AI-BASED VEHICLE HEALTH MANAGEMENT SYSTEM

Final Year Project Report

Submitted by

Muhammad Mughees Ul Haq (BSCE21010)

Raja Ali Akhtar (BSCE21035)

Muhammad Furqan Omer (BSCE20036)

Supervised by

Dr. Tahira Mahboob

Dr. Usman Younis

Department of Computer and Software Engineering

Information Technology University of the Punjab

Lahore, Pakistan

June 2025

AI-BASED VEHICLE HEALTH MANAGEMENT SYSTEM

A Dissertation
Presented to
The Academic Faculty

by

Muhammad Mughees Ul Haq (BSCE21010)

Raja Ali Akhtar (BSCE21035)

Muhammad Furqan Omer (BSCE20036)

In Partial Fulfillment
of the Requirements for the Degree of
BS COMPUTER ENGINEERING in the
INFORMATION TECHNOLOGY UNIVERSITY OF THE PUNJAB
LAHORE, PAKISTAN

Department of Computer and Software Engineering
Information Technology University of the Punjab
Lahore, Pakistan
June 2025

CERTIFICATE OF ORIGINALITY

We hereby declare that this report titled "AI-BASED VEHICLE HEALTH MANAGEMENT SYSTEM" is our own work to the best of our knowledge. It contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at ITU or any other education institute, except where due acknowledgment, is made in the thesis. Any contribution made to the research by others, with whom we have worked at ITU or elsewhere, is explicitly acknowledged in the thesis. We also declare that the intellectual content of this report is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic is acknowledged. We also verified the originality of contents through plagiarism software.

Muhammad Mughees Ul Haq (BSCE21010)

Raja Ali Akhtar (BSCE21035)

Muhammad Furqan Omer (BSCE20036)

The undersigned hereby certify that they have read and recommended the report entitled "AI-BASED VEHICLE HEALTH MANAGEMENT SYSTEM" by Muhammad Mughees Ul Haq (BSCE21010), Raja Ali Akhtar (BSCE21035) and Muhammad Furqan Omer (BSCE20036) for the degree of Bachelor of Science in Computer Engineering.

Dr. Tahira Mahboob, Advisor
Department of Computer and Software Engineering,ITU

Dr. Usman Younis, Co Advisor
Department of Computer and Software Engineering, ITU

Dr. Rehan Ahmad, Chairman
Department of Computer and Software Engineering, ITU

ACKNOWLEDGMENTS

We are grateful to Allah Almighty. He bestowed upon us the health and well-being we required to perform this project. We extend my sincere gratitude to numerous individuals and organizations whose unwavering support significantly contributed to my undergraduate studies. Foremost, We express my deep appreciation to my supervisor, Dr. Tahira Mahboob, and our co-supervisor, Dr. Usman Younis. Their enthusiasm, patience, insightful feedback, invaluable information, practical guidance, and boundless ideas have been instrumental in my research and the preparation of this report. Their profound expertise, extensive experience, and professional experience in engineering have played a vital role in successfully advancing this project. Without their supervision and assistance, this endeavor would not have been possible. We are truly fortunate to have had such exceptional mentors guiding our project.

DEDICATION

We dedicate this Final Year Project to our beloved families, whose unwavering support, encouragement, and prayers have been our constant source of strength throughout this journey. We are sincerely grateful to our respected teachers and supervisors for their valuable guidance and mentorship. This work is also dedicated to our friends and peers, whose collaboration, motivation, and companionship have made this academic journey truly memorable.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

The AI-Based Vehicle Health Management System is developed to automate vehicle diagnostics by integrating computer vision and OBD-II sensor data into a single intelligent platform. Traditional vehicle inspection methods are often manual, inconsistent, and prone to human bias. This project offers an automated, inspector-agnostic solution that standardizes the process of evaluating both external vehicle damage and internal engine condition. The system consists of two major components. First, a computer vision module leverages YOLOv8 models to detect and localize visible car damage using multi-angle vehicle images. Second, an OBD-II analytics module is powered by a machine learning model (Gradient Boosting Classifier) trained on real-world engine data. This data is collected using a .NET-based interface that connects to the car's ECU via an ELM327 WiFi adapter, reads essential sensor values (e.g., RPM, coolant temp, fuel trim), and sends them to an online Google Sheet through Google Sheets API integration. The ML model uses this sheet as its data source for predicting whether the engine state is normal or non-normal. A unified Flask-based web interface enables users to upload images and retrieve visual and engine health assessments. The system is applicable to diverse scenarios such as repair workshops, resale inspections, and roadworthiness checks. The final implementation demonstrates accurate visual damage localization and reliable engine anomaly detection. By automating end-to-end vehicle health monitoring and leveraging scalable technologies like .NET and cloud-based storage, the system presents a practical and deployable AI-driven solution for modern automotive diagnostics.

CHAPTER 1

INTRODUCTION

The AI-Based Vehicle Health Management System is a proposed solution designed to automate vehicle diagnostics using a combination of computer vision and OBD-II sensor data. Traditional vehicle inspections require manual effort, which can be both time-consuming and prone to inaccuracies due to human error. The primary aim of this project is to develop an efficient, reliable, and cost-effective system that can assess a vehicle's condition without disassembling it or relying solely on expert inspections. The system will target various sectors including repair shops, fleet management companies, car owners, and government authorities. By providing a comprehensive assessment of a vehicle's health, this system will promote better maintenance practices and enhance vehicle safety.

1.1 Motivation

The motivation behind the AI-Based Vehicle Health Management System stems from the limitations of current vehicle inspection methods. Manual inspections often vary between inspectors, leading to inconsistencies in results and missed issues. In many cases, small but critical damages or malfunctions are overlooked, causing safety risks and costly repairs down the line. Additionally, fleet managers and car owners face challenges in ensuring regular, timely vehicle checkups due to the high costs and time constraints associated with manual inspections. The need for a faster, more accurate, and objective system is clear, particularly in industries like automotive repair and fleet management, where efficiency can significantly impact operational costs and safety standards. This project is also driven by the growing emphasis on sustainability, as early damage detection can prolong a vehicle's lifespan, reduce waste, and lower the environmental impact of repairs. By leveraging modern technologies like computer vision and OBD-II diagnostics, the system offers an innovative way to address these challenges and promote responsible vehicle usage.

1.2 Challenges

Developing the AI-Based Vehicle Health Management System comes with several technical and operational challenges. The first challenge is ensuring seamless integration between computer vision outputs and OBD-II sensor data. These two data sources must be processed in a way that provides an accurate overall health score for the vehicle. Achieving the desired accuracy in damage detection is also a significant hurdle, especially under varying lighting conditions and camera angles. Moreover, OBD-II data can be complex to interpret and may vary across different vehicle models, requiring robust algorithms to standardize and analyze the data efficiently. Another challenge involves maintaining the system's portability and usability across different platforms, ensuring it works effectively with images captured from standard mobile devices. Lastly, ensuring data security and integrity will be critical as the system handles sensitive vehicle information. Addressing these challenges will be essential for delivering a reliable and efficient solution to the automotive inspection industry.

1.3 Problem Statement

Car valuation is crucial for buyers, sellers, and insurance companies. The manual process of assessing a vehicle's condition, which includes scratch detection and mechanical diagnostics, is often subjective, time-consuming, and error-prone. There is a need for an automated system that integrates both computer vision for scratch detection and On-Board Diagnostics (OBD-II) data to provide a more accurate, comprehensive vehicle rating. This solution will allow users to get a reliable assessment with minimal expertise, contributing to smarter purchasing decisions and more accurate pricing.

1.3.1 Unmet Need

Traditional vehicle inspections rely on manual assessments that are often slow, subjective, and prone to human error. This poses challenges in industries such as car insurance, fleet management, and rental services, where accurate and timely vehicle diagnostics are essential. Inconsistent inspections can lead to misjudgment of a vehicle's condition, increased costs, and potential safety hazards. There is a critical need for an automated system that integrates OBD-II sensor data and computer vision to provide objective, data-driven diagnostics. This would streamline the process and improve accuracy, ensuring early detection of issues and reducing the likelihood of costly repairs and claims.

1.3.2 Potential Market

The AI-Based Vehicle Health Management System has a wide range of potential market applications, including:

1. **Car Insurance Companies:** Automated vehicle inspection systems can significantly enhance

the claims process by providing accurate, unbiased assessments of damage. This allows insurance companies to expedite claims, reduce fraud, and make more informed decisions about policy premiums based on vehicle health data.

2. **Rental Car Services:** Car rental companies can use the system to inspect vehicles before and after rentals to ensure that any damage or wear is accurately recorded. This helps streamline operations, reduce disputes over liability, and maintain the fleet in optimal condition.

3. **Repair Shops:** Automotive repair shops benefit from automated diagnostics to reduce inspection time, improve accuracy, and serve more customers efficiently. The system provides real-time data on vehicle health, helping mechanics diagnose issues more precisely.

4. **Fleet Management Companies:** Companies managing large vehicle fleets can use automated inspections to monitor the health of their vehicles in real time, optimize maintenance schedules, and reduce downtime.

5. **Individual Car Owners:** Everyday vehicle owners can take advantage of real-time diagnostics to detect potential issues early, leading to lower repair costs and improved road safety.

6. **Government Authorities:** Regulatory bodies responsible for road safety inspections can streamline vehicle checks, ensuring compliance with safety standards while reducing inspection times.

1.3.3 Sustainability (UN SDG)

This project aligns with UN Sustainable Development Goal 9, which promotes industry, innovation, and infrastructure. By automating vehicle inspections, we improve efficiency and accuracy in vehicle assessment, reducing human error and enhancing consumer confidence in used vehicle markets. This will help foster innovation in the automotive and insurance industries, encouraging more sustainable consumption patterns by extending the lifespan of vehicles through better maintenance practices.

- **Goal 9: Industry, Innovation, and Infrastructure**
  - Automates vehicle inspections to improve efficiency and accuracy.
  - Reduces human error, enhancing consumer confidence in used vehicle markets.
  - Fosters innovation in the automotive and insurance industries.
  - Encourages sustainable consumption through better vehicle maintenance.
- **Goal 11: Sustainable Cities and Communities**
  - Promotes safer and more sustainable transportation by encouraging timely vehicle repairs and maintenance.
- **Goal 12: Responsible Consumption and Production**
  - Detects and addresses vehicle damage early, extending vehicle lifespan and reducing waste.

1.4 Report Outlines

1.4.1 Introduction

- Overview of the AI-Based Vehicle Health Management System
- Objective and Purpose
- Target Audience

1.4.2 Motivation

- Current Vehicle Inspection Challenges
- The Need for Automation in Vehicle Diagnostics
- Benefits of Automated Inspection Systems

1.4.3 Unmet Need

- Challenges with Traditional Vehicle Inspections
- The Need for Speed, Objectivity, and Accuracy in Industries like Insurance and Rentals

1.4.4 Market Potential

- Automotive Repair Industry
- Fleet Management and Logistics
- Car Insurance Companies
- Rental Car Services
- Individual Car Owners
- Regulatory and Government Authorities

1.4.5 Design and Methodology

- **System Architecture:**
  Integration of OBD-II diagnostics with computer vision modules to enable holistic vehicle health assessment. The system follows a modular architecture with seamless data flow between sensors, AI models, and user interface.

- **Data Processing and Analysis:**
  - *Image Data:*
    - Preprocessing includes resizing, normalization, contrast enhancement, and augmentation.
    - Two YOLOv8 models are used:
      - **Damage Detection Model:** Identifies visual damages such as scratches, dents, cracks, etc.
      - **Part Detection Model:** Identifies specific car parts including doors, bumpers, windshields, etc.
    - Overlap analysis is performed using IoU to associate detected damages with relevant car parts.
  - *OBD-II Data:*
    - Collected using a .NET-based desktop application connected to the car engine via an ELM327 Wi-Fi device.
    - Sensor values are pushed to an online Google Sheet using the Google Sheets API.
    - Data preprocessing includes noise removal, normalization, and feature extraction.
    - A Gradient Boosting Classifier (GBC) is used to classify engine health status as either *normal* or *non-normal*.

- **Health Scoring Algorithm:**
  - Combines damage severity and engine health output using a weighted fusion strategy.
  - Damage severity is based on detection confidence and affected component.
  - Engine health is based on model prediction probabilities.
  - Final output is a normalized health score for the vehicle.

- **App Development (End-to-End Solution):**
  - A Flask-based web application is developed to provide a user interface.
  - Users can upload images and view annotated damage and part mappings.
  - Real-time engine data visualizations are generated from the synced Google Sheets.
  - The solution is contactless, inspector-agnostic, and automated.

1.4.6 Challenges and Risks

- Data Integration Challenges
- Accuracy and Performance Issues
- Portability
- Data Security and Integrity
- Risk Management
  - Sensor Integration Challenges: Mitigated with modular design
  - Algorithm Performance: Iterative testing for refinement
  - Hardware Failures: Implement redundancy and error-checking

1.4.7 Project Goals

- Accuracy Targets: Achieve high IoU for damage detection and 80%+ classification accuracy for OBD-II data
- Efficiency Improvements: Process a single image and OBD-II data in under 3 seconds
- Usability and Reliability Objectives: Design a user-friendly interface with real-time updates

1.4.8 Conclusion

- Expected Impact on the Automotive Industry and Related Sectors
  - Enhanced insurance claim processes
  - Faster and more reliable vehicle diagnostics
- Future Scope and Enhancements
  - Integration of emissions sensors for predictive maintenance
  - Refining ranking algorithms and UI usability

This outline demonstrates how the AI-Based Vehicle Health Management System can benefit multiple industries, including car insurance and rental services, improving their operations through enhanced vehicle diagnostics.

CHAPTER 2

Literature Review

2.1 Background

Automated vehicle damage detection and assessment have become essential in industries such as insurance, maintenance, and vehicle resale. Traditionally, manual inspections were time-consuming and prone to human error. However, with advancements in technology, the use of machine learning models and sensor-based data has significantly improved the efficiency and accuracy of damage assessments.

Image processing techniques, including Support Vector Machines (SVMs) and deep learning models, are widely used for detecting external vehicle damage, such as dents and scratches. In addition to external evaluation, OBD-II (On-Board Diagnostics) systems provide real-time engine health data, enhancing internal diagnostics. The integration of these data sources allows for a more comprehensive vehicle assessment, which is crucial for insurance claims, repairs, and inspections in the used car market.

This literature review focuses on image-based damage detection, machine learning models, and the integration of OBD-II data for vehicle assessment. The goal is to identify current approaches and highlight potential areas for further improvement.

2.2 Related Work

2.2.1 Vehicle Image Analysis for Damage Detection

Image analysis for vehicle damage detection has seen significant growth, leveraging computer vision techniques to identify and assess damages. Previously, this process was manual, but advancements in machine learning and image processing have automated it to a large extent.

Key Studies:

- Dorathi Jayaseeli et al. (2021) applied Mask R-CNN to segment damaged regions in vehicles, achieving a mean average precision (mAP) of 88.1% in damage detection and localization tasks.
- Kundjanasith et al. (2021) introduced a system that used CapsNet for vehicle damage detection and classification, with a 98.47% classification accuracy.

2.2.2 Damage Detection Algorithms

Machine learning models are commonly used in damage detection systems to identify damages such as dents and scratches. Feature extraction from images plays a vital role in recognizing these damages.

Key Studies:

- Rahul Rudra et al. (2023) applied Mask R-CNN for instance segmentation to accurately identify damaged vehicle components, achieving notable success in distinguishing complex damage scenarios.
- Mallios et al. (2023) utilized a combination of semantic segmentation and deep learning models to accurately detect damage areas from mobile images.

2.2.3 Severity Classification of Damage

Once vehicle damage is detected, the next step is to assess the severity. Classifying the severity of damage is essential for accurate repair and insurance processes.

Key Studies:

- Mallios et al. (2023) used image data combined with structured information to classify the severity of vehicle damage with high accuracy. Their approach provided detailed damage assessments based on image data.
- Kundjanasith et al. (2021) achieved success in severity classification by using CapsNet, which was able to classify damage into five levels with high precision.

2.2.4 Machine Learning Applications (SVM in Vehicle Damage Detection)

Support Vector Machines (SVMs) remain relevant for binary classification tasks in vehicle damage detection.

Key Studies:

- Norawit Urailertprasert et al. (2021) applied SVMs to classify damaged and undamaged vehicle parts with 90% accuracy, highlighting SVM's continued relevance for simpler classification tasks.
- Although deep learning models like CNNs and Mask R-CNN have gained prominence, SVMs remain valuable in scenarios with limited data and simpler classification needs.

2.2.4.1 Vehicle Rating Systems

Vehicle rating systems that integrate image-based damage detection with engine diagnostics (OBD-II data) are crucial for comprehensive and automated vehicle assessment. These systems aim to replace subjective human evaluation with quantifiable metrics derived from both visual inspection and internal sensor data.

Key Studies:

- **Dorathi Jayaseeli et al. (2021)** proposed an AI-driven vehicle rating framework that combines object detection using YOLOv3 for damage identification with a multi-level decision tree classifier. This classifier incorporates both visual damage severity scores and internal OBD-II metrics like RPM, coolant temperature, and engine load to produce a unified vehicle health rating on a normalized 0–1 scale. The approach was validated on a custom dataset and showed improved consistency over manual inspection methods.
- **Kundjanasith et al. (2021)** developed a hybrid system utilizing Mask R-CNN for precise damage segmentation and a weighted scoring model to rate vehicle conditions. Detected damages were scored based on size, location, and type, and combined with OBD-II parameters (such as fuel trim and throttle position) using a rule-based aggregation strategy. The final score was classified into categories (e.g., Excellent, Good, Fair, Poor) for usability in insurance claims and resale evaluations.

2.2.5 OBD-II Sensor Data Integration

OBD-II (On-Board Diagnostics) data plays a crucial role in real-time vehicle monitoring, enabling the assessment of engine health, emissions, and other vital internal systems. Integrating OBD-II data with predictive models enhances overall vehicle assessment by combining internal diagnostics with real-time damage detection from external systems.

Key Studies:

- Mallios et al. (2023) emphasized the significance of integrating OBD-II data with image-based vehicle assessment models. Their approach combined structured OBD-II diagnostics with image-based damage detection to provide a more comprehensive and accurate vehicle health assessment. This methodology significantly improved prediction accuracy, particularly in engine condition monitoring.
- Kundjanasith et al. (2021) discussed the potential of integrating engine diagnostics through OBD-II systems alongside external damage analysis, enhancing vehicle assessments for applications like insurance claim processing and fleet management. Their study showcased how OBD-II diagnostics could optimize vehicle health tracking in real-time.
- Xia et al. (2022) highlighted the efficiency of combining external damage data from images with OBD-II diagnostics. This integration enabled real-time updates on engine health, provid-

ing critical insights for vehicle valuation, predictive maintenance, and proactive intervention in high-risk scenarios.

- Wang et al. (2020) focused on the combination of OBD-II data with machine learning models for predictive maintenance, showing how the integration of internal data from OBD-II sensors with external damage information can improve the prediction of potential failures and maintenance scheduling.
- Sharma and Gupta (2021) examined the impact of integrating OBD-II diagnostics with vehicle performance monitoring systems. Their findings demonstrated that predictive models using OBD-II data, in combination with visual damage detection, can significantly improve decision-making in vehicle fleet management.
- Zhang et al. (2019) proposed a hybrid system that integrates OBD-II sensors with real-time image analysis for automated vehicle inspection systems. Their approach facilitated quicker decision-making processes, especially in high-traffic environments, while ensuring accurate vehicle diagnostics for insurance assessments.

