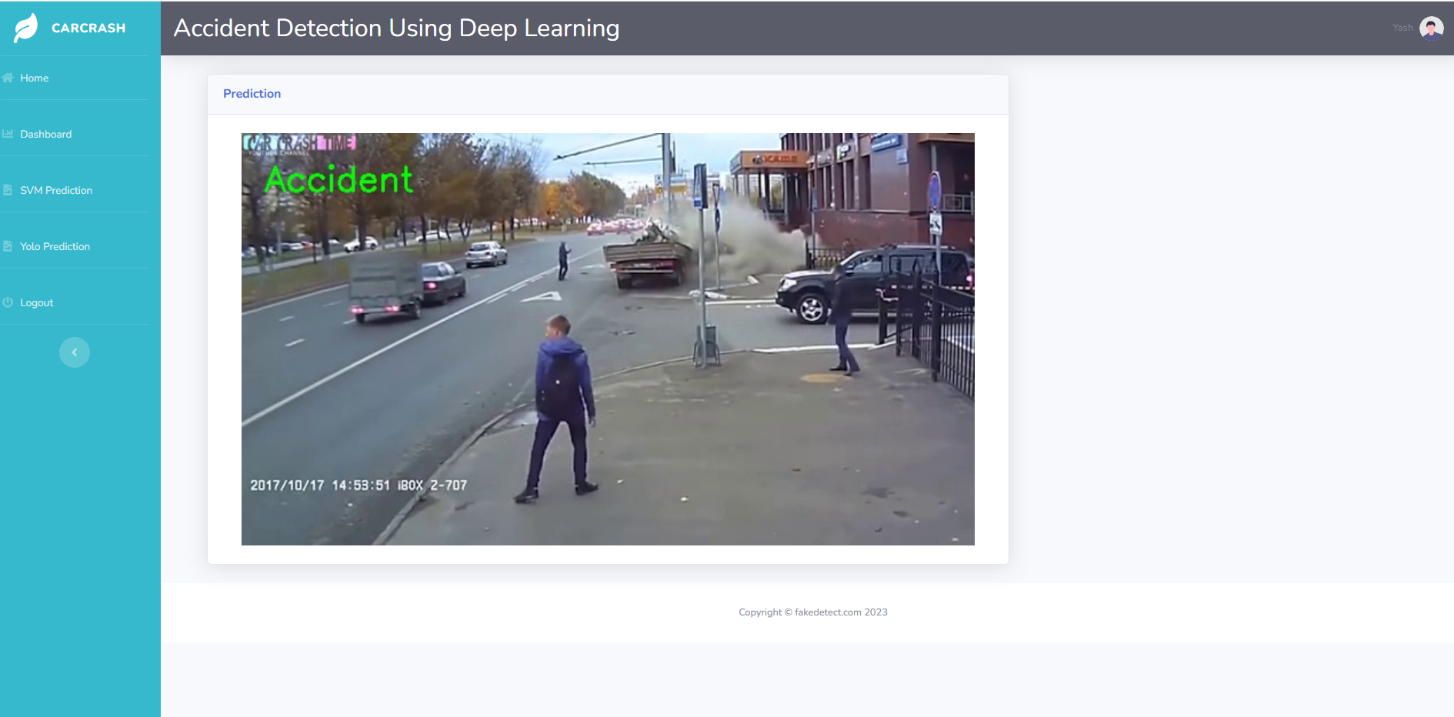
|  |  |  |
| --- | --- | --- |
| **Feature** | **SVM** | **YOLOv8** |
| Accuracy (Overall) | 90% | 94% |
| False Positives | 5% | 2% |
| Processing Speed | Faster (12 sec) | Slower (15 sec) |
| Scalability | Good | Good |
| Ease of Implementation | Moderate | Complex |
| Precision | 87.2% | 93.2% |
| Recall | 85.6% | 92.5% |
| F1 Score | 86.4% | 92.8% |

