RESEARCH PROJECT REPORT – 1

ON

ALERTME : ACCIDENT DETECTION AND REAL TIME ALERT SYSTEM

A Research Project Report submitted in partial fulfilment of the requirements for the degree of Bachelor of Technology in Information Technology

Submitted by

G.KHUSHI	**(22RH1A1266)**
K.TEJA SREE	**(22RH1A1278)**
K.ANISHWA	**(22RH1A1285)**

Under the Esteemed Guidance of
Mrs. K. GEETHA PRATHIBHA
Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY ENGINEERING COLLEGE FOR WOMEN

Autonomous Institution, UGC, Govt. of India
Programmes Accredited by NBA, Accredited by NAAC with A+ Grade, Govt. of India.
Affiliated to JNTUH, Approved by AICTE, ISO 9001:2015 Certified Institute.
National Ranking by NIRF Innovation-Rank Band(151-300), AAAA Rated by career 360 Magazine.
AAAA+ Rated by Digital Learning Magazine, 12th Top Engineering College of Super Band-Excellent
CSR-2023. Green Ranking "Gold Band" Sustainable Institution of India.
Maisammaguda (V), Dhullapally (Post), (Via) Kompally, Medchal Malkajgiri Dist. T.S-500100
www.mallareddyecw.com

July - 2025

MALLA REDDY ENGINEERING COLLEGE FOR WOMEN
Autonomous Institution, UGC, Govt. of India
Programmes Accredited by NBA, Accredited by NAAC with A+ Grade
Affiliated to JNTUH