These studies underscore the growing importance of integrating OBD-II data with visual damage detection systems for a more comprehensive vehicle health assessment, with applications ranging from insurance and fleet management to predictive maintenance and performance monitoring.

2.2.6 Summary

Our system integrates image analysis, machine learning models (including YOLO, Mask R-CNN), and OBD-II data for comprehensive vehicle condition assessment. The CapsNet model has achieved 90% classification accuracy for vehicle damage detection, while Mask R-CNN provides an 80% mean average precision (mAP) for damage localization. The integration of these models with real-time OBD-II diagnostics offers a robust solution for automating both vehicle damage detection and severity classification. This combination significantly enhances the efficiency of processes such as insurance claim processing, vehicle repairs, and resale evaluations, offering a holistic approach to vehicle health and condition monitoring.

Table 2.1: Summary of Literature on Vehicle Damage Detection and OBD-II Integration

Paper	**Year**	**Algorithm / Method**	**Input Data**	**Results / Findings**
Dorathi Jayaseeli et al.	2021	Mask R-CNN	Vehicle Images	Achieved mAP of 88.1% for damage detection and localization.
Kundjanasith et al.	2021	CapsNet	Vehicle Images	98.47% classification accuracy for damage detection.
Rahul Rudra et al.	2023	Mask R-CNN	Vehicle Images	Accurate component-level damage detection in complex scenarios.
Mallios et al.	2023	Semantic Segmentation + DL	Mobile Images	Precise damage region identification.
Mallios et al.	2023	Feature Fusion with Structured Info	Images + OBD-II	Accurate severity classification from fused data.
Norawit Urailertprasert et al.	2021	SVM	Vehicle Images	90% accuracy for binary damage classification.
Dorathi Jayaseeli et al.	2021	YOLOv3 + Decision Tree	Images + OBD-II	Unified health rating (0–1 scale) from visual and engine data.
Kundjanasith et al.	2021	Mask R-CNN + Rule-Based Scoring	Images + OBD-II	Damage + OBD-II scoring system with output in condition categories.
Mallios et al.	2023	Hybrid Fusion Model	Images + OBD-II	Improved anomaly detection and fault prediction.
Xia et al.	2022	ML Model	Images + OBD-II	Real-time updates on engine health for predictive maintenance.

2.2.7 Sources Cited

- Mallios, Dimitrios, Xiaofei Li, Niall McLaughlin, Jesus Martinez-del-Rincon, Clare Galbraith, and Rory Garland. (2023). Vehicle Damage Severity Estimation for Insurance Operations Using In-The-Wild Mobile Images. IEEE Access. PP. 1-1. 10.1109/ACCESS.2023.3299223.
- Jayawardena, Srimal. (2013). Image Based Automatic Vehicle Damage Detection.
- Thonglek, Kundjanasith, Urailertprasert, Norawit, Pattiyathanee, Patchara, and Chantrapornchai, Chantana. (2021). Damaged Vehicle Parts Detection Platforms using Deep Learning Techniques. ECTI Transactions on Computer and Information Technology (ECTI-CIT). 15. 313-323. 10.37936/ecti-cit.2021153.223151.
- Jayaseeli, J D, Dorathi, Jayaraj, Greeta, Kanakarajan, Mehaa, and Malathi, D. (2021). Car Damage Detection and Cost Evaluation Using MASK R-CNN. 10.1007/978-981-16-3153-5_31.

- Rudra, Rahul, Sarker, Praloy, Khan, Farhan Hai, Guha, Amartya, and Maity, Romit. (2023). Instance Segmentation for Car Damage Detection with Mask-RCNN.

CHAPTER 3

PROPOSED METHODOLOGY AND ARCHITECTURE

3.1 System Working:

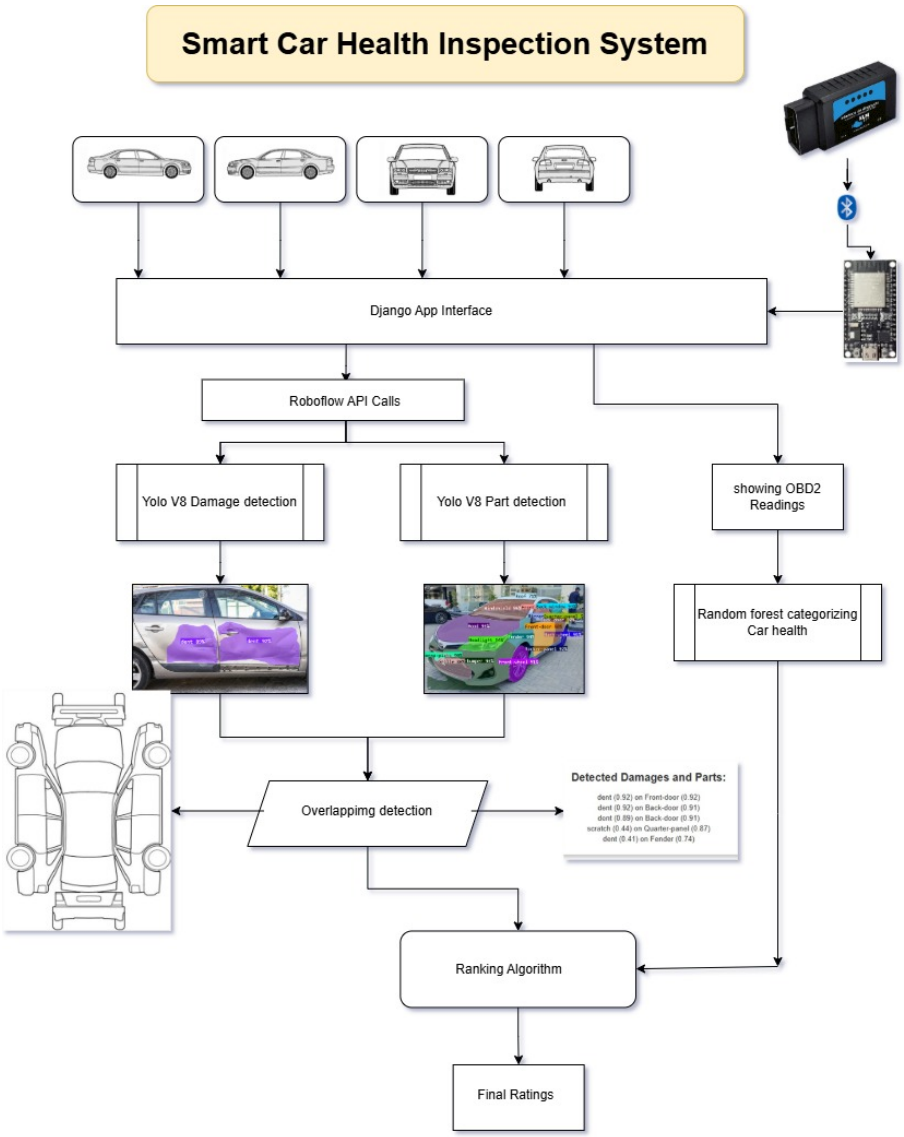


Figure 3.1: Methodology and System Architecture Flow Diagram

Figure 3.1 illustrates the comprehensive architecture of the AI-BASED VEHICLE HEALTH MANAGEMENT SYSTEM, which integrates computer vision and OBD-II-based diagnostics for evaluating a vehicle's condition. The system begins by accepting four-angle images of the vehicle front, rear, left, and right which are uploaded through a Flask-based interface. Simultaneously, an ESP32 microcontroller connected to an OBD-II adapter retrieves real-time diagnostic data from the car's engine and transmits it to the system via Wi-Fi.

The computer vision pipeline employs two YOLOv8 models: one trained for damage detection and the other for car part identification. The damage detection model identifies visual anomalies such as dents, scratches, or broken parts in the uploaded images, while the part detection model segments individual car components like doors, bumpers, and mirrors. These outputs are then fed into an overlapping detection module that maps each detected damage to the corresponding car part by analyzing the spatial overlap of bounding boxes or segmentation masks.

Parallel to this, the engine health assessment module utilizes the collected OBD-II data and processes it through a Random Forest classifier to predict the car's mechanical condition, distinguishing between normal and abnormal statuses. The results from both the computer vision and OBD modules are compiled into a structured damage-part summary.

Finally, a ranking algorithm combines visual and mechanical insights to generate an overall car health rating. This rating, which reflects both external damage and internal engine status, is then displayed to the user as the final output. The architecture ensures a robust and intelligent system for real-time vehicle inspection, providing a holistic evaluation of a car's health condition.

3.2 Data Collection

3.2.1 Image Data

In the data collection phase for image analysis, multiple datasets are being integrated to enhance the system's ability to assess vehicle damage. The Stanford car dataset is combined with smaller datasets that provide specific details about the severity and locality of damage. This integration allows the system to learn from a diverse range of examples, improving the accuracy of external damage detection and classification.The major portion is CarDD dataset developed by Chinese Academy of Sciences. https://cardd-ustc.github.io/

Figure 3.2: Vehicle segmentation for part identification

3.2.2 OBD-II Data

- The ELM327 device retrieves real-time diagnostic data from the vehicle's OBD-II port. Metrics include:
  - Engine RPM
  - Fuel pressure
  - Coolant temperatures
  - Oil temperatures
  - Coolant pressure
  - Oil pressure
- This real-time data is supplemented with existing benchmarks to improve the reliability of the system's internal diagnostics.

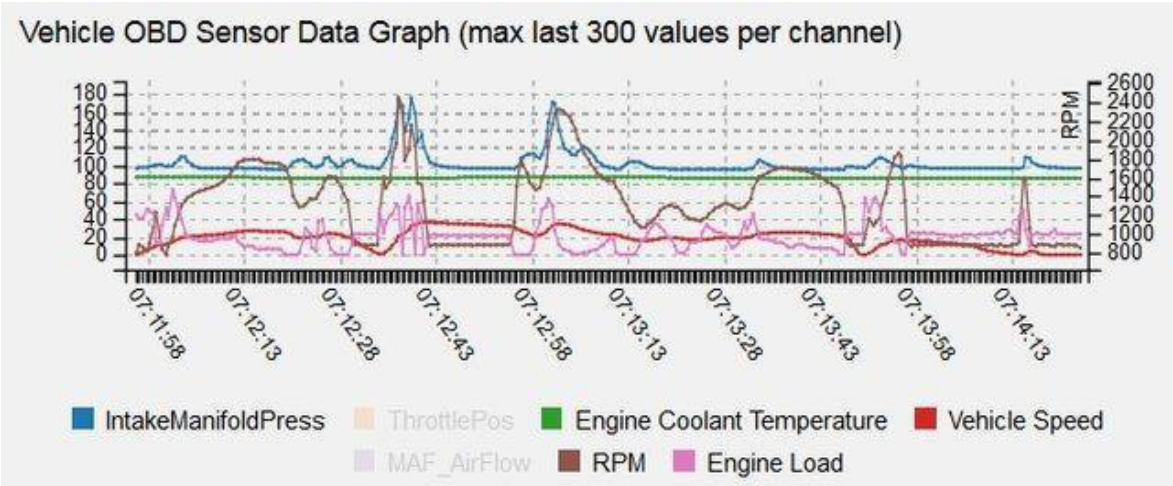


Figure 3.3: Graphical visualization of OBD-II sensor data across key engine metrics [1].

Figure 3.3: presents a graphical visualization of real-time data collected from various OBD-II engine sensors over a short driving session, showcasing a maximum of 300 values per channel. The graph displays multiple engine performance parameters including Intake Manifold Pressure, Throttle Position, Engine Coolant Temperature, Vehicle Speed, MAF Airflow, RPM, and Engine Load. Each metric is color-coded for clarity, with RPM plotted against the secondary y-axis on the right due to its larger scale.

From the graph, RPM shows periodic spikes, indicating engine acceleration and deceleration phases, while the Intake Manifold Pressure and Engine Coolant Temperature remain relatively stable with slight fluctuations. Vehicle Speed and Throttle Position exhibit dynamic changes corresponding to driver inputs. Engine Load and MAF Airflow also show variations aligned with RPM trends, suggesting synchronized behavior between airflow and engine effort. This visualization aids in understanding the relationship between sensor parameters and driving behavior, serving as a diagnostic tool to assess engine health and performance.

3.3 Preprocessing

3.3.1 Image Data Preprocessing

1. **Resizing:**
    - Images are resized to a uniform size (e.g., 416x416) suitable for YOLO model input.
2. **Normalization:**
    - Pixel values are normalized to a range of 0–1 for faster and more stable convergence during model training.
3. **Augmentation:**
    - Techniques such as flipping, rotation, and brightness adjustments are applied to increase dataset diversity.
4. **Filtering:**
    - Gaussian and median filters are used to reduce noise and enhance key features.
5. **Label Encoding:**
    - Labels for part identification and damage classification are encoded in YOLO format, ensuring compatibility with the model.

3.3.2 OBD-II Data Preprocessing

1. **Data Cleaning:**

- Incomplete or erroneous diagnostic data is removed to ensure high-quality analysis.

2. **Standardization:**
   - Numerical metrics like RPM and speed are standardized for consistency across data points.
3. **Feature Extraction:**
   - Features such as average fuel consumption and engine load percentage are derived.
4. **Handling Missing Data:**
   - Strategies like imputation are used to fill in gaps in sensor data.
5. **Normalization:**
   - Diagnostic values are scaled to a uniform range for use in machine learning models.

3.4 YOLO Model for Damage and Part Detection

YOLO (You Only Look Once) is the primary model used for detecting damages and identifying car parts. Its real-time processing capabilities make it ideal for applications requiring high speed and accuracy.

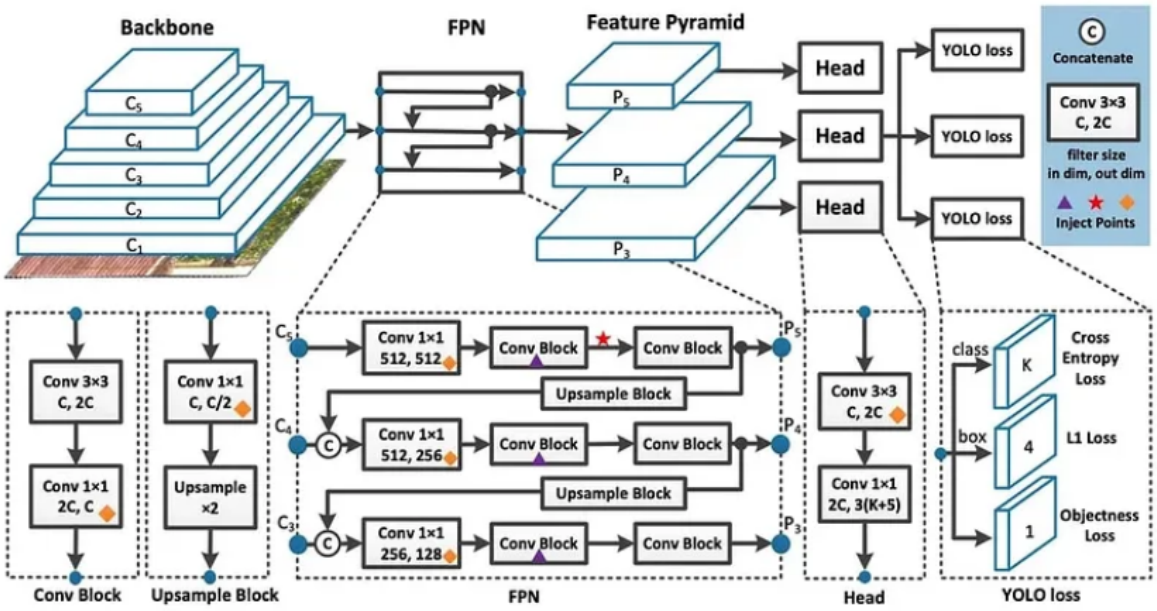


Figure 3.4: YOLO-based neural network architecture for object detection, highlighting backbone extraction, feature pyramid fusion, and detection head with loss components [2].

The YOLO v8 architecture is a state-of-the-art real-time object detection model. It consists of several key components, each playing a vital role in feature extraction and prediction. The first component is the **Backbone**, which is responsible for extracting hierarchical features from the input image. The backbone typically uses a convolutional neural network to progressively downscale the image while capturing essential features. The backbone generates feature maps at different levels, labeled as C_1, C_2, C_3, C_4, C_5, where each level corresponds to a different abstraction level in the feature extraction process.

Next, the architecture incorporates a **Feature Pyramid Network (FPN)**, which enhances feature extraction by combining features from different scales. The FPN allows the model to detect objects of varying sizes more effectively. The arrows in the diagram show how features flow between the different pyramid levels, where the higher levels capture more abstract features and the lower levels focus on finer details. These multi-scale feature maps are referred to as P_3, P_4, and P_5, with P_3 capturing small objects, P_4 capturing medium-sized objects, and P_5 detecting larger objects.

To refine the features further, the architecture includes **Convolutional Blocks**, which apply convolutional operations to the feature maps. The convolutional layers, such as 1×1 and 3×3 kernels, are used to process the features and extract important information. The 1×1 convolutional layers help reduce the depth of the feature maps, while the 3×3 convolutional layers provide a larger receptive field to capture spatial information.

Upsample Blocks are also part of the architecture, which increase the spatial resolution of the feature maps. This step is crucial for detecting smaller objects, as it enhances the feature map's size and resolution. The upsampled features are then passed through the **Head** of the network, which is responsible for making the final predictions. The Head outputs the predicted object class, bounding box coordinates, and objectness score.

The **YOLO Loss** function is used during training to optimize the model's performance. The loss function includes the *class loss* (cross-entropy loss), which evaluates the prediction accuracy for the object class; the *box loss*, which measures how well the predicted bounding box matches the ground truth; and the *objectness loss*, which quantifies how accurately the model predicts the presence of an object in the predicted region. These losses are essential for training the model to detect objects with high precision.

Finally, the model uses **Concatenate** operations to merge feature maps from various stages, enabling it to handle objects at different scales. Injection points, marked by red stars in the diagram, represent locations where additional features or data can be added to improve the model's performance. The final output of the model consists of three components: the predicted class (K), the bounding box coordinates (4), and the objectness score (1), which together make up the complete object detection result.