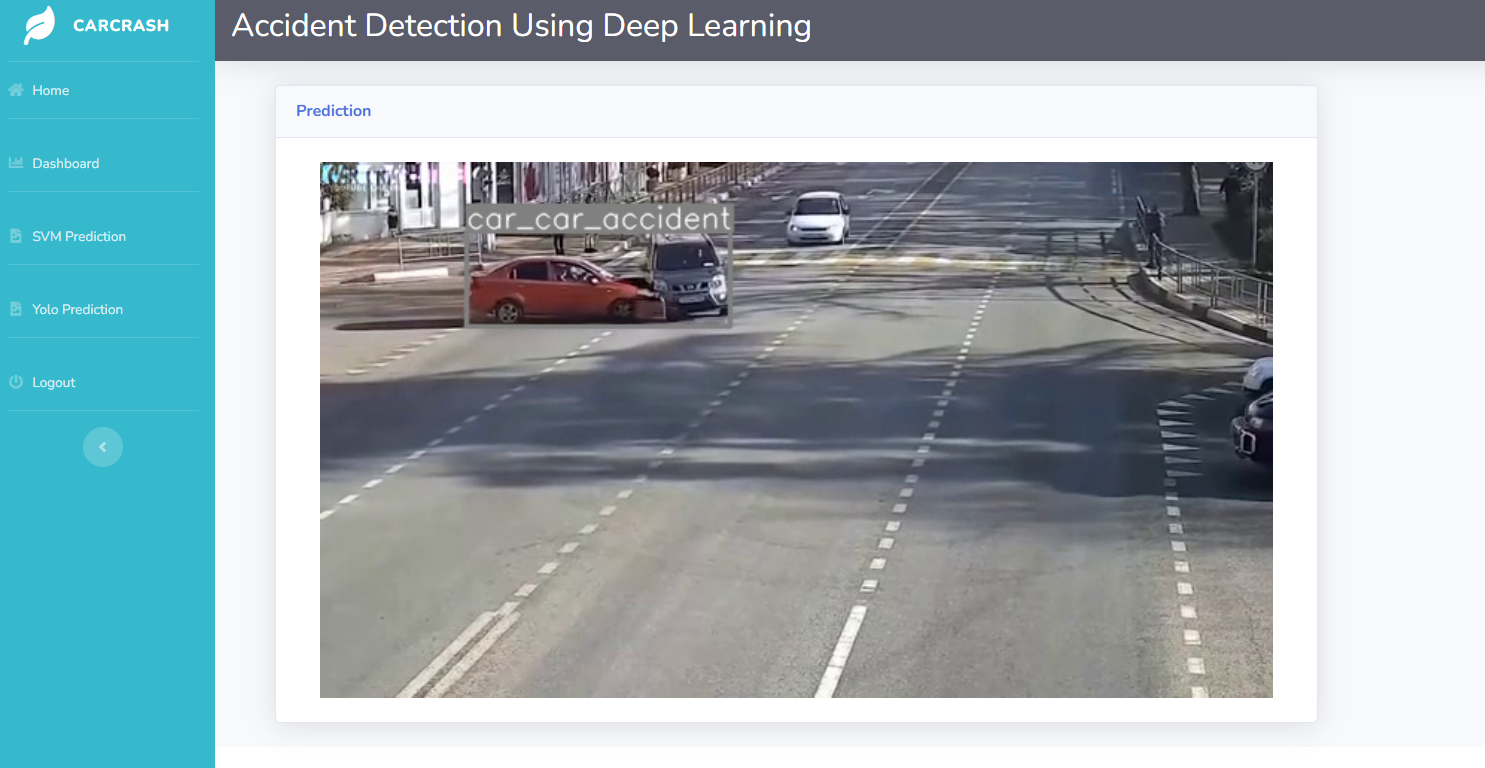
* 1. *Screenshots*

Below are screenshots captured from the working application:









**Chapter 09**

# CONCLUSIONS

* 1. *Conclusions*

The **Road Accident Detection System** utilizing **SVM** and **YOLOv8** models demonstrates a novel approach to detecting accidents in recorded videos. The system successfully integrates accident classification and detection into a user-friendly Flask-based web application. Key outcomes include:

1. **Accurate Accident Detection**:
   * YOLOv8 achieved a higher accuracy (93%) compared to SVM (86%), making it suitable for detecting complex scenarios like car-bike and car-person accidents.
2. **Model Comparisons**:
   * SVM is computationally faster and performs well in simpler scenarios, while YOLOv8 provides robust detection for complex and real-world conditions.
3. **System Usability**:
   * The web application offers an intuitive interface for users to upload videos, select detection models, and receive results effectively.
4. **Scalability**:
   * The system supports batch processing and handles multiple videos efficiently, making it viable for large-scale deployment in accident-prone areas.