3.4.1 Damage Detection

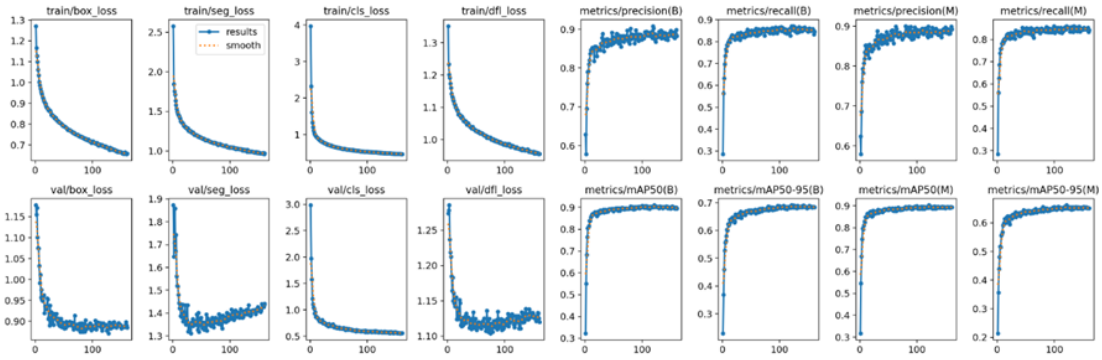


Figure 3.5: Training and validation performance metrics across 120 epochs, including losses, precision, recall, and mean average precision (mAP) for both bounding boxes and segmentation masks

Table 3.1: Explanation of Training and Validation Metrics

Metric	Description
train/box_loss	Measures alignment accuracy of predicted bounding boxes with ground truth; lower values indicate better localization.
train/seg_loss	Evaluates segmentation mask accuracy; decreasing loss suggests improved pixel-wise classification.
train/cls_loss	Classification error for object labels; reduction implies better label predictions.
train/obj_loss	Objectness confidence error; lower values indicate fewer false detections.
val/box_loss	Validation box loss; should follow similar trends as training box loss, else overfitting may be present.
val/seg_loss	Mask loss on validation data; evaluates segmentation generalization ability.
val/cls_loss	Validation class prediction error; decreasing trend is desirable.
val/obj_loss	Object confidence loss for validation; ensures model's detection remains consistent on unseen data.
metrics/precision(B)	Measures how many predicted boxes are correct (bounding box precision).
metrics/recall(B)	Measures how many actual objects are detected (bounding box recall).
metrics/precision(M)	Precision of mask predictions; evaluates pixel accuracy of segmentation.
metrics/recall(M)	Recall for masks; how much of the object is correctly segmented.
metrics/mAP50(B)	mAP@IoU=0.5 for boxes; common benchmark metric for detection accuracy.
metrics/mAP50-95(B)	Average precision across multiple IoUs (0.5 to 0.95); evaluates robustness of detection.
metrics/mAP50(M)	mAP@0.5 for segmentation; evaluates pixel-level detection accuracy.
metrics/mAP50-95(M)	Strict segmentation evaluation metric across IoUs; ensures robustness in mask predictions.

- The YOLO-based model identifies various damage types such as:
  - Scratches
  - Dents
  - Cracks
  - Broken parts
- Outputs include polygon with confidence scores for each damage type.

3.4.2 Part Detection

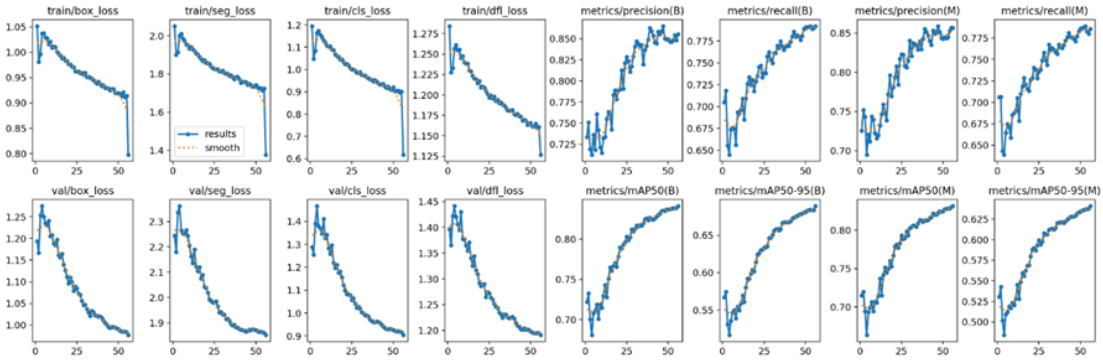


Figure 3.6: Training and validation performance metrics across 54 epochs, including losses, precision, recall, and mean average precision (mAP) for both bounding boxes and segmentation masks

Table 3.2: Explanation of Part Detection Metrics and Losses

Metric / Loss	Explanation and Observations
train/box_loss	Bounding box regression loss during training. Decreasing trend shows better object localization.
train/seg_loss	Measures how well the segmentation mask matches the ground truth. Decreasing curve indicates improving segmentation accuracy.
train/cls_loss	Classification loss for part categories. Downward trend suggests improved label predictions.
train/dfl_loss	Distribution Focal Loss (DFL) is used for bounding box quality refinement by predicting discrete probability distributions over locations. Decline indicates better bounding box precision.
val/box_loss	Validation loss for bounding box predictions. Lower values confirm generalization to unseen data.
val/seg_loss	Validation segmentation loss. Consistently falling values demonstrate stable mask prediction.
val/cls_loss	Part classification loss on validation images. Trend confirms generalization of the classifier.
val/dfl_loss	Validation Distribution Focal Loss. Tracks bounding box quality on validation images.
metrics/precision(B)	Bounding box precision. Steady increase implies fewer false positives in part localization.
metrics/recall(B)	Bounding box recall. Improvement shows that more true objects are being correctly detected.
metrics/precision(M)	Mask-level precision. Increasing curve shows accurate part segmentation with fewer false positives.
metrics/recall(M)	Mask-level recall. Indicates successful segmentation coverage over ground truth areas.
metrics/mAP50(B)	mAP at IoU 0.5 for bounding boxes. Measures detection accuracy with a moderate overlap requirement.
metrics/mAP50-95(B)	Mean Average Precision at multiple IoUs (0.5–0.95) for boxes. Comprehensive detection metric.
metrics/mAP50(M)	mAP@0.5 for masks. Shows segmentation success at standard IoU.
metrics/mAP50-95(M)	Stricter mAP evaluation over multiple IoUs for masks. Indicates high segmentation quality.

- A second YOLO model detects and identifies specific car parts, including:
  - Back-bumper

  - Back-door (Left)
  - Back-door (Right)
  - Back-wheel (Left)
  - Back-wheel (Right)
  - Back-window (Left)
  - Back-window (Right)
  - Back-windshield
  - Fender (Left)
  - Fender (Right)
  - Front-bumper
  - Front-door (Left)
  - Front-door (Right)
  - Front-wheel (Left)
  - Front-wheel (Right)
  - Front-window (Left)
  - Front-window (Right)
  - Grille
  - Headlight (Left)
  - Headlight (Right)
  - Hood
  - License-plate
  - Mirror (Left)
  - Mirror (Right)
  - Quarter-panel (Left)
  - Quarter-panel (Right)
  - Rocker-panel (Left)
  - Rocker-panel (Right)
  - Roof
  - Tail-light (Left)
  - Tail-light (Right)
  - Trunk
  - Windshield
- This model ensures accurate localization of damages on the vehicle.

3.4.3 Polygon Overlap

- Detected polygon for damages and parts are compared to determine which part is affected by the damage. For example:
  - If a damage polygon overlaps with the front door, the system assigns the damage to that part.The IOU ratio selects the damaged part.

3.5 Health Scoring Algorithm

The health scoring algorithm combines damage severity, part criticality, and OBD-II data to calculate a comprehensive health score:

3.5.1 Input Data

- Images of the car taken from different angles (front, back, left, right).
- Damage detection results from the machine learning model, including:
  - Location of damage (e.g., front bumper, left door).
  - Type of damage (e.g., scratch, dent, broken part).

3.5.2 Weighting Different Types of Damage

- Assign weights to each type of damage based on severity:
  - Scratch: Low severity, weight = 1
  - Dent: Medium severity, weight = 2
  - Broken part: High severity, weight = 3

3.5.3 Weighting the Locality (Damaged Part)

- Assign weights to different car parts (localities) based on their criticality:
  - Windshield: Critical part, weight = 3
  - Engine: Critical part, weight = 3
  - Bumper: Less critical, weight = 1

– Door: Medium criticality, weight = 2

```
damage_weights = {
    'crack': 6, 'dent': 10, 'glass shatter': 12, 'lamp broken': 10, 'scratch': 4, 'tire flat': 8
}
part_weights = {
    'Back-bumper': 6, 'Back-door': 8, 'Back-wheel': 6, 'Back-window': 6, 'Back-windshield': 10,
    'Fender': 6, 'Front-bumper': 6, 'Front-door': 8, 'Front-wheel': 6, 'Front-window': 10,
    'Grille': 4, 'Headlight': 8, 'Hood': 6, 'License-plate': 4, 'Mirror': 4,
    'Quarter-panel': 6, 'Rocker-panel': 6, 'Roof': 4, 'Tail-light': 4,
    'Trunk': 4, 'Windshield': 10
}
```

Figure 3.7: All weights are self assigned to damage types and locality.

3.5.4 Damage Score Calculation

- For each damaged part, calculate a damage score using the type of damage, the locality, and the area:

$$\text{Damage Score} = \text{Weight (Type of Damage)} \times \text{Weight (Part)} \quad \text{(Equation 3.1)}$$

3.5.5 Summing All Damage Scores

- Sum the damage scores for all detected damaged parts:

$$\text{Total Damage Score} = \sum (\text{Damage Scores for all parts}) \quad \text{(Equation 3.2)}$$

3.5.6 Normalization of Total Damage Score

- Define a maximum possible damage score based on the worst-case scenario (e.g., large severe damages across critical parts).
- Normalize the Total Damage Score to a value between 0 and 10:

$$\text{Normalized Score} = \left(\frac{\text{Total Damage Score}}{\text{Max Possible Damage Score}} \times 10 \right) \quad \text{(Equation 3.3)}$$

- Subtract the normalized score from 10 to get the Health Rating:

$$\text{Health Rating} = 10 - \text{Normalized Score} \quad \text{(Equation 3.4)}$$

3.5.7 Final Output

- The final Health Rating will be a number between 0 and 10, considering type, location, and size (area) of damage.

3.6 Model for OBD-II Data

For processing OBD-II data, a Gradient Boosting Classifier (GBC) is used to classify the vehicle's health status based on sensor readings.

3.6.1 Gradient Boosting Classifier (GBC)

- The Gradient Boosting Classifier is an ensemble learning method that builds multiple decision trees sequentially. Each tree corrects the errors of the previous one, resulting in a more accurate model. GBC is highly effective for handling complex, high-dimensional data, making it well-suited for predicting the health of the engine based on various OBD-II readings.

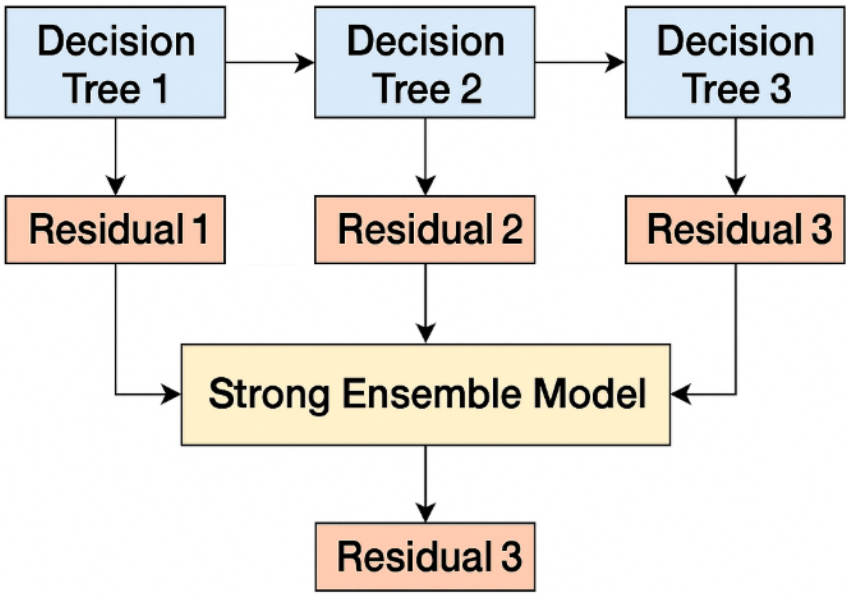


Figure 3.8: Visual representation of the Gradient Boosting Classifier (GBC) workflow, illustrating how multiple decision trees are trained sequentially. Each tree attempts to correct the errors of the previous one, resulting in a progressively more accurate model used for engine health prediction based on OBD-II sensor data.

3.6.2 Features Used in the Model

Table 3.3: Features Utilized in the GBC Model for Engine Health Prediction

Feature	**Description**
Engine RPM	Revolutions per minute of the engine, indicating engine activity level.
Lubricating Oil Pressure	Pressure of the lubrication oil, reflecting oil flow health.
Fuel Pressure	Pressure of the fuel system, used to identify fuel delivery anomalies.
Coolant Pressure	Pressure in the coolant system to detect potential leaks or overheating.
Lubricating Oil Temperature	Temperature of the lubrication oil, associated with engine stress and performance.
Coolant Temperature	Temperature of the coolant, critical for detecting overheating conditions.

3.6.3 Ranking System

3.6.4 Input Data

- Car ID is fed as an input to OBD system.
  - enter CarID (e.g., 103, 201).

3.6.5 Model Functionality

- The system takes the input features and returns 0 and 1 meaning normal and non normal engine.All the instances are passed to the model and returns engine health.

3.6.6 Rating and Confidence Calculation

The rating is calculated based on the proportion of ”NORMAL” predictions. The formula for the rating is:

$$\text{Rating} = \left(\frac{\text{normal\_count}}{\text{normal\_count} + \text{non\_normal\_count}} \right) \times 10 \qquad \text{(Equation 3.5)}$$

Where:

- normal_count is the number of instances predicted as ”NORMAL”.
- non_normal_count is the number of instances predicted as ”NON-NORMAL”.

Confidence Calculation

The confidence is calculated as the proportion of ”NORMAL” or ”ABNORMAL” predictions:

For NORMAL prediction confidence:

$$\text{Confidence (NORMAL)} = \frac{\text{normal\_count}}{\text{normal\_count} + \text{non\_normal\_count}} \qquad \text{(Equation 3.6)}$$

For NON-NORMAL prediction confidence:

$$\text{Confidence (NON-NORMAL)} = \frac{\text{non\_normal\_count}}{\text{normal\_count} + \text{non\_normal\_count}} \qquad \text{(Equation 3.7)}$$

Example

If there are 8 ”normal” predictions and 2 ”non-normal” predictions:

- normal_count $= 8$
- non_normal_count $= 2$
- total $= 10$
- Rating: Rating $= \left(\frac{8}{10}\right) \times 10 = 8.0$
- Confidence (NORMAL): Confidence $= \frac{8}{10} = 0.80$ (or 80% confidence in ”NORMAL”)

3.6.7 Final Output

- The final Health Rating will be a number between 0 and 10, considering the confidence.

3.7 Results

3.7.1 Computer Vision Part

Table 3.4: Model Performance Metrics for Damage and Part Detection

Metric	**Damage Detection (Scenario A)**	**Part Detection (Scenario B)**
mAP@50	86.5%	84.1%
Precision	87.3%	85.6%
Recall	82.5%	79.2%

Figure presents a comparative evaluation of object detection performance in two scenarios (A and B) based on three key metrics: mean Average Precision at 50% IoU threshold (mAP@50), Precision, and Recall. Scenario A achieved higher values across all metrics, with a mAP@50 of 86.5%, Precision of 87.3%, and Recall of 82.5%. In contrast, Scenario B recorded slightly lower performance with a mAP@50 of 84.1%, Precision of 85.6%, and Recall of 79.2%.

3.7.1 Evaluation Metrics

Precision, Recall, and mAP@50 are commonly used metrics to evaluate object detection models. These metrics provide insights into the accuracy and completeness of the model's predictions.

3.7.2 Precision and Recall

Precision and Recall are defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{(Equation 3.8)}$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad \text{(Equation 3.9)}$$

where:

- TP = True Positives (correct detections),
- FP = False Positives (incorrect detections),
- FN = False Negatives (missed detections).

3.7.3 Mean Average Precision (mAP@50)

mAP@50 refers to the mean of the average precisions across all classes, where a detection is considered correct if the Intersection over Union (IoU) with the ground truth is at least 0.5.

$$\text{AP} = \int_0^1 p(r)\, dr \quad \text{(Equation 3.10)}$$

$$\text{mAP@50} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \quad \text{(Equation 3.11)}$$

where:

- $p(r)$ is the precision as a function of recall,
- N is the total number of object classes,
- AP_i is the Average Precision for the i^{th} class at IoU threshold 0.5.

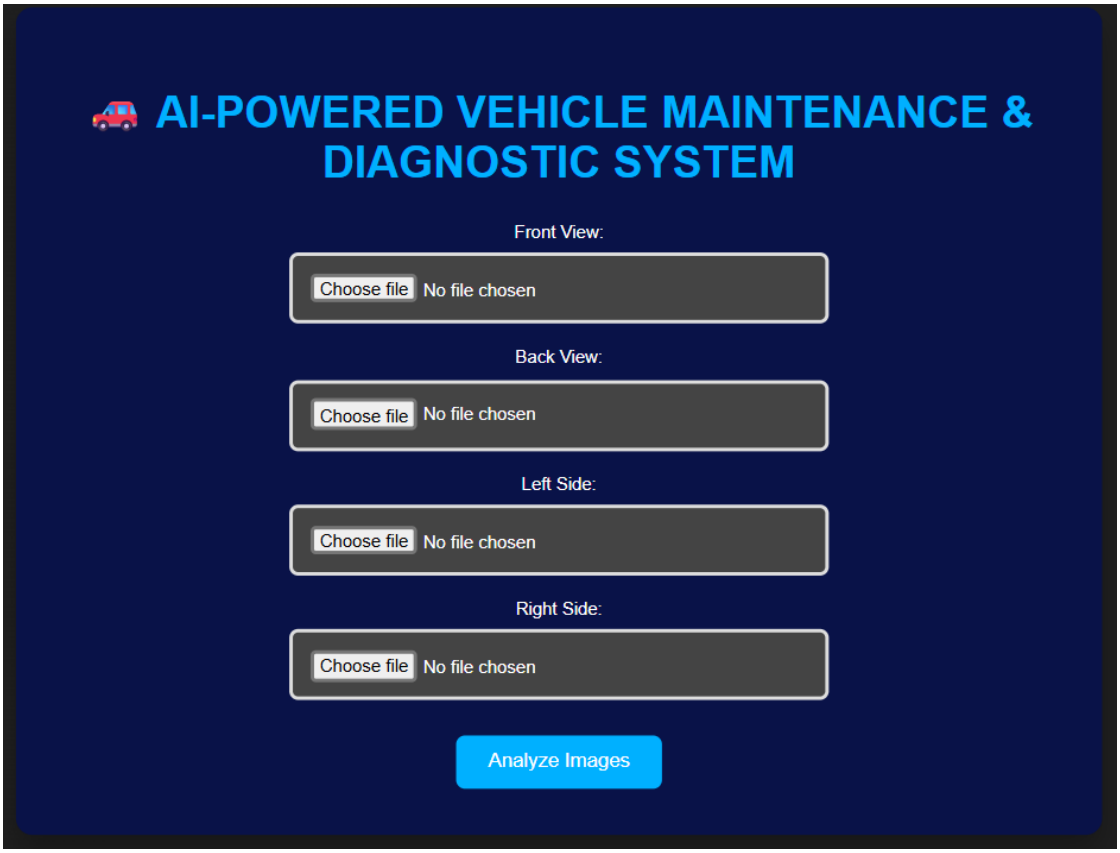
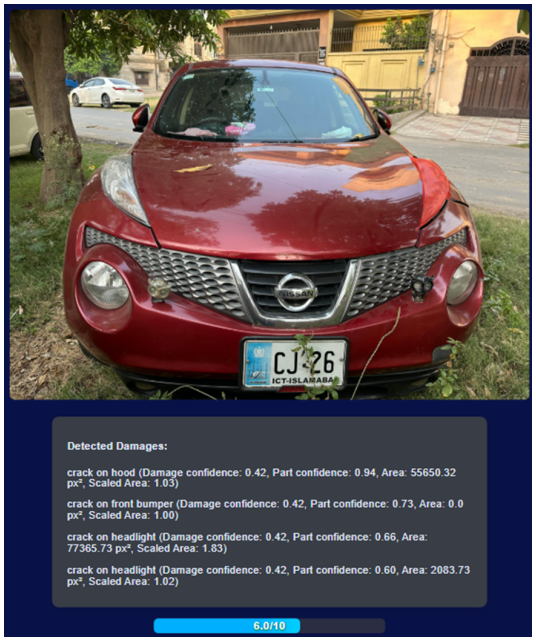
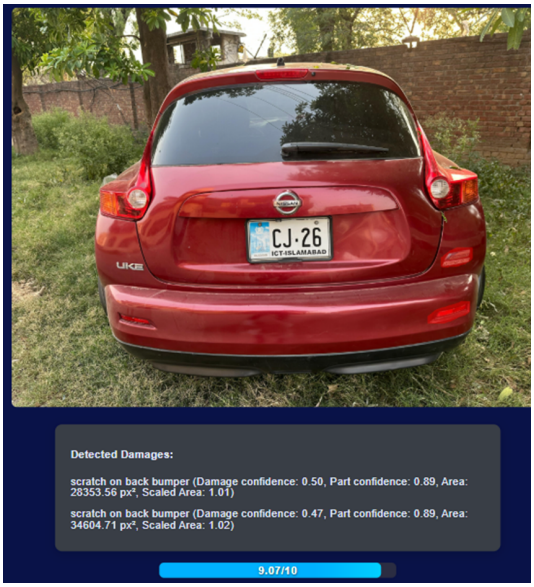


Figure 3.9: User interface for uploading vehicle images across multiple viewpoints, enabling AI-based visual diagnostics

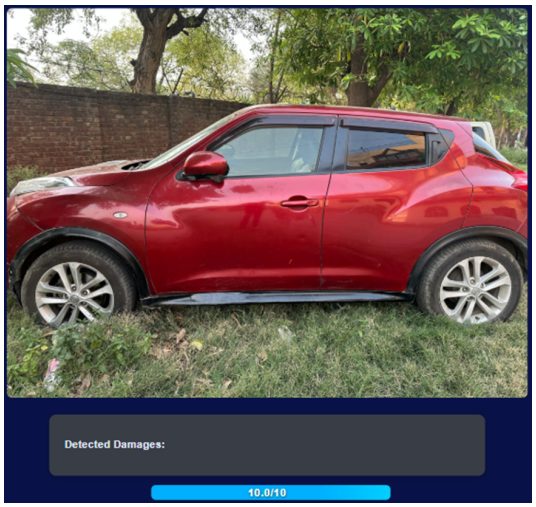
Figure 3.9 presents the graphical interface designed for AI-driven vehicle diagnostics. It allows users to upload images of the car from four key perspectives—front, rear, left, and right ensuring comprehensive visual input. Each section includes clearly labeled upload fields for ease of use, while the ”Analyze Images” button initiates the backend evaluation process. The layout emphasizes clarity and accessibility, providing an intuitive workflow even for users with limited technical background



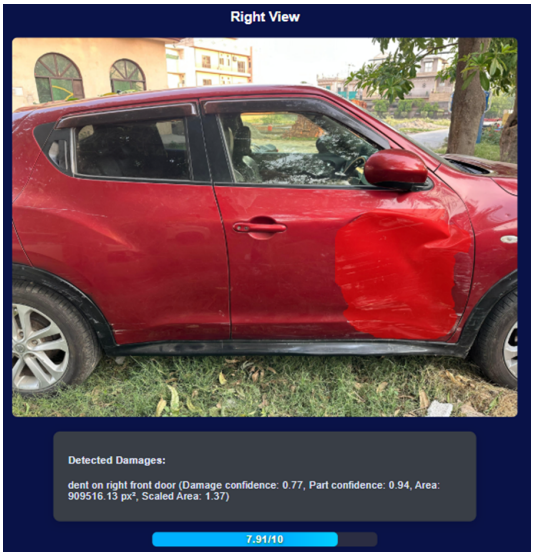
(a) Damage Detection On Front



(b) Damage Detection On Back



(c) Damage Detection On Left



(d) Damage Detection On Right

Figure 3.10: Model evaluation results for damage detection across all vehicle

Figure 3.10 showcases the AI model's performance in detecting surface damage across all vehicle perspectives. Each subfigure illustrates a distinct view—front, rear, left, and right—of the same vehicle, annotated with detected anomalies such as cracks, scratches, and dents. The model provides detailed metadata including damage and part confidence scores along with affected surface areas. Based on this analysis, the system generates a final health rating represented as a numeric value ranging from 0 to 10, where 0 indicates the poorest condition and 10 reflects optimal vehicle health. This visual output demonstrates the model's capability to localize and classify damage with precision, forming a crucial step toward automated vehicle inspection systems.

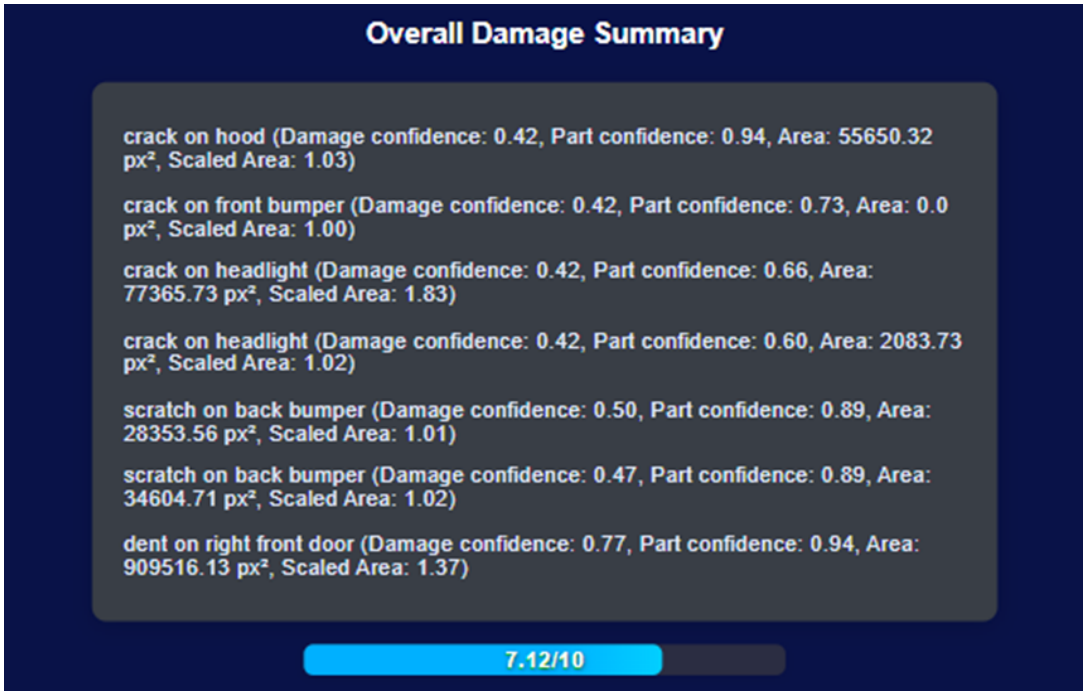


Figure 3.11: Consolidated damage summary showing predicted confidence scores, affected vehicle parts, and scaled damage areas, with an overall vehicle health score of 7.12/10

Figure 3.11 presents the overall damage summary, which consolidates the output of the computer vision models. Each detected damage is listed with its respective damage confidence score, part confidence score, estimated damage area in pixels, and a scaled damage area normalized with respect to the overall image. This summary is generated after processing all four vehicle angle images through trained YOLOv8 models. The system calculates an overall vehicle health score (e.g., 7.12/10 in this case) using a ranking algorithm that incorporates the number of damages, their severities, part importance, and damage area scaling factors.

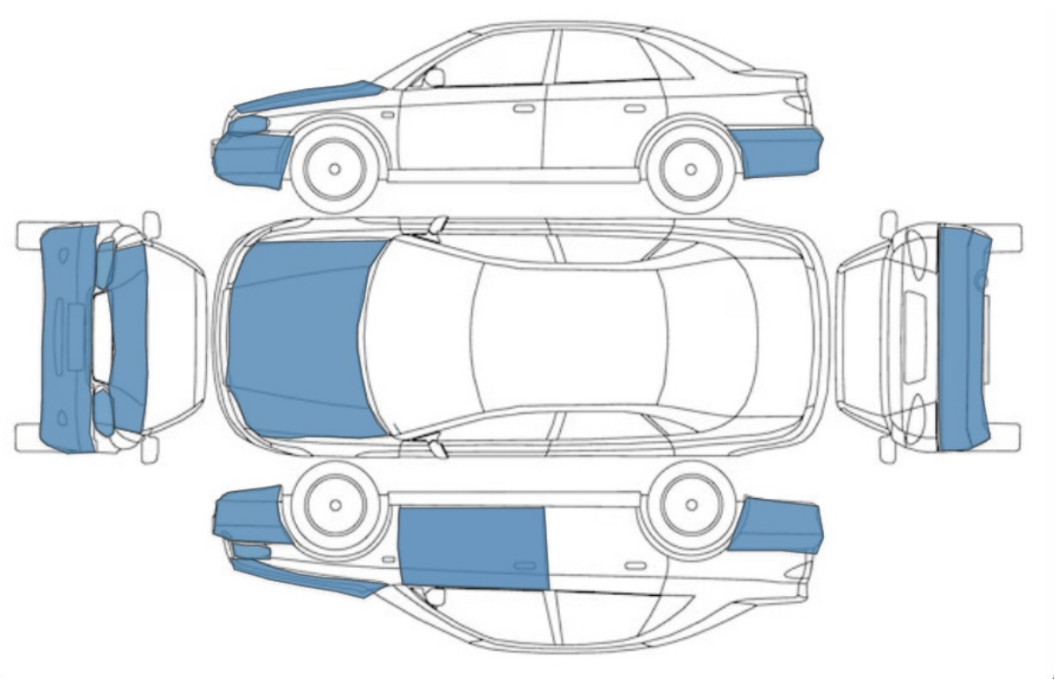


Figure 3.12: Vehicle inspection sheet interface highlighting target parts across multiple viewpoints for automated annotation and damage localization

Figure 3.12 shows the vehicle inspection sheet interface, which uses SVG (Scalable Vector Graphics) to highlight the affected parts across multiple vehicle viewpoints—top, front, back, left, and right. Each part is assigned a unique ‘id‘ and coordinate mapping within the SVG file, allowing real-time interaction through JavaScript or Python Flask. When a damage is detected on a part (e.g., ”right front door”), its corresponding ‘id‘ in the SVG is programmatically highlighted using color overlays. This dynamic SVG mapping enables automatic annotation and intuitive visualization of which areas of the vehicle have sustained damage, ensuring clear, multi-angle representation for inspection purposes.

The integration of SVG with damage metadata allows for scalable, lightweight, and fully automated mapping of detected issues without manual drawing or fixed image overlays. This mechanism is essential for providing users and service technicians with an interactive and precise visual summary of the car’s condition.

3.7.4 OBD Part

The final Engine Health Rating will be a value between 0 and 10, based on the classification result and the confidence level from the Gradient Boosting Classifier (GBC). A higher rating indicates a healthier engine, while a lower rating suggests potential issues. This rating takes into account various sensor readings, such as Engine RPM, Lubricating Oil Pressure, Fuel Pressure, and others, to provide a comprehensive assessment of the vehicle’s engine condition.

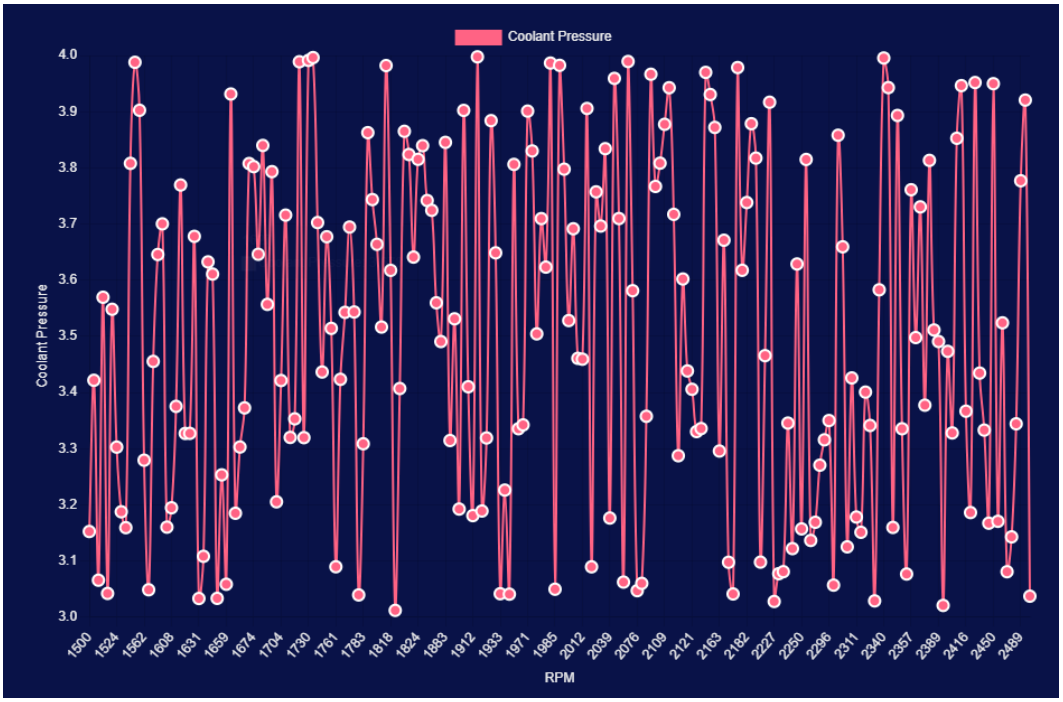


Figure 3.13: Coolant pressure trends across varying RPM levels, visualized for engine health prediction

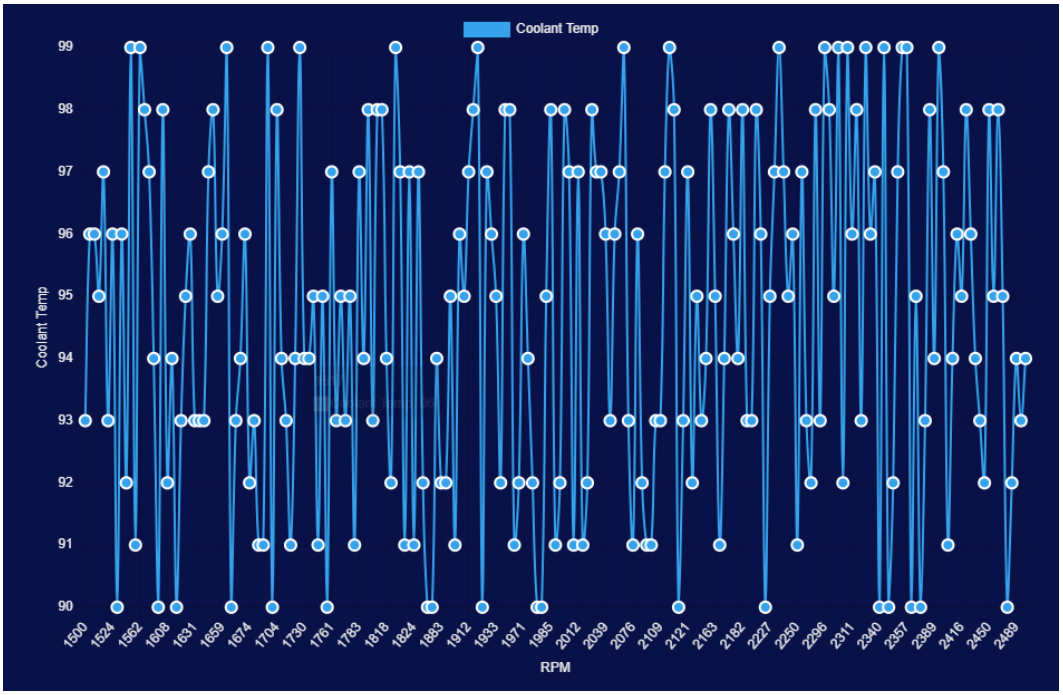


Figure 3.14: Coolant Temperature trends across varying RPM levels, visualized for engine health prediction

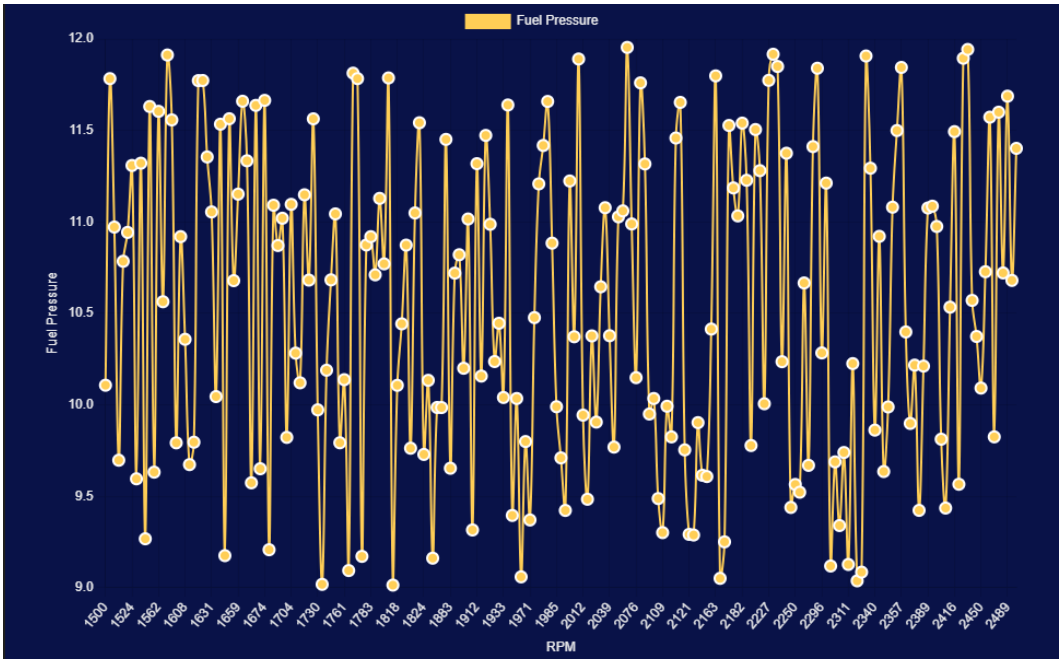


Figure 3.15: Fuel pressure trends across varying RPM levels, visualized for engine health prediction