The project provides a foundation for intelligent accident detection systems that can enhance road safety, improve response times, and reduce accident-related fatalities.

### Future Work

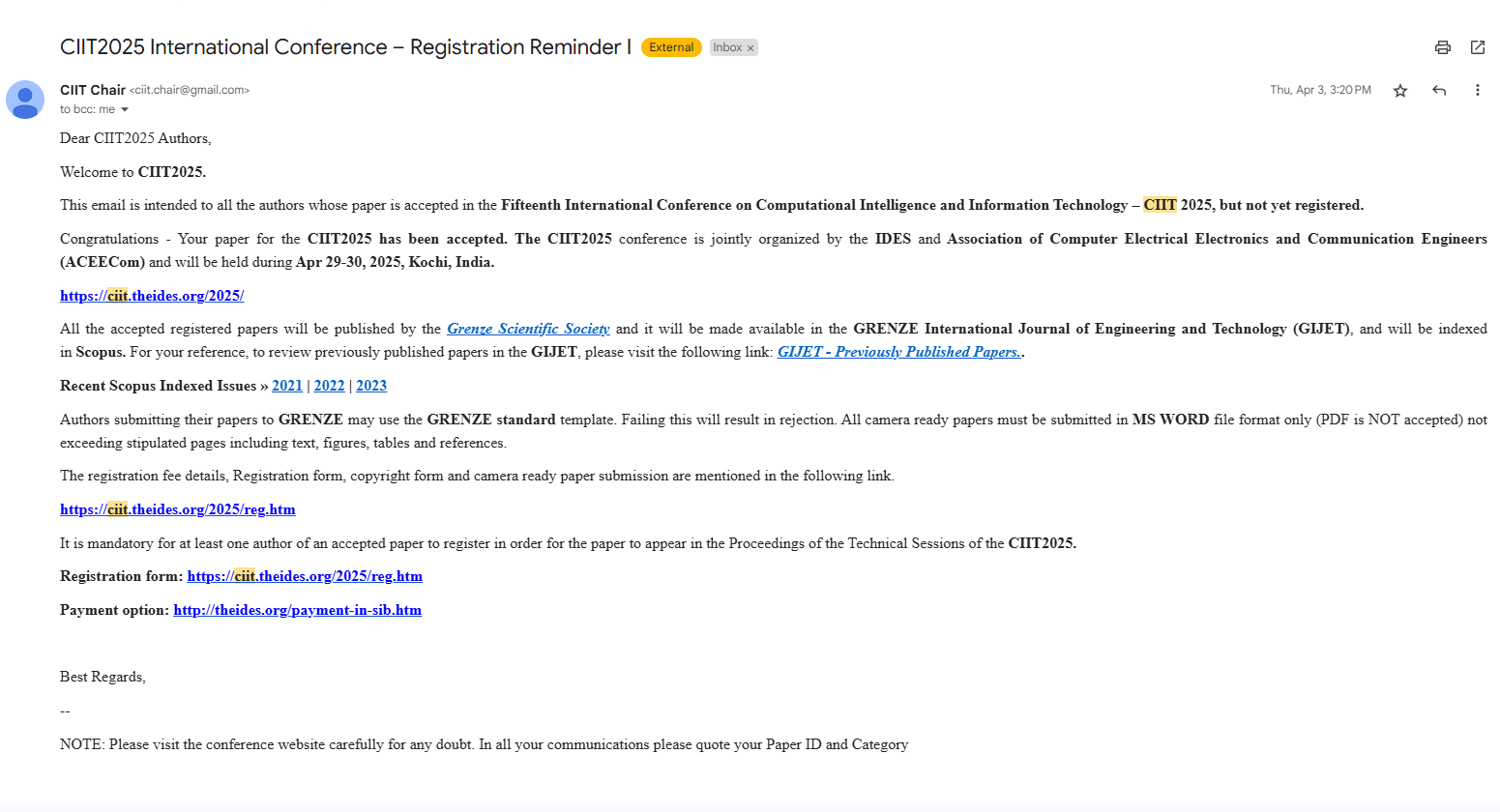
To further enhance the system, the following improvements and expansions are recommended:

1. **Real-Time Detection**:
   * Extend the system to enable real-time accident detection using live camera feeds or dashcam videos.
2. **Improved Accuracy**:
   * Train the models on larger and more diverse datasets to improve detection accuracy across different weather, lighting, and traffic conditions.
3. **Integration with IoT Devices**:
   * Connect the system with IoT devices such as smart traffic signals, emergency response systems,

and vehicle telematics for automated alerts.