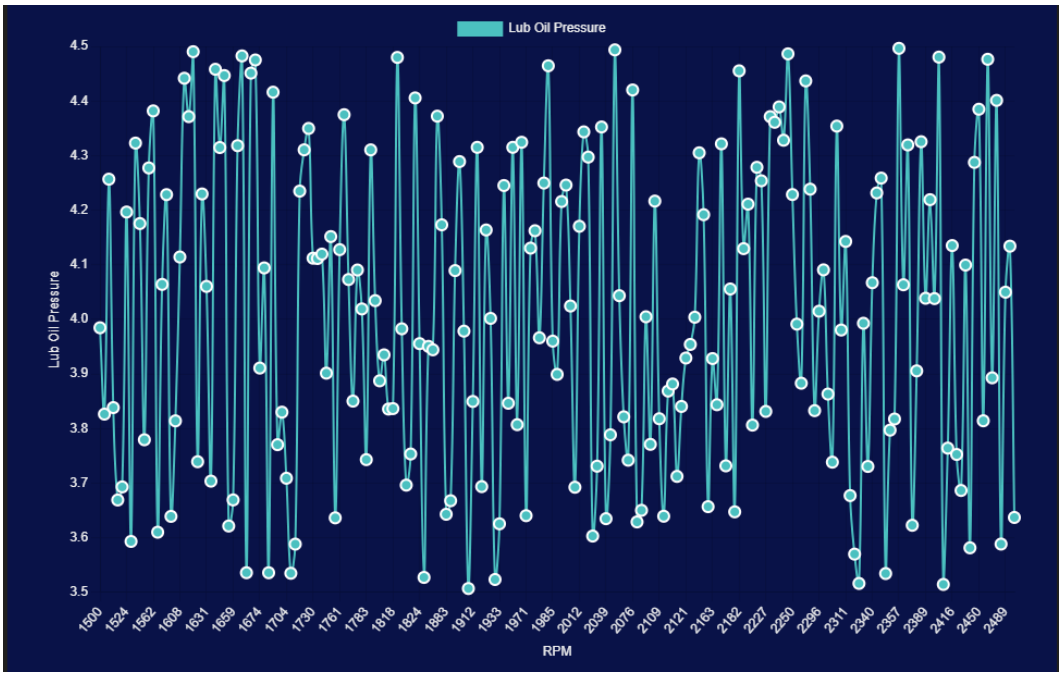


Figure 3.16: Lube Oil pressure trends across varying RPM levels, visualized for engine health prediction

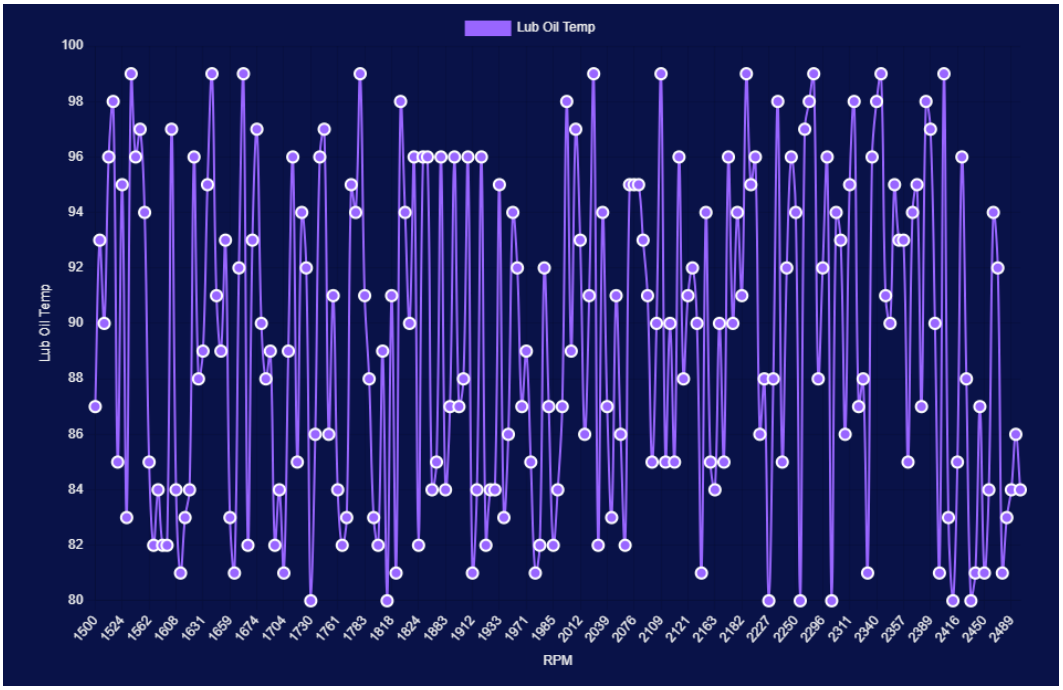


Figure 3.17: Lube Oil Temperature trends across varying RPM levels, visualized for engine health prediction

A higher score indicates a healthier engine.

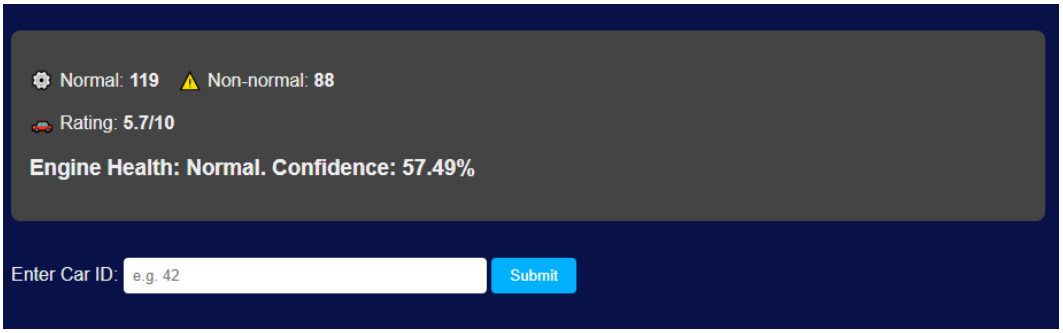


Figure 3.18: OBD-II diagnostics dashboard displaying engine health rating, classification confidence, and entry interface for vehicle-specific queries

3.8 Agile Methodology

For this project, the Agile methodology was the preferred approach due to the need for iterative development and frequent testing. Agile allowed us to get continuous feedback loops, improving the system in short development cycles. Each phase was developed incrementally, with functional testing and updated during every sprint.

3.8.1 Agile Benefits

- **Frequent Deliverables:** Delivered working modules (OBD-II integration, computer vision, emissions) in each sprint.
- **User Feedback:** Regular feedback from stakeholders refined the interface and functionality.
- **Flexibility:** Easily adapted to unforeseen technical challenges and change requests.

CHAPTER 4

MILESTONES, WORK DIVISION, COST AND RISK MANAGEMENT

4.1 Milestones

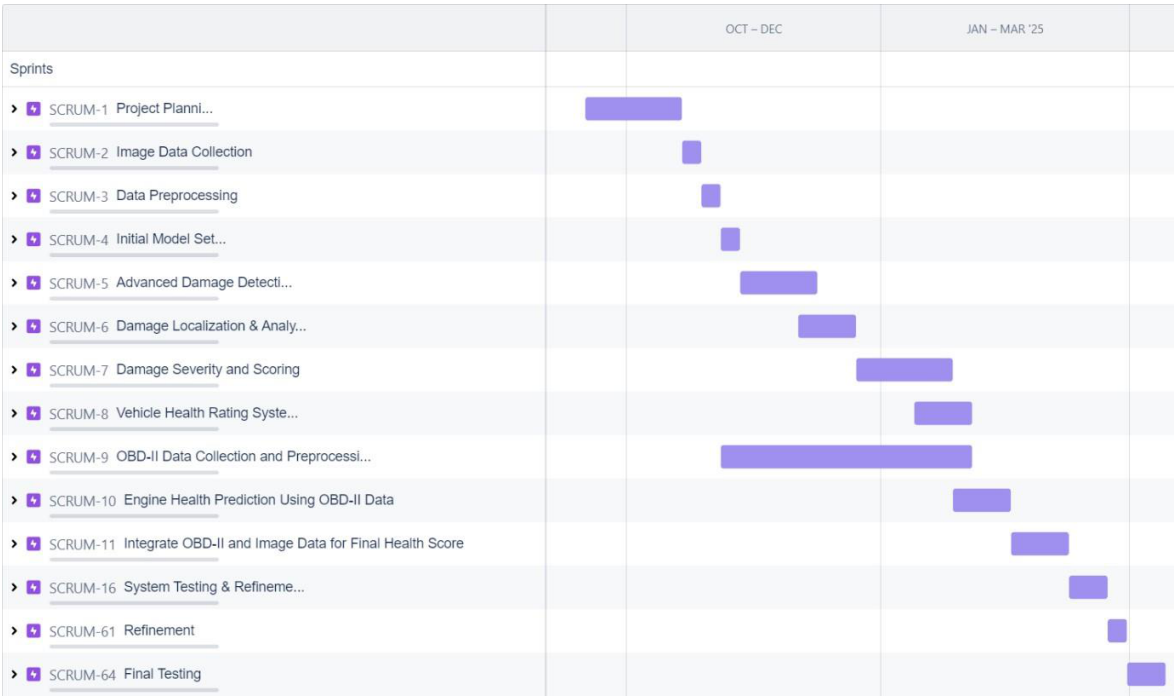


Figure 4.1: Gantt chart depicting project development timeline across 14 SCRUM sprints, covering planning, model development, OBD-II integration, and final testing phases

The development of the AI-Based Vehicle Health Management System involved several key milestones, each with an estimated time for completion. The project was divided into two main parts: the CV (Computer Vision) part and the OBD (On-Board Diagnostics) part, with distinct tasks and timelines for each. For the CV part, we completed dataset collection in 3 weeks, followed by 6 weeks of model research and training. Meanwhile, the OBD part took 5 weeks for data extraction and another 5 weeks to develop the anomaly detection model. Once both parts were ready, the integration of the CV and OBD systems into the front-end was completed in 4 weeks, with 2 weeks allocated for final testing and deployment, ensuring the system was fully functional and ready for use. These milestones are further detailed in the following subsections:

4.1.1 Computer Vision Part

YOLO-Based Model Development

1. **Damage Detection Model:**

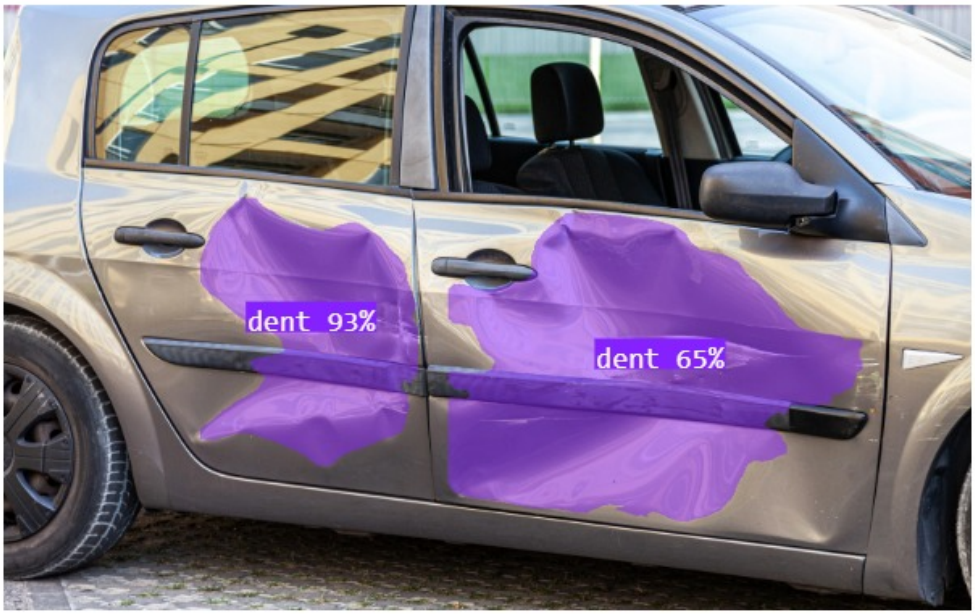


Figure 4.2: Visual representation of AI-predicted door dents with confidence scores of 93% and 65%, aiding part-specific damage localization

Dataset:

- Collected a dataset with annotated labels for damage types.
- Applied data augmentation using flipping, rotation, and scaling techniques.

Model Training:

- Used YOLOv8 for high accuracy and real-time inference capabilities.
- Optimized hyperparameters such as learning rate and batch size.
- Achieved mAP@50 of 84.3% and an average inference time of 42 ms/image.

Challenges and Solutions:

- Addressed imbalance in damage classes through targeted data augmentation.

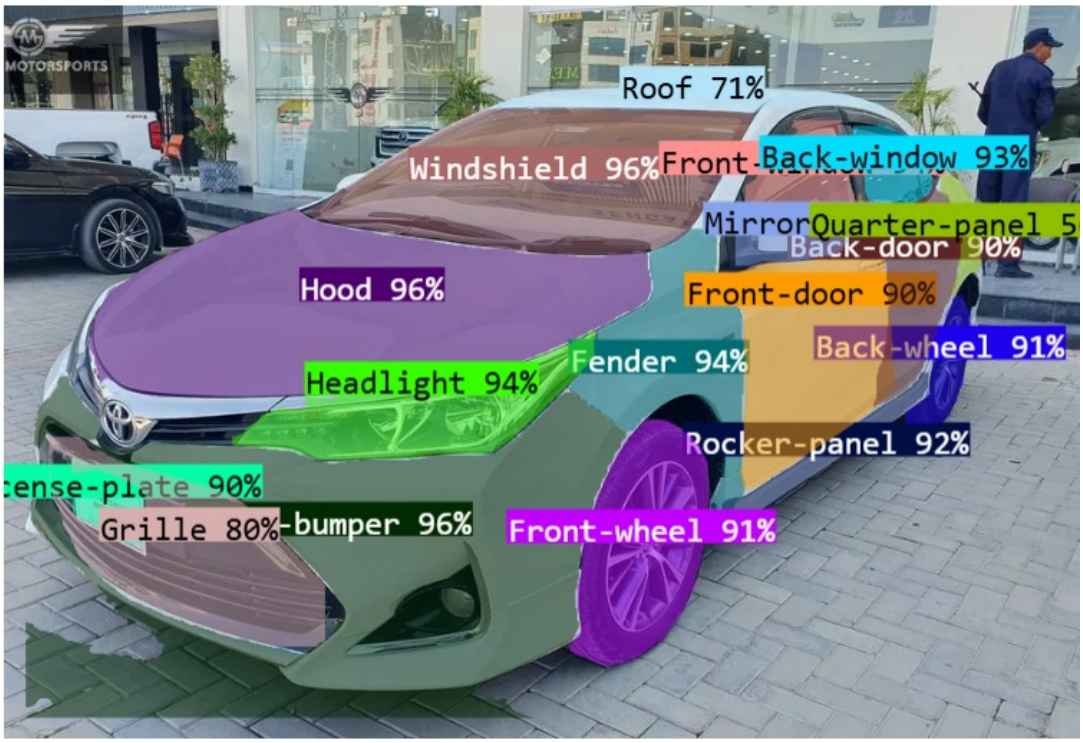


Figure 4.3: AI-based part recognition visualization displaying 16 identified vehicle components with corresponding confidence scores, enabling targeted diagnostics and decision-making

2. **Part Detection Model:**

Dataset:

- Annotated car parts like doors, bumpers, windows, and headlights.
- Balanced underrepresented classes (e.g., roofs) through oversampling.

Model Training:

- YOLOv8 used for efficient detection of small objects.
- Achieved an mAP50 of 87.6% and average inference time of 38 ms/image.

Challenges and Solutions:

- Complex images with overlapping parts were addressed by increasing IoU thresholds.

3. **Polygon Overlap Detection:**

Objective: Map damages to specific car parts using polygon overlaps.

Implementation:

- Extracted polygon box coordinates for both damages and parts.
- Calculated Intersection over Union (IoU) to determine overlaps.
- Assigned damages to parts if IoU exceeded a threshold (e.g., IoU 0.5).

Result:

- Example: “Scratch (confidence: 0.85) on Front-Door (confidence: 0.78).”

```
# Initialize the client
CLIENT = InferenceHTTPClient(
    api_url="https://detect.roboflow.com",
    api_key="x4GqQ1Icif9iqC6c703m"
)
```

Figure 4.4: Python code snippet for initializing an HTTP client to access the Roboflow inference API

API Hosting and Integration:

- **Platform:**
  - Roboflow.com
- **Steps Taken:**
  - Exported trained YOLO models to Roboflow.
  - Configured API endpoints for damage and part detection.
  - Tested API response times and accuracy.
- **Outputs:**
  - Damage Detection API: Provides polygon, damage class, and confidence scores.
  - Part Detection API: Provides polygon, part class, and confidence scores.

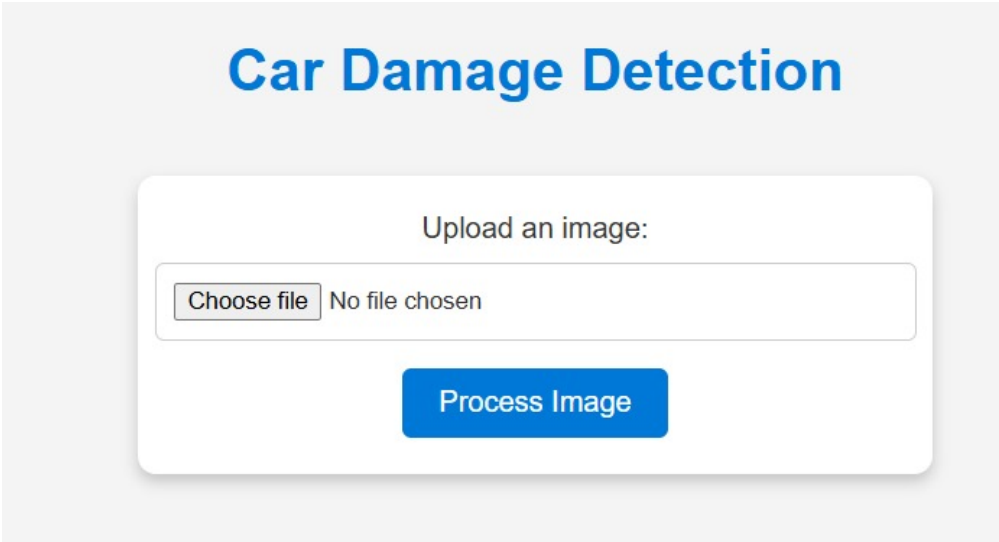


Figure 4.5: Web interface for uploading vehicle images in the car damage detection system, featuring a file selection prompt and trigger for AI-based image processing

Django Integration:

- **Backend:**
  - Integrated APIs for damage and part detection.
  - Processed uploaded images to detect damages and parts.
  - Implemented logic to match overlaps between damages and parts.
- **Frontend:**
  - Built a user-friendly interface to upload images.
  - Displayed annotated outputs with bounding boxes and labels.
- **Features:**
  - Real-time processing and display of results.
  - Detailed annotations showing damaged parts and corresponding damage types.

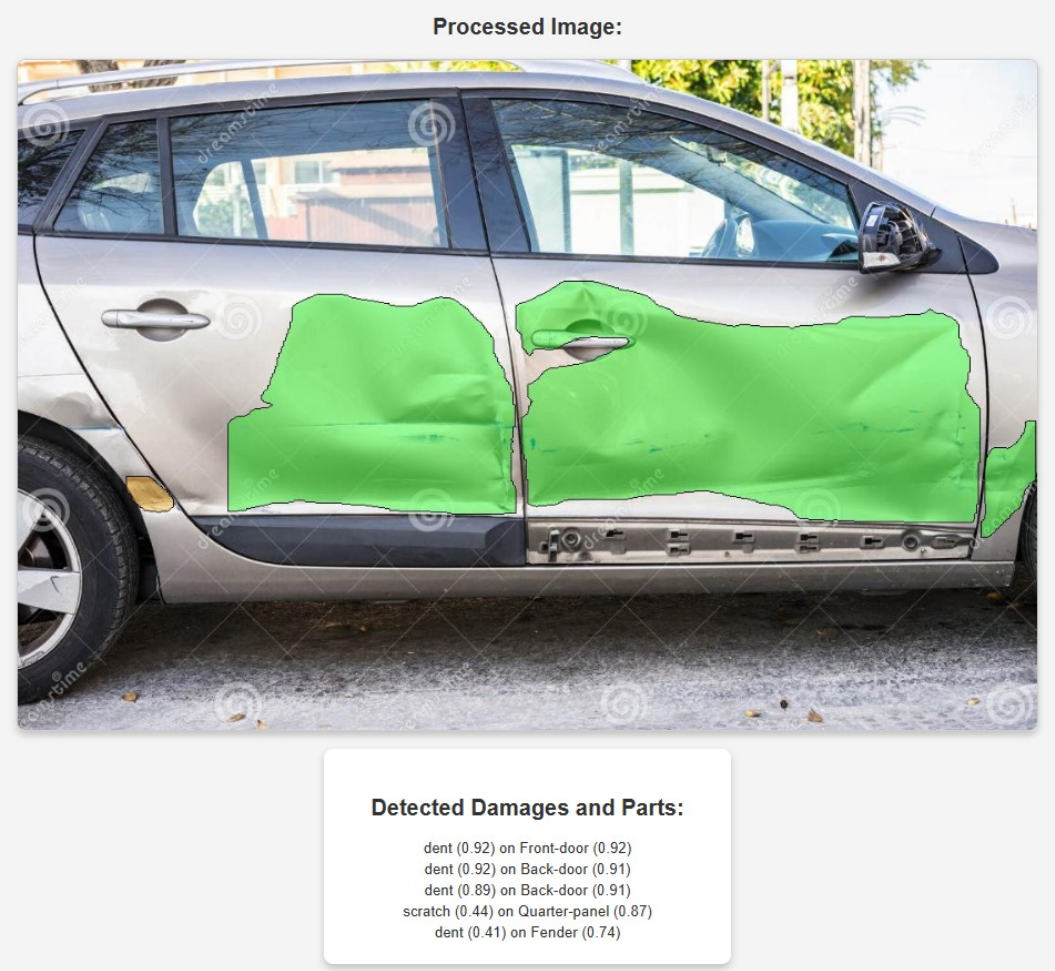


Figure 4.6: Per-instance visualization of vehicle damages with part-level annotations and confidence scores, including multiple dents and a scratch across front and back doors, quarter panel, and fender

Visualization:

- **Outputs:**
  - Annotated images with polygons for damages and parts.
  - Overlaid damage labels and confidence scores on corresponding car parts.
- **User Interaction:**
  - Users can upload car images through the Django app.
  - Processed images are displayed with clear damage-to-part mappings.

Performance Metrics:

Metric	Damage Model	Part Model
mAP@50	84.3%	87.6%
Inference Time (Avg)	42 ms/image	38 ms/image
Precision	82.1%	89.4%
Recall	80.2%	86.7%

Table 4.1: Comparison of Damage Model and Part Model performance metrics, including mAP@50, average inference time, precision, and recall

Challenges and Solutions:

- **Polygon Overlaps:**
  - Issue: False positives in damage-to-part mapping due to small IoUs.
  - Solution: Tuned IoU thresholds and added confidence score weighting.
- **Imbalanced Data:**
  - Issue: Limited examples of certain parts and damages in the dataset.
  - Solution: Data augmentation and oversampling of underrepresented classes.
- **Inference Time:**
  - Issue: Slow processing for larger batch sizes.
  - Solution: Reduced batch sizes and optimized API configurations.

4.1.2 OBD-II Part

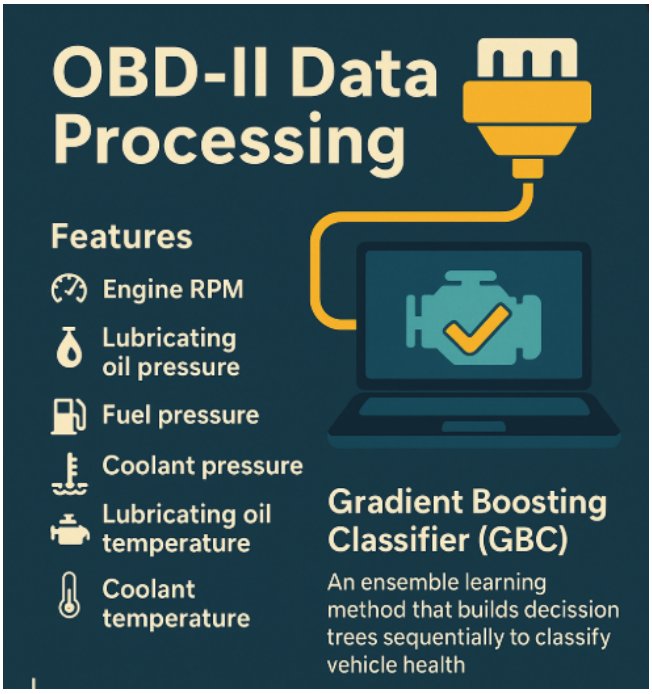


Figure 4.7: The diagram shows how OBD-II data is collected via the ELM327 tool, processed through a .NET system, and used by a Gradient Boosting Classifier to predict engine health. *Image generated with generator model.*

Establishing Communication with Vehicle's Diagnostics:

- The .NET Core application facilitates seamless communication between the vehicle's OBD-II system and the backend system. The connection is established using TCP protocol to transfer data efficiently.
- The connection is established using a predefined IP address (`192.168.0.10`) and port (`35000`).
- Real-time Data Processing is implemented, enabling the query of various critical vehicle parameters such as Engine RPM, Fuel Pressure, and Coolant Temperature.
- Error Handling is incorporated to manage connection failures, incorrect data responses, and timeouts, ensuring smooth communication between the OBD-II system and the backend.

Data Logging and Processing:

- **Key Vehicle Parameters Queried:**
  - **Battery Voltage:** Monitors the vehicle's battery health.
  - **Engine RPM:** Tracks engine revolutions per minute.

  - **Vehicle Speed:** Displays the speed of the vehicle in real-time.
  - **Fuel System Status:** Indicates the operational status of the fuel system.
  - **Engine Load:** Represents the engine's workload as a percentage.
  - **Coolant Temperature:** Monitors the engine coolant temperature for overheating issues.
  - **Oil Temperature:** Tracks the temperature of the engine oil for maintenance purposes.
  - **Short and Long Term Fuel Trim:** Displays adjustments to fuel injection based on air-fuel mixture data.
  - **Distance Traveled with MIL On:** Measures the distance covered while the Malfunction Indicator Lamp (MIL) is active.
  - **Throttle Position and Relative Throttle Position:** Shows the throttle's current and relative positions for driver input analysis.
- These data points provide comprehensive insights into the vehicle's performance and health.
- Logging real-time OBD-II data into Google Sheets through the Google Sheets API on Google Cloud for continuous monitoring and historical analysis.
- Ensuring accurate data retrieval and real-time monitoring of key vehicle diagnostics.

Data Validation and Debugging:

- Each data query is validated to ensure successful data retrieval from the OBD-II tool, ensuring that only accurate and valid data is logged.
- Debugging outputs are available for monitoring system behavior, providing insights into any errors or failures in data communication, helping to identify and fix issues.
- This ensures that only accurate and verified data is logged or processed further, preventing misleading outputs due to communication errors.

Model Integration:

- Integrating Gradient Boosting Classifier (GBC) to predict the Engine Health Rating based on real-time OBD-II data.
- Combining confidence levels with predictions for more accurate engine health assessments.

Real-Time Monitoring and Alerts:

- Enhancing real-time monitoring with predictive maintenance capabilities based on vehicle diagnostics.
- Generating maintenance alerts for proactive intervention, based on model predictions.

Cross-Platform Compatibility:

- Ensuring the .NET Core application runs seamlessly across multiple platforms (Windows, Linux, and macOS).
- Optimizing the system for various operational environments.

Error Handling and Debugging:

- Implementing robust error handling for issues like connection failures, timeouts, and incorrect data responses.
- Using comprehensive debugging tools to ensure smooth operation and data accuracy.

System Scalability and Modularity:

- Designing the .NET Core application using modular programming principles for easy extensions and scalability.
- Adding new diagnostic parameters or system features without affecting existing operations.

4.2 Cost

To support the development and delivery of our automotive diagnostics platform, we have allocated a total budget of Pkr 2,82,800 across key service areas essential for both technical implementation and outreach. The largest portion goes to licensing and software subscriptions, covering tools such as analytics platforms, development environments, and firmware update licenses. Testing and prototyping, as well as marketing/demo materials, also take up a significant share, these ensure robust performance and user visibility. Other critical costs include hardware acquisition such as the OBD-II scanner and embedded edge devices, along with cloud services for remote diagnostics and data handling. A smaller portion is reserved for agile project management tools, streamlining team coordination and iteration cycles. While the majority of hardware and prototyping expenses are one-time, several services such as cloud hosting, development tools, and licensing incur monthly subscription fees that contribute to ongoing operational costs.

One-time Expenses: OBD-II scanner, embedded system, demo materials, prototyping hardware.

Monthly/Recurring Costs: Cloud services, agile tools, software subscriptions.

Component	% of Budget i-e PKR 2,82,800	Type
Licensing/Subscriptions	29.7%	Recurring
Testing & Prototyping	19.8%	One-time
Marketing Materials	19.8%	One-time
OBD-II Scanner	9.9%	One-time
Edge Device	9.9%	One-time
Cloud Services	9.9%	Recurring
Agile Tools	1.0%	Recurring

Table 4.2: Budget allocation by component, indicating percentage distribution and expense type based on a total of PKR 2,82,800

4.3 Computer Vision Tool: Roboflow

To facilitate the development of the damage detection and vehicle part identification models, **Roboflow** was used as the end-to-end computer vision (CV) management platform. Roboflow provided a streamlined interface and a comprehensive set of features that significantly simplified dataset preparation, model training, and deployment.

- **Dataset Annotation:** Roboflow enabled precise polygon-based annotation for both damage categories (e.g., dent, crack, scratch) and vehicle components (e.g., doors, bumpers, windshield). These annotations were used to create separate datasets for damage detection and part localization tasks.
- **Preprocessing and Augmentation:** The platform offered automated data augmentation techniques such as horizontal and vertical flipping, random rotations, noise injection, brightness/-contrast adjustments, and object-centering. These enhancements improved model generalization and performance under diverse lighting and angle conditions.
- **Dataset Splitting:** Roboflow automatically handled the splitting of datasets into training, validation, and test sets (commonly in 70/20/10 proportions), ensuring a balanced distribution of classes across all subsets.
- **Model Export and Compatibility:** The annotated datasets were exported in `YOLOv8` format, compatible with Ultralytics YOLOv8 training scripts. This allowed direct integration into our model training pipeline using PyTorch.
- **Hosted Inference API:** Trained models were deployed using Roboflow's hosted inference API. The endpoint URL and API key were integrated into the Flask-based web application to enable real-time predictions via HTTP requests.

- **Version Control and Dataset Management:** Roboflow maintained version control over dataset updates and labeling changes, allowing seamless iteration and experimentation with different configurations.

Through the use of Roboflow, the computer vision pipeline was significantly accelerated and simplified, enabling rapid experimentation and scalable deployment of object detection models in a production-grade setup.

CHAPTER 5

CONCLUSION

5.1 Conclusion

The AI-Based Vehicle Health Management System is a significant step toward automating vehicle health evaluations by integrating computer vision and real-time diagnostics. By leveraging YOLO-based models for damage and part detection and incorporating OBD-II data for engine health assessment, the system delivers a comprehensive and reliable health report.

5.1.1 Key Features and Benefits

1. **Accuracy:** YOLO models provide precise damage detection and localization, ensuring reliable mapping to car parts.
2. **Efficiency:** Real-time diagnostic data from the ELM327 device minimizes inspection time.
3. **Comprehensiveness:** The health scoring algorithm combines external and internal diagnostics, offering a holistic view of vehicle health.
4. **Flexibility:** The system's modular architecture allows for seamless integration of additional sensors and datasets.

The data preprocessing pipeline ensures high performance, with techniques like resizing, normalization, augmentation, and standardization improving model accuracy and robustness. The use of YOLO models enhances processing speed, making the system suitable for real-world applications such as insurance claims, fleet management, and automotive repair services.

By addressing challenges such as sensor integration and algorithm optimization, the system minimizes errors, reduces subjectivity, and lowers operational costs compared to traditional manual inspections. This project demonstrates how advanced technologies can revolutionize the automotive industry, improving reliability, safety, and user confidence.

5.2 Future Work

Future enhancements to the AI-Based Vehicle Health Management System include:

1. **Integration of Emission Sensors:**
    - Adding emission sensors to monitor exhaust data and evaluate compliance with environmental standards.
    - This integration would provide more detailed insights into engine health and potential environmental impacts.
2. **Predictive Maintenance:**
    - Develop algorithms to predict potential failures based on historical OBD-II and diagnostic data.
    - Enable proactive interventions to prevent costly repairs and improve vehicle longevity.
3. **Enhanced User Interface:**
    - Design an interactive dashboard to visualize detected damages, diagnostics, and health ratings intuitively.
    - Include features for comparing historical health reports and tracking maintenance trends.
4. **Scalability and Deployment:**
    - Optimize the system for deployment in large-scale applications, such as insurance companies and rental services.
    - Implement mobile and web-based solutions to improve accessibility.
5. **Advanced Data Analysis:**
    - Incorporate deep learning techniques for anomaly detection in OBD-II data.
    - Use ensemble methods to enhance damage classification accuracy.
6. **Improved Health Scoring Algorithm:**
    - Refine the algorithm to account for more granular damage metrics, such as the area affected and its repair cost implications.

- Include weight adjustments based on real-world feedback from users and industry stakeholders.

7. **Expanded Dataset Collection:**

   - Collaborate with industry partners to collect diverse and annotated datasets, improving model generalization across various vehicle types and conditions.

REFERENCES

[1] J. Xia, J. Huang, and Y. Lin, "Integrating image-based and sensor-based data for real-time vehicle health monitoring," *Journal of Intelligent Transportation Systems*, 2022.

[2] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[3] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn for object detection and segmentation," *arXiv preprint arXiv:1703.06870*, 2017. [Online]. Available: https://arxiv.org/abs/1703.06870

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *arXiv preprint arXiv:1506.02640*, 2015. [Online]. Available: https://arxiv.org/abs/1506.02640

[5] "Obd-ii standards and diagnostic information," Online; OBDsol Knowledge Base, 2020, https://www.obdsol.com/knowledgebase/on-board-diagnostics/what-is-obd/.

[6] Towards Data Science, "The random forest algorithm," Online, 2021, https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd33.

[7] C. Lin, "A guide to support vector machines," National Taiwan University, Tech. Rep., 2002, https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

[8] J. Krause, "Stanford cars dataset for vehicle recognition," Online, 2013, https://ai.stanford.edu/~jkrause/cars/car_dataset.html.

[9] D. Mallios, X. Li, N. McLaughlin, J. M. del Rincon, and R. Garland, "Vehicle damage severity estimation for insurance operations using in-the-wild mobile images," *IEEE Access*, 2023.

[10] R. Rudra, P. Sarker, F. Khan, A. Guha, and R. Maity, "Instance segmentation for car damage detection with mask-rcnn," *arXiv preprint arXiv:2301.09801*, 2023. [Online]. Available: https://arxiv.org/abs/2301.09801

[11] J. Jayaseeli, J. Dorathi, G. Jayaraj, M. Kanakarajan, and D. Malathi, “Car damage detection and cost evaluation using mask r-cnn,” in *Advances in Computer and Computational Sciences*, 2021.

[12] Y. Wang, Z. Zhang, and Y. Li, “Predictive maintenance based on vehicle sensor data using machine learning,” *IEEE Trans. on Industrial Informatics*, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9052591

[13] K. Kundjanasith, N. Urailertprasert, P. Pattiyathanee, and C. Chantrapornchai, “Damaged vehicle parts detection platforms using deep learning techniques,” *ECTI Trans. on Computer and Information Technology*, vol. 15, no. 3, pp. 313–323, 2021.

[14] Y. Ykara, “Elm327-wifi-obdii-tool,” GitHub repository, 2023, https://github.com/ykara61/ELM327-WIFI-OBDII-TOOL.