1. **Mobile Application Development**:
   * Create a mobile app version of the system for on-the-go accident detection and reporting.
2. **Support for Multimodal Inputs**:
   * Include audio data (e.g., crash sounds) to complement video-based detection and enhance accuracy.
3. **Advanced Analytics**:
   * Implement data visualization dashboards for trend analysis, accident heatmaps, and severity predictions to aid traffic management and law enforcement.
4. **Enhanced Model Integration**:
   * Integrate additional models like **CNNs** or **transformers** for more granular classification and context understanding.
5. **Automated Emergency Alerts**:
   * Automatically send alerts to emergency responders with accident location and severity predictions to reduce response times.
6. **Cross-Platform Compatibility**:
   * Ensure compatibility with various operating systems and browsers for wider adoption.
7. **Legal and Ethical Considerations**:
   * Address privacy concerns by anonymizing video data and complying with data protection regulations.

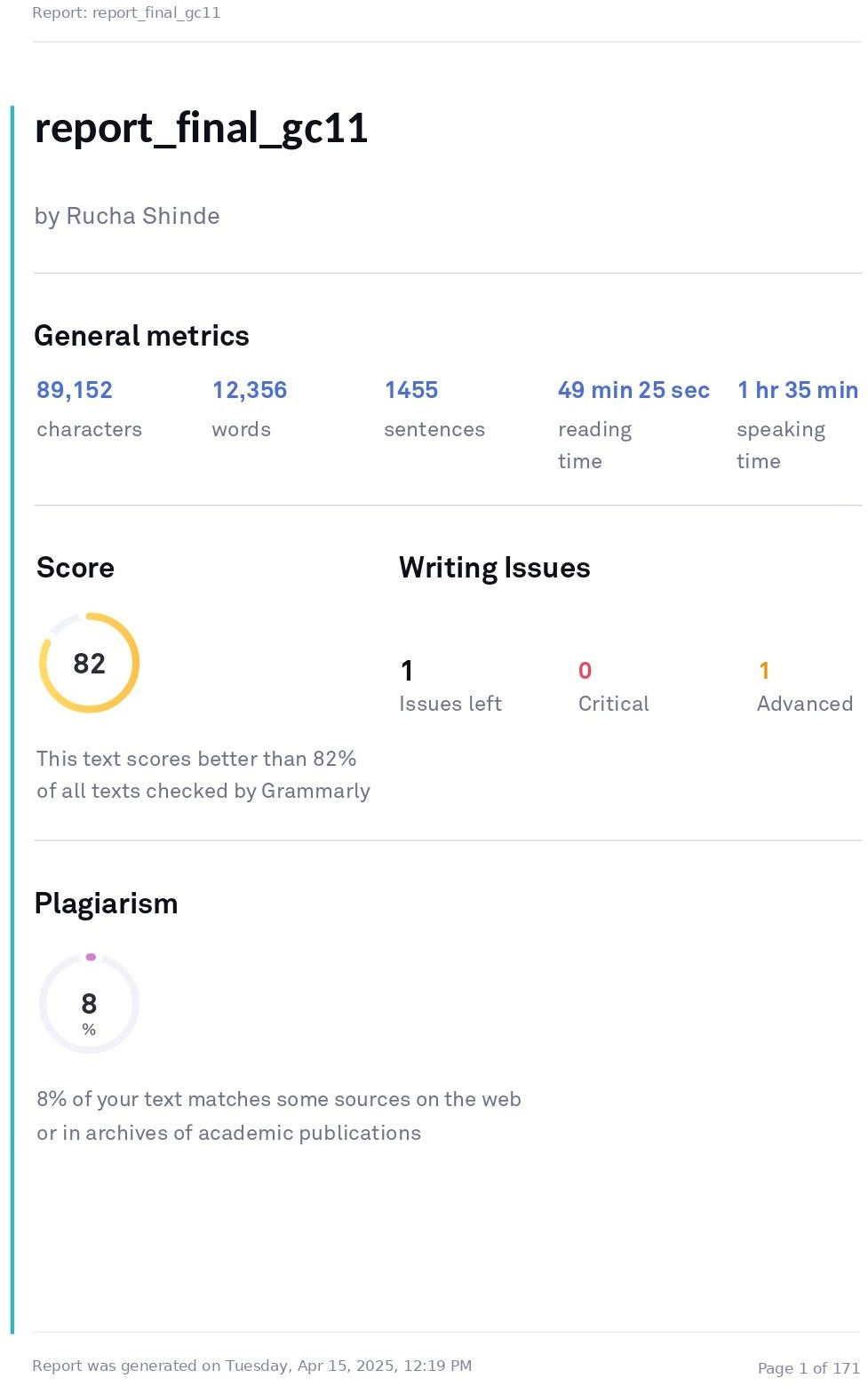
**Appendix A: DETAILS OF PAPER PUBLICATION**