Flask Application Code

```
import os
import time
import requests
import joblib
import pandas as pd
from flask import Flask, render_template, request, url_for
from werkzeug.utils import secure_filename
from inference_sdk import InferenceHTTPClient
from PIL import Image, ImageDraw
from shapely.geometry import Polygon
from flask_executor import Executor
from flask_caching import Cache

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = './static/uploads'
app.config['PROCESSED_FOLDER'] = './static/processed'
app.config['CACHE_TYPE'] = 'simple'
cache = Cache(app)
executor = Executor(app)

os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
os.makedirs(app.config['PROCESSED_FOLDER'], exist_ok=True)

# --- Engine Health Model (OBD) ---
MODEL_PATH = "hhmodel.pkl"  # CHANGE TO YOUR PATH
model = joblib.load(MODEL_PATH)
CSV_URL = f"https://docs.google.com/spreadsheets/d/e/2PACX-1
    vQwqdFUJi6LQlHTR9nQM6HRp8h5AXVAAK9JBVhwaNkpUNIwTcXHBTVHHiaFZ8SAiulE
    -sBR6t2ffi6q/pub?output=csv"
car_data = pd.read_csv(CSV_URL)
car_data.columns = car_data.columns.str.strip()
PARAMETERS = [
    ("lub_oil_pressure",        "Lub Oil Pressure"),
    ("fuel_pressure",           "Fuel Pressure"),
    ("coolant_pressure",        "Coolant Pressure"),
    ("lub_oil_temp",            "Lub Oil Temp"),
    ("coolant_temp",            "Coolant Temp"),
    ("Temperature_difference", "Temperature Difference"),
]

# --- Damage Detection API ---
DAMAGE_CLIENT = InferenceHTTPClient(api_url="https://detect.
    roboflow.com", api_key="x4GqQ1Icif9iqC6c703m")
LOCALITY_CLIENT = InferenceHTTPClient(api_url="https://detect
    .roboflow.com", api_key="haFKo8MxrbCGAN839WXu")
damage_weights = {
    'crack': 6, 'dent': 10, 'glass shatter': 12, 'lamp broken
        ': 10, 'scratch': 4, 'tire flat': 8
}
part_weights = {
    'Back-bumper': 6, 'Back-door': 8, 'Back-wheel': 6, 'Back-
        window': 6, 'Back-windshield': 10,
    'Fender': 6, 'Front-bumper': 6, 'Front-door': 8, 'Front-
        wheel': 6, 'Front-window': 10,
    'Grille': 4, 'Headlight': 8, 'Hood': 6, 'License-plate':
        4, 'Mirror': 4,
    'Quarter-panel': 6, 'Rocker-panel': 6, 'Roof': 4, 'Tail-
        light': 4,
    'Trunk': 4, 'Windshield': 10
}

@app.route('/', methods=['GET', 'POST'])
def index():
    results = None
    collective_damage = None
    final_rating_damage = None
    final_damaged_parts = None
    chart_data = None
    normal_count = 0
```

```
    non_normal_count = 0
    final_rating = None
    rating_out_of_10 = None
    obd_result = None
    car_id_input = None

    if request.method == 'POST':
        # Make sure all 4 images and Car ID are provided
        sides = ['front', 'back', 'left', 'right']
        files_ok = all(side in request.files and request.files[side].filename for side in sides)
        car_id_input = request.form.get('car_id')
        if files_ok and car_id_input:
            # -------- Damage Detection --------
            results, ratings, collective_damage = [], [], []
            final_damaged_parts = set()
            futures = []
            for side in sides:
                file = request.files.get(side)
                filename = secure_filename(file.filename)
                file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
                file.save(file_path)
                future = executor.submit(process_image, file_path, side)
                futures.append(future)
            for idx, future in enumerate(futures):
                img_url, damages, rating, damaged_parts = future.result()
                results.append({'side': sides[idx].capitalize(), 'image_url': img_url, 'damage_info': damages, 'health_rating': rating})
                collective_damage.extend(damages)
                ratings.append(rating)
                final_damaged_parts.update(damaged_parts)
            min_rating = min(ratings)
            weighted_ratings = [rating + (min_rating - rating) * 0.5 for rating in ratings]
            final_rating_damage = round(sum(weighted_ratings) / len(weighted_ratings), 2)
            final_damaged_parts = list(final_damaged_parts)
            # -------- OBD/Engine Health for Car ID --------
            try:
                car_id = int(car_id_input)
            except ValueError:
                obd_result = f"Invalid Car ID: {car_id_input}"
                return render_template("index.html", **locals())
            car = car_data[car_data["car_id"] == car_id].sort_values("rpm")
            if car.empty:
                obd_result = f"No data found for Car ID {car_id}"
                return render_template("index.html", **locals())
            chart_data = {}
            rpm_list = car["rpm"].tolist()
            for col, label in PARAMETERS:
                chart_data[label] = {
                    "labels": rpm_list,
                    "data":   car[col].tolist()
                }
            for _, row in car.iterrows():
                df = pd.DataFrame({
                    "Engine rpm":              [row["rpm"]],
                    "Lub oil pressure":        [row["lub_oil_pressure"]],
                    "Fuel pressure":           [row["fuel_pressure"]],
```

```
                    "Coolant pressure":       [row["coolant_pressure"]],
                    "lub oil temp":           [row["lub_oil_temp"]],
                    "Coolant temp":           [row["coolant_temp"]],
                    "Temperature_difference": [row["Temperature_difference"]],
                })
                p = model.predict(df)[0]
                normal_count += (p == 0)
                non_normal_count += (p != 0)
            total = normal_count + non_normal_count
            if total > 0:
                if normal_count > non_normal_count:
                    health, conf = "Normal", normal_count / total
                else:
                    health, conf = "Non-Normal", non_normal_count / total
                rating_out_of_10 = round((normal_count / total) * 10, 1)
                final_rating = f"Engine Health: {health}. Confidence: {conf:.2%}"

    return render_template(
        "index.html",
        results=results,
        collective_damage=collective_damage,
        final_rating_damage=final_rating_damage,
        final_damaged_parts=final_damaged_parts,
        chart_data=chart_data,
        normal_count=normal_count,
        non_normal_count=non_normal_count,
        final_rating=final_rating,
        rating_out_of_10=rating_out_of_10,
        obd_result=obd_result,
        car_id_input=car_id_input
    )
@cache.memoize(timeout=60 * 60)
def process_image(file_path, side):
    retries = 3
    for attempt in range(retries):
        try:
            loc_preds = LOCALITY_CLIENT.infer(file_path, model_id="carpartsdetect/1")['predictions']
            break
        except requests.exceptions.Timeout as e:
            print(f"Request timeout error: {e}. Retrying... ({attempt + 1}/{retries})")
            time.sleep(2)
        except Exception as e:
            print(f"Error occurred during inference: {e}")
            return "", [], 0, []
    dmg_preds = DAMAGE_CLIENT.infer(file_path, model_id="hi-xg1ga/2")['predictions']
    img = Image.open(file_path).convert("RGBA")
    overlay = Image.new("RGBA", img.size, (0, 0, 0, 0))
    draw = ImageDraw.Draw(overlay)
    damage_info, total_score = [], 0
    damaged_parts = []
    for dmg in [d for d in dmg_preds if d['confidence'] >= 0.5]:
        d_poly = Polygon([(p['x'], p['y']) for p in dmg['points']])
        d_weight = damage_weights.get(dmg['class'], 1)
        if not d_poly.is_valid:
            d_poly = d_poly.buffer(0)
        for loc in [l for l in loc_preds if l['confidence'] >= 0.6]:
```

```
                l_poly = Polygon([(p['x'], p['y']) for p in loc['
                    points']])
                if not l_poly.is_valid:
                    l_poly = l_poly.buffer(0)
                if d_poly.intersects(l_poly):
                    part = loc['class'].replace('-', ' ').lower()
                    p_weight = part_weights.get(loc['class'], 1)
                    intersect = d_poly.intersection(l_poly)
                    area = round(intersect.area, 2) if intersect.
                        geom_type == 'Polygon' else 0.0
                    max_area = max(d_poly.area, l_poly.area)
                    scaled_area = 1 + (area / max_area) if
                        max_area > 0 else 1
                    score = d_weight * p_weight * scaled_area
                    total_score += score
                    prefix = ''
                    if side.lower() == 'left':
                        prefix = 'left '
                    elif side.lower() == 'right':
                        prefix = 'right '
                    description = (f"{dmg['class']} on {prefix}{
                        part} "
                                   f"(Damage confidence: {dmg['
                                       confidence']:.2f}, "
                                   f"Part confidence: {loc['
                                       confidence']:.2f}, "
                                   f"Area: {area} px , "
                                   f"Scaled Area: {scaled_area:.2
                                       f})")
                    damage_info.append(description)
                    if area > 0:
                        draw.polygon(intersect.exterior.coords,
                            fill=(255, 0, 0, 120))
                    damaged_part = f"{prefix}{part}"
                    if damaged_part not in damaged_parts:
                        damaged_parts.append(damaged_part)
    max_possible = 5 * 5 * len(part_weights)
    normalized_score = (total_score / max_possible) * 10 if
        max_possible else 0
    health_rating = round(10 - normalized_score, 2)
    final_img = Image.alpha_composite(img, overlay).convert("
        RGB")
    processed_path = os.path.join(app.config['
        PROCESSED_FOLDER'], os.path.basename(file_path))
    final_img.save(processed_path, "JPEG")
    result_image_url = url_for('static', filename=f'processed
        /{os.path.basename(processed_path)}')
    return result_image_url, damage_info, health_rating,
        damaged_parts

if __name__ == "__main__":
    app.run(debug=True)
```

Listing 5.1: Flask-based Backend Code for Car Health Inspection Syste

FrontEnd Flask Application Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>AI-Powered Vehicle Maintenance and Diagnostic
        System</title>
    <style>
```

```
        body {
            font-family: 'Arial', sans-serif;
            background: linear-gradient(to right, #1f1f1f,
               #212121);
            color: white;
            padding: 40px;
            margin: 0;
            text-align: center;
        }

        .container {
            max-width: 900px;
            margin: auto;
            background: rgb(9, 18, 72);
            padding: 40px;
            border-radius: 15px;
            box-shadow: 0 15px 30px rgba(0, 0, 0, 0.3);
            margin-top: 20px;
        }

        .damage-card {
            background: #393e46;
            color: #e3eafc;
            border-radius: 12px;
            margin: 25px auto 18px auto;
            padding: 18px 26px 18px 26px;
            max-width: 700px;
            box-shadow: 0 4px 20px rgba(0,0,0,0.15);
            text-align: left;
            font-size: 1.13rem;
            font-weight: 400;
            letter-spacing: 0.02em;
        }

        .progress-container {
            background: #2b2d42;
            border-radius: 8px;
            height: 26px;
            width: 80%;
            max-width: 400px;
            margin: 10px auto 22px auto;
            position: relative;
            box-shadow: 0 2px 10px rgba(0,0,0,0.07);
        }

        .progress-bar {
            background: linear-gradient(90deg, #00b0ff 70%,
               #02cfff 100%);
            height: 100%;
            border-radius: 8px;
            transition: width 0.6s cubic-bezier(.4,2.3,.3,1);
            text-align: right;
            font-weight: 700;
            color: #fff;
            font-size: 1.1rem;
            display: flex;
            align-items: center;
            justify-content: flex-end;
            padding-right: 12px;
        }

        .rating-text {
            position: absolute;
            width: 100%;
            left: 0;
            top: 0;
            height: 100%;
            display: flex;
            align-items: center;
            justify-content: center;
            color: #fff;
            font-weight: bold;
            font-size: 1.14rem;
```

```
            pointer-events: none;
            text-shadow: 1px 2px 4px #222c, 0px 1px 0px #0e1b;
        }

        .final-damaged-parts {
            background: #393e46;
            padding: 20px 30px;
            border-radius: 12px;
            box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
            margin: 30px auto 20px auto;
            max-width: 600px;
            font-size: 1.1rem;
            font-weight: 400;
            color: #e3eafc;
            letter-spacing: 0.03em;
        }

        .final-damaged-parts h2 {
            font-size: 1.75rem;
            color: #00b0ff;
            font-weight: 700;
            margin-bottom: 15px;
            text-transform: uppercase;
            letter-spacing: 1px;
        }

        .final-damaged-parts ul {
            list-style-type: none;
            padding: 0;
        }

        .final-damaged-parts li {
            padding: 8px 15px;
            margin: 8px 0;
            background: #444;
            border-radius: 8px;
            transition: background 0.3s ease, transform 0.3s ease;
        }

        .final-damaged-parts li:hover {
            background: #00b0ff;
            color: #fff;
            transform: translateX(5px);
        }

        .final-damaged-parts li:last-child {
            margin-bottom: 0;
        }

        h1 {
            font-size: 2.5rem;
            color: #00b0ff;
            margin-bottom: 30px;
            font-weight: bold;
            text-transform: uppercase;
        }

        h2 {
            color: #aad;
        }

        button {
            padding: 15px 30px;
            font-size: 18px;
            background-color: #00b0ff;
            color: white;
            border: none;
            border-radius: 8px;
            cursor: pointer;
            transition: background-color 0.3s ease;
            margin-top: 10px;
        }

        button:hover {
            background-color: #008bb2;
```

```
        }
        input[type="file"] {
            padding: 15px;
            font-size: 16px;
            border: 2px solid #ddd;
            border-radius: 8px;
            margin: 10px 0 20px 0;
            background-color: #444;
            width: 100%;
            max-width: 450px;
            color: white;
        }
        input[type="text"] {
            padding: 12px;
            font-size: 17px;
            border: 2px solid #ddd;
            border-radius: 8px;
            margin: 10px 0 18px 0;
            background-color: #444;
            width: 100%;
            max-width: 250px;
            color: white;
        }
        .image-container {
            position: relative;
            display: block;
            margin: 20px 0;
            max-width: 100%;
        }
        .car-image {
            width: 100%;
            height: auto;
            border-radius: 10px;
        }
        svg {
            position: absolute;
            top: 0;
            left: 0;
            pointer-events: none;
            width: 100%;
            height: 100%;
        }
        [class^="highlight-"] {
            stroke: rgba(60, 60, 60, 0.2);
            stroke-width: 1;
            transition: fill 0.3s ease;
            filter: drop-shadow(0 1px 1.5px rgba(0, 0, 0,
               0.05));
            cursor: pointer;
        }
        .highlight-front-bumper          { fill: rgba(70, 130,
            180, 0.25); }
        .highlight-headlight-right       { fill: rgba(70, 130,
            180, 0.22); }
        .highlight-headlight-left        { fill: rgba(70, 130,
            180, 0.22); }
        .highlight-roof                  { fill: rgba(70, 130,
            180, 0.28); }
        .highlight-mirror-right          { fill: rgba(70, 130,
            180, 0.20); }
        .highlight-mirror-left           { fill: rgba(70, 130,
            180, 0.20); }
        .highlight-car-hood              { fill: rgba(70, 130,
            180, 0.25); }
        .highlight-front-windshield      { fill: rgba(70, 130,
            180, 0.18); }
        .highlight-back-windshield       { fill: rgba(70, 130,
            180, 0.18); }
```

```
        .highlight-front-left-tire       { fill: rgba(70, 130,
            180, 0.23); }
        .highlight-front-right-tire      { fill: rgba(70, 130,
            180, 0.23); }
        .highlight-back-left-tire        { fill: rgba(70, 130,
            180, 0.23); }
        .highlight-back-right-tire       { fill: rgba(70, 130,
            180, 0.23); }
        .highlight-rocker-panel-left     { fill: rgba(70, 130,
            180, 0.20); }
        .highlight-rocker-panel-right    { fill: rgba(70, 130,
            180, 0.20); }
        .highlight-front-left-door       { fill: rgba(70, 130,
            180, 0.27); }
        .highlight-front-right-door      { fill: rgba(70, 130,
            180, 0.27); }
        .highlight-back-left-door        { fill: rgba(70, 130,
            180, 0.27); }
        .highlight-back-right-door       { fill: rgba(70, 130,
            180, 0.27); }
        .highlight-quarter-panel-left    { fill: rgba(70, 130,
            180, 0.26); }
        .highlight-quarter-panel-right   { fill: rgba(70, 130,
            180, 0.26); }
        .highlight-back-bumper           { fill: rgba(70, 130,
            180, 0.25); }
        .highlight-trunk                 { fill: rgba(70, 130,
            180, 0.25); }
        .highlight-grille                { fill: rgba(70, 130,
            180, 0.21); }
        .highlight-license-plate         { fill: rgba(70, 130,
            180, 0.19); }
        .highlight-left-tail-light       { fill: rgba(70, 130,
            180, 0.22); }
        .highlight-right-tail-light      { fill: rgba(70, 130,
            180, 0.22); }
        .highlight-fender-left           { fill: rgba(70, 130,
            180, 0.26); }
        .highlight-fender-right          { fill: rgba(70, 130,
            180, 0.26); }
        .highlight-back-window-left      { fill: rgba(70, 130,
            180, 0.18); }
        .highlight-back-window-right     { fill: rgba(70, 130,
            180, 0.18); }
        .highlight-front-window-left     { fill: rgba(70, 130,
            180, 0.18); }
        .highlight-front-window-right    { fill: rgba(70, 130,
            180, 0.18); }
        [class^="highlight-"]:hover {
            fill: rgba(70, 130, 180, 0.45);
        }
    </style>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></
        script>
</head>
<body>
<div class="container">
    <h1> AI-Powered Vehicle Maintenance & Diagnostic System</
        h1>
    <form method="POST" enctype="multipart/form-data">
        <label for="car_id">Enter Car ID:</label>
        <input type="text" id="car_id" name="car_id"
            placeholder="e.g. 42" required><br><br>
        <label for="front">Front View:</label><br>
        <input type="file" id="front" name="front" accept="
            image/*" required><br>
        <label for="back">Back View:</label><br>
        <input type="file" id="back" name="back" accept="
            image/*" required><br>
        <label for="left">Left Side:</label><br>
        <input type="file" id="left" name="left" accept="
            image/*" required><br>
        <label for="right">Right Side:</label><br>
```

```
        <input type="file" id="right" name="right" accept="image/*" required><br>
        <button type="submit">Analyze & Predict</button>
    </form>

    {% if results %}
        {% for res in results %}
            <h2>{{ res.side }} View</h2>
            <div class="image-container">
                <img src="{{ res.image_url }}" class="car-image" alt="{{ res.side }} view processed image">
            </div>
            <div class="damage-card">
                <h4>Detected Damages:</h4>
                {% for info in res.damage_info %}
                    <p>{{ info }}</p>
                {% endfor %}
            </div>
            <div class="progress-container">
                <div class="progress-bar" style="width: {{ res.health_rating * 10 }}%;"></div>
                <div class="rating-text">{{ res.health_rating }}/10</div>
            </div>
            <hr>
        {% endfor %}
            <h2>Overall Damage Summary</h2>
            <div class="damage-card">
                {% for damage in collective_damage %}
                    <p>{{ damage }}</p>
                {% endfor %}
            </div>
            <div class="progress-container">
                <div class="progress-bar" style="width: {{ final_rating_damage * 10 }}%;"></div>
                <div class="rating-text">{{ final_rating_damage }}/10</div>
            </div>
        <div class="final-damaged-parts">
            <h2>Damaged Parts List</h2>
            <ul>
                {% for part in final_damaged_parts %}
                    <li>{{ part }}</li>
                {% endfor %}
            </ul>
        </div>
        <div class="image-container" style="position:relative;">
            <img src="{{ url_for('static', filename='images/report.jpg') }}" class="car-image" alt="Car">
            <svg width="100%" height="auto" viewBox="0 0 626 408">
                {% for part in final_damaged_parts %}
                                {% for part in final_damaged_parts %}
            {% if part == 'back bumper' or part == 'Back bumper' %}
                <polygon class="highlight-back-bumper" style="display: block;" points="
                589,145,573,139,559,138,558,160,558,177,558,196,558,
                " />
                <polygon class="highlight-back-bumper" style="display: block;" points="
                484,80,490,86,487,93,487,104,480,107,463,108,444,109
                " />
                <polygon class="highlight-back-bumper" style="display: block;" points="
                487,302,445,297,434,328,466,328,486,330,490,320
                " />
            {% elif part == 'back door' or part == 'Back door' or part == 'left back door' %}
```

```
311                    <polygon class="highlight-back-left-door"
                          style="display: block;" points="
                          394,52,399,73,387,83,379,99,379,105,325,104,327,56
                          " />
312                {% elif part == 'back door' or part == 'Back door
                    ' or part == 'right back door' %}
313                    <polygon class="highlight-back-right-door"
                          style="display: block;" points="
                          326,303,329,351,394,354,399,333,386,321,379,304
                          " />
314                {% elif part == 'back wheel' or part == 'Back
                    wheel' or part == 'left back wheel' %}
315                    <circle class="highlight-back-left-tire"
                          style="display: block;" cx="412" cy="105"
                          r="26" />
316                    <polygon class="highlight-back-left-tire"
                          style="display: block;" points="
                          585,255,606,256,604,269,586,270,574,271,588,263
                          " />
317                {% elif part == 'back wheel' or part == 'Back
                    wheel' or part == 'right back wheel' %}
318                    <circle class="highlight-back-right-tire"
                          style="display: block;" cx="413" cy="302"
                          r="25" />
319                    <polygon class="highlight-back-right-tire"
                          style="display: block;" points="
                          606,138,572,138,584,141,587,147,587,153,600,155,607,
                          " />
320                {% elif part == 'back window' or part == 'Back
                    window' or part == 'left back window' %}
321                    <polygon class="highlight-back-window-left"
                          style="display: block;" points="
                          382,28,395,53,331,55,336,23" />
322                    <polygon class="highlight-back-window-left"
                          style="display: block;" points="
                          397,261,381,248,339,249,333,264" />
323                {% elif part == 'back window' or part == 'Back
                    window' or part == 'right back window' %}
324                    <polygon class="highlight-back-window-right"
                          style="display: block;" points="
                          394,354,333,354,337,384,357,384,381,380"
                          />
325                    <polygon class="highlight-back-window-right"
                          style="display: block;" points="
                          397,146,333,145,339,160,371,161,384,160"
                          />
326                {% elif part == 'back windshield' or part == '
                    Back windshield' %}
327                    <polygon class="highlight-back-windshield"
                          style="display: block;" points="
                          388,21,423,33,444,44,450,47,432,47,411,34,391,27,383
                          " />
328                    <polygon class="highlight-back-windshield"
                          style="display: block;" points="
                          439,156,448,174,451,187,451,200,451,214,452,228,444,
                          " />
329                    <polygon class="highlight-back-windshield"
                          style="display: block;" points="
                          450,361,436,357,418,369,399,378,386,383,393,386,427,
                          " />
330                    <polygon class="highlight-back-windshield"
                          style="display: block;" points="
                          526,157,502,166,503,242,525,250" />
331                {% elif part == 'fender' or part == 'Fender' or
                    part == 'left fender' %}
332                    <polygon class="highlight-fender-left" style=
                          "display: block;" points="
                          242,58,234,57,209,61,175,66,156,74,165,78,159,82,143
                          " />
333                {% elif part == 'fender' or part == 'Fender' or
                    part == 'right fender' %}
```