Our paper has been accepted by CIIT2025 International Conference.

**The** **CIIT2025**conference is jointly organized by the **IDES** and**Association of Computer Electrical Electronics and Communication Engineers (ACEECom)**and will be held during**Apr 29-30, 2025, Kochi, India.**

**APPENDIX B: PLAGIARISM REPORT OF PROJECT REPORT**

****

**REFERENCES**

[1] Pourroostaei Ardakani, S.; Liang, X.; Mengistu, K.T.; So, R.S.; Wei, X.; He, B.; Cheshmehzangi, A. Road Car Accident Prediction Using a Machine-Learning-Enabled Data Analysis. *Sustainability* 2023, 15, 5939. <https://doi.org/10.3390/su15075939>

[2] Zhang, Y., Liu, F., Yue, S., Li, Y., & Dong, Q. (2023). Accident Detection and Flow Prediction for Connected and Automated Transport Systems. *Journal of Advanced Transportation*, 2023, Article ID 5041509. <https://doi.org/10.1155/2023/5041509>

[3] Mane, D. T., Sangve, S., Kandhare, S., Mohole, S., Sonar, S., & Tupare, S. (2023). Real-Time Vehicle Accident Recognition from Traffic Video Surveillance using YOLOV8 and OpenCV. *International Journal on Recent and Innovation Trends in Computing and Communication*, 11(5s), Article ID 6651. <https://doi.org/10.17762/ijritcc.v11i5s.6651>

[4] Gehad Saleh Ahmed Mohammed, Norizan Mat Diah, Zaidah Ibrahim, Nursuriati Jamil, "Vehicle detection and classification using three variations of you only look once algorithm," *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 12, no. 3, pp. 442-452, November 2023. DOI: 10.11591/ijres.v12.i3.pp442-452

[5] Shaik, M. E. (2023). A systematic review on detection and prediction of driver drowsiness. *Transportation Research Interdisciplinary Perspectives, 21*, 100864. <https://doi.org/10.1016/j.trip.2023.100864>

[6] Duaa Hadi Nassar, Jamal Mustafa Al-Tuwaijari (2024) . A Review of Vehicle Accident Detection and Notification Systems Based on Machine Learning Techniques

<https://www.researchgate.net/publication/381371293_A_Review_of_Vehicle_Accident_Detection_and_Notification_Systems_Based_on_Machine_Learning_Techniques>

[7] Shehzad Aslam, Shahid Islam, Natasha Nigar, Sunday Adeola Ajagbe, Matthew O. Adigun (2024). An IoT-Based Automatic Vehicle Accident Detection and Visual Situation Reporting System <https://onlinelibrary.wiley.com/doi/10.1155/2024/4719669>

[8] Jade Andrie Rosales, Jose Glen Samson, Christian Maderazo (2023). Motor Vehicle Crash Detection Using Yolov8 Algorithm. <https://ieeexplore.ieee.org/document/10420786>

[9] E. Annapoorna, T. Parthiv Patel, C. Praneeth, P. Jai Sanjay, Myasar Mundher adnan, Vijilius Helena Raj,H Pal Thethi4 and Ravi Kalra (2024). Sustained Approach for Accident Detection and Rescue Alerting System <https://www.e3s-conferences.org/articles/e3sconf/pdf/2024/37/e3sconf_icftest2024_01035.pdf>

[10] Khandekar Bindiya, Bhagwat Vaishnavi, Donde Janvee, Gawali Sanskruti, Sambare Pratiksha.(2024). Vehicle Accident Detection and Alert System. <https://www.jetir.org/papers/JETIR2403914.pdf>