```
                <polygon class="highlight-fender-right" style=
                  "display: block;" points="
                  243,303,240,325,243,347,230,348,203,344,178,340,156,3
                  " />
              {% elif part == 'front bumper' or part == 'Front
                 bumper' %}
                <polygon class="highlight-front-bumper" style
                  ="display: block;" points="
                  64,137,62,146,63,155,62,172,60,181,60,195,60,212,59,
                  " />
                <polygon class="highlight-front-bumper" style
                  ="display: block;" points="
                  171,290,169,300,171,311,179,324,142,322,132,318,134,
                  " />
                <polygon class="highlight-front-bumper" style
                  ="display: block;" points="
                  179,85,140,82,134,89,131,100,134,108,145,113,162,114
                  " />
             {% elif part == 'front door' or part == 'left
                front door' %}
                <polygon class="highlight-front-left-door"
                  style="display: block;" points="
                  328,55,326,105,240,105,242,59" />
              {% elif part == 'front door' or part == 'right
                 front door' %}
                <polygon class="highlight-front-right-door"
                  style="display: block;" points="
                  326,303,329,351,243,348,240,326,241,303"
                  />
              {% elif part == 'front wheel' or part == 'left
                 front wheel' %}
                <circle class="highlight-front-left-tire"
                  style="display: block;" cx="203" cy="104"
                  r="26" />
                <polygon class="highlight-front-left-tire"
                  style="display: block;" points="
                  29,256,16,256,15,270,50,271,31,264" />
              {% elif part == 'front wheel' or part == 'right
                 front wheel' %}
                <circle class="highlight-front-right-tire"
                  style="display: block;" cx="204" cy="302"
                  r="25" />
                <polygon class="highlight-front-right-tire"
                  style="display: block;" points="
                  51,136,15,136,16,152,30,153,32,140" />
              {% elif part == 'front window' or part == 'left
                 front window' %}
                <polygon class="highlight-front-window-left"
                  style="display: block;" points="
                  332,25,296,26,262,43,254,57,280,57,307,55,324,55
                  " />
                <polygon class="highlight-front-window-left"
                  style="display: block;" points="
                  334,248,303,249,280,254,249,263,272,265,295,265,317,
                  " />
              {% elif part == 'front window' or part == 'right
                 front window' %}
                <polygon class="highlight-front-window-right"
                   style="display: block;" points="
                  324,352,332,383,294,381,263,364,244,349"
                  />
                <polygon class="highlight-front-window-right"
                   style="display: block;" points="
                  327,143,333,160,306,158,276,154,247,143"
                  />
              {% elif part == 'grille' or part == 'Grille' %}
                <polygon class="highlight-grille" style="
                  display: block;" points="
                  72,176,74,209,70,231,60,227,59,188,58,178"
                   />
              {% elif part == 'headlight' or part == 'Headlight
                 ' %}
```

```
                    <polygon class="highlight-headlight-left"
                       style="display: block;" points="
                       67,233,61,237,61,245,61,251,63,257,69,261,73,259,73,
                       " />
                    <polygon class="highlight-headlight-left"
                       style="display: block;" points="
                       163,74,154,72,147,74,141,77,140,83,147,85,155,83,163
                       " />
                    <polygon class="highlight-headlight-right"
                       style="display: block;" points="
                       73,146,65,148,63,154,63,164,62,172,67,175,72,170,73,
                       " />
                    <polygon class="highlight-headlight-right"
                       style="display: block;" points="
                       165,327,159,325,151,325,143,324,144,331,151,333,165,
                       " />
                {% elif part == 'hood' or part == 'Hood' %}
                    <polygon class="highlight-car-hood" style="
                       display: block;" points="
                       234,146,223,162,220,181,216,197,216,209,219,226,222,
                       " />
                    <polygon class="highlight-car-hood" style="
                       display: block;" points="
                       91,148,93,168,92,191,92,207,93,224,92,240,89,259,74,
                       " />
                    <polygon class="highlight-car-hood" style="
                       display: block;" points="
                       232,56,220,52,207,55,187,57,170,60,155,64,144,67,148
                       " />
                    <polygon class="highlight-car-hood" style="
                       display: block;" points="
                       234,348,210,345,196,344,182,342,167,338,154,333,142,
                       " />
                {% elif part == 'License-plate' or part == '
                   license plate' or part == 'License plate' %}
                    <polygon class="highlight-license-plate"
                       style="display: block;" points="
                       550,180,550,230,544,230,538,230,536,207,536,185,538,
                       " />
                    <polygon class="highlight-license-plate"
                       style="display: block;" points="
                       55,183,56,223,46,224,46,184" />
                {% elif part == 'Mirror' or part == 'right mirror
                   ' or part == 'left mirror' %}
                    <rect class="highlight-mirror-left" style="
                       display: block;" x="90" y="262" width="10"
                        height="16" />
                    <rect class="highlight-mirror-left" style="
                       display: block;" x="268" y="47" width="12"
                        height="10" />
                    <rect class="highlight-mirror-left" style="
                       display: block;" x="522" y="258" width="11
                       " height="16" />
                    <polygon class="highlight-mirror-left" style=
                       "display: block;" points="
                       264,259,249,262,258,266,260,273,268,275"
                       />
                    <rect class="highlight-mirror-right" style="
                       display: block;" x="90" y="131" width="9"
                       height="15" />
                    <polygon class="highlight-mirror-right" style
                       ="display: block;" points="
                       268,133,261,135,258,140,251,144,260,146,264,150
                       " />
                    <rect class="highlight-mirror-right" style="
                       display: block;" x="256" y="351" width="13
                       " height="10" />
                    <rect class="highlight-mirror-right" style="
                       display: block;" x="524" y="130" width="10
                       " height="15" />
                {% elif part == 'quarter panel' or part == 'left
                   quarter panel' %}
```

```
                <polygon class="highlight-quarter-panel-left"
                    style="display: block;" points="
                    475,51,476,60,467,64,473,68,479,70,480,79,454,78,431
                    " />
            {% elif part == 'quarter panel' or part == 'right
                quarter panel' %}
                <polygon class="highlight-quarter-panel-right
                    " style="display: block;" points="
                    480,329,434,329,419,335,399,332,394,356,413,356,406,
                    " />
            {% elif part == 'rocker panel' or part == 'left
                rocker panel'  %}
                <rect class="highlight-rocker-panel-left"
                    style="display: block;" x="235" y="105"
                    width="150" height="8" />
            {% elif part == 'rocker panel' or part == 'right
                rocker panel' %}
                <rect class="highlight-rocker-panel-right"
                    style="display: block;" x="236" y="294"
                    width="147" height="9" />
            {% elif part == 'roof' %}
                <polygon class="highlight-roof" style="
                    display: block;" points="
                    393,165,371,166,344,164,321,164,311,164,291,161,287,
                    " />
                <polygon class="highlight-roof" style="
                    display: block;" points="
                    385,382,371,384,351,385,326,384,308,384,295,381,285,
                    " />
                <polygon class="highlight-roof" style="
                    display: block;" points="
                    119,164,120,175,120,182,120,198,121,212,122,227,120,
                    " />
                <polygon class="highlight-roof" style="
                    display: block;" points="
                    387,20,374,18,360,19,345,17,328,18,308,19,291,20,284
                    " />
                <polygon class="highlight-roof" style="
                    display: block;" points="
                    499,162,498,175,498,189,498,202,498,213,498,225,498,
                    " />
            {% elif part == 'tail light' or part == 'right
                tail light' or part == 'left tail light' %}
                <polygon class="highlight-left-tail-light"
                    style="display: block;" points="
                    482,62,466,64,478,70,485,75" />
                <polygon class="highlight-left-tail-light"
                    style="display: block;" points="
                    545,237,540,252,538,264,543,267,547,261,550,254,550,
                    " />
                <polygon class="highlight-right-tail-light"
                    style="display: block;" points="
                    483,335,468,342,472,347,482,346" />
                <polygon class="highlight-right-tail-light"
                    style="display: block;" points="
                    544,141,550,147,551,162,546,168,540,151,540,145
                    " />
            {% elif part == 'Trunk' or part == 'trunk' %}
                <polygon class="highlight-trunk" style="
                    display: block;" points="
                    557,167,550,164,546,170,538,148,527,158,525,178,523,
                    " />
                <polygon class="highlight-trunk" style="
                    display: block;" points="
                    480,156,439,157,450,183,452,208,447,238,438,249,479,
                    " />
                <polygon class="highlight-trunk" style="
                    display: block;" points="
                    480,356,472,354,438,358,451,362,468,361"
                    />
                <polygon class="highlight-trunk" style="
                    display: block;" points="
```

```
                    479,49,449,46,438,49,468,52" />
            {% elif part == 'Windshield' or part == '
               windshield' %}
                <polygon class="highlight-front-windshield"
                    style="display: block;" points="
                    290,160,284,181,283,199,284,214,286,231,291,247,273,
                    " />
                <polygon class="highlight-front-windshield"
                    style="display: block;" points="
                    288,26,282,23,254,38,225,55,237,59,260,40"
                     />
                <polygon class="highlight-front-windshield"
                    style="display: block;" points="
                    237,349,264,367,289,381,281,383,255,372,234,360,223,
                    " />
                <polygon class="highlight-front-windshield"
                    style="display: block;" points="
                    122,164,120,175,122,190,122,204,122,219,121,231,122,
                    " />
            {% endif %}
        {% endfor %}
                {% endfor %}
        </svg>
        </div>

    {% endif %}

    {% if chart_data %}
        <h2 style="color:#00b0ff;">OBD Data Analysis (Car ID:
            {{ car_id_input }})</h2>
        <div class="chart-container">
            {% for label, cd in chart_data.items() %}
                <h3 style="color:#aad;">{{ label }} vs RPM</
                    h3>
                <canvas id="chart-{{ loop.index0 }}" height="
                    200"></canvas>
            {% endfor %}
        </div>
        <div class="summary">
            <p> Normal: <strong>{{ normal_count }}</strong>
                Non-normal: <strong>{{ non_normal_count }}</
                strong></p>
            <p> Rating: <strong>{{ rating_out_of_10 }}/10</
                strong></p>
            <h3>{{ final_rating }}</h3>
        </div>
    {% elif obd_result %}
        <div class="result">
            <h3>{{ obd_result }}</h3>
        </div>
    {% endif %}

</div>


{% if chart_data %}
<script>
    const palette = [
      '#FF6384', '#36A2EB', '#FFCE56',
      '#4BC0C0', '#9966FF', '#FF9F40'
    ];
    const chartData = {{ chart_data|tojson }};
    Object.keys(chartData).forEach((label, idx) => {
      const ctx = document.getElementById(`chart-${idx}`).
          getContext('2d');
      const color = palette[idx % palette.length];
      new Chart(ctx, {
        type: 'line',
        data: {
          labels: chartData[label].labels,
          datasets: [{
            label: label,
            data: chartData[label].data,
            fill: false,
            tension: 0.4,
```

```
                pointRadius: 5,
                borderWidth: 2,
                borderColor: color,
                backgroundColor: color,
                pointBackgroundColor: color,
                pointBorderColor: '#ffffff'
              }]
            },
            options: {
              animation: { duration: 2000, easing: '
                 easeInOutQuart' },
              scales: {
                x: { title: { display: true, text: 'RPM', color:
                   '#eee' }, ticks: { color: '#eee' } },
                y: { title: { display: true, text: label, color:
                   '#eee' }, ticks: { color: '#eee' } }
              },
              plugins: { legend: { labels: { color: '#eee' } } }
            }
          });
        });
</script>
{% endif %}


</body>
</html>
```

Listing 5.2: FrontEnd Flask Application Code

AI-BASED VEHICLE HEALTH MANAGEMENT SYSTEM

by Muhammad Mughees Ul Haq

Submission date: 25-Jun-2025 10:41AM (UTC+0500)
Submission ID: 2705724979
File name: FYP_Final.pdf (11.82M)
Word count: 16140
Character count: 93744

AI-BASED VEHICLE HEALTH MANAGEMENT SYSTEM

ORIGINALITY REPORT

9%	5%	4%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Higher Education Commission Pakistan Student Paper	3%
2	R. N. V. Jagan Mohan, B. H. V. S. Rama Krishnam Raju, V. Chandra Sekhar, T. V. K. P. Prasad. "Algorithms in Advanced Artificial Intelligence - Proceedings of International Conference on Algorithms in Advanced Artificial Intelligence (ICAAAI-2024)", CRC Press, 2025 Publication	<1%
3	Submitted to University of Northampton Student Paper	<1%
4	huggingface.co Internet Source	<1%
5	Submitted to University of Hertfordshire Student Paper	<1%
6	www.subframe.com Internet Source	<1%
7	"Proceedings of the 5th International Conference on Big Data Analytics for Cyber-Physical System in Smart City—Volume 1", Springer Science and Business Media LLC, 2025 Publication	<1%

8	Onofre Martorell, Antoni Buades, Jose Luis Lisani. "Multiscale Detection of Circles, Ellipses and Line Segments, Robust to Noise and Blur", IEEE Access, 2021 Publication	<1 %
9	Submitted to Middlesex University Student Paper	<1 %
10	Submitted to University of Glasgow Student Paper	<1 %
11	Thangaprakash Sengodan, Sanjay Misra, M Murugappan. "Advances in Electrical and Computer Technologies", CRC Press, 2025 Publication	<1 %
12	fastercapital.com Internet Source	<1 %
13	Submitted to Liverpool John Moores University Student Paper	<1 %
14	Submitted to Oxford Brookes University Student Paper	<1 %
15	Furkan Kinli, Baris Ozcan, Furkan Kirac. "Patch-wise Contrastive Style Learning for Instagram Filter Removal", 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2022 Publication	<1 %
16	"Pattern Recognition and Computer Vision", Springer Science and Business Media LLC, 2019 Publication	<1 %

17	Submitted to International Islamic University Malaysia Student Paper	<1 %
18	Submitted to Baylor University Student Paper	<1 %
19	Submitted to Tilburg University Student Paper	<1 %
20	unpkg.com Internet Source	<1 %
21	Submitted to FH Campus Wien Student Paper	<1 %
22	Poonam Nandal, Mamta Dahiya, Meeta Singh, Arvind Dagur, Brijesh Kumar. "Progressive Computational Intelligence, Information Technology and Networking", CRC Press, 2025 Publication	<1 %
23	www.mdpi.com Internet Source	<1 %
24	Submitted to CSU, San Jose State University Student Paper	<1 %
25	Submitted to City University Student Paper	<1 %
26	Submitted to Faculty of Organization and Informatics Student Paper	<1 %
27	Submitted to NCC Education Student Paper	<1 %
28	Submitted to University of Northumbria at Newcastle Student Paper	<1 %

29	universe.roboflow.com Internet Source	<1%
30	Submitted to BB9.1 PROD Student Paper	<1%
31	Wenhao Ding, Shuaijun Li, Guilin Zhang, Xiangyu Lei, Huihuan Qian. "Vehicle Pose and Shape Estimation Through Multiple Monocular Vision", 2018 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2018 Publication	<1%
32	Submitted to OCR Student Paper	<1%
33	dig.watch Internet Source	<1%
34	eprints.utm.my Internet Source	<1%
35	"Intelligent Computing and Optimization", Springer Science and Business Media LLC, 2024 Publication	<1%
36	Submitted to Keimyung University Student Paper	<1%
37	git.etud.insa-toulouse.fr Internet Source	<1%
38	Submitted to University Der Es Salaam Student Paper	<1%
39	arxiv.org Internet Source	<1%

40	vriendvandeshow.nl Internet Source	<1 %
41	www.lib.umd.edu Internet Source	<1 %
42	Clash, Shelaniece Qiaunna. "Cyber Reliability Analysis of Autonomous Systems", Morgan State University, 2025 Publication	<1 %
43	Mario Rašić, Mario Tropčić, Pjetra Karlović, Dragana Gabrić, Marko Subašić, Predrag Knežević. "Detection and Segmentation of Radiolucent Lesions in the Lower Jaw on Panoramic Radiographs Using Deep Neural Networks", Medicina, 2023 Publication	<1 %
44	Submitted to Woking College Student Paper	<1 %
45	academic-accelerator.com Internet Source	<1 %
46	docplayer.net Internet Source	<1 %
47	ebin.pub Internet Source	<1 %
48	www.geeksforgeeks.org Internet Source	<1 %
49	2ch.hk Internet Source	<1 %
50	Aleksander Pedersen, Tanita F. Brustad. "A Study on Hybrid Sensor Technology in Winter Road Assessment", Safety, 2020	<1 %

	Publication	
51	Antonella Santone, Francesco Mercaldo, Luca Brunese. "A Method for Real-Time Lung Nodule Instance Segmentation Using Deep Learning", Life, 2024 Publication	<1 %
52	Submitted to Middle East College Student Paper	<1 %
53	Qawasmeh, Saif Al-Dean Rushdi. "Using Machine Learning to Detect Insider Threats; Real-Time Behavior Analysis and Risk Assessment.", North Carolina Agricultural and Technical State University Publication	<1 %
54	blog.garagepro.in Internet Source	<1 %
55	deepblue.lib.umich.edu Internet Source	<1 %
56	idr.mnit.ac.in Internet Source	<1 %
57	itegam-jetia.org Internet Source	<1 %
58	lib.dr.iastate.edu Internet Source	<1 %
59	manualzz.com Internet Source	<1 %
60	nrch.com.au Internet Source	<1 %
61	Submitted to smci Student Paper	<1 %

62	www.hybrid-analysis.com Internet Source	<1 %
63	Arman Neyestani, Imran Ahmed, Pasquale Daponte, Luca De Vito. "Concrete Crack Detection and Segmentation in Civil Infrastructures Using UAVs and Deep Learning", 2023 7th International Conference on Internet of Things and Applications (IoT), 2023 Publication	<1 %
64	Submitted to University of Portsmouth Student Paper	<1 %
65	blog.roboflow.com Internet Source	<1 %
66	Sufyan bin Uzayr. "Mastering HTML - A Beginner's Guide", CRC Press, 2023 Publication	<1 %
67	www.seo.com Internet Source	<1 %
68	git.xpub.nl Internet Source	<1 %
69	idegeo.centrogeo.org.mx Internet Source	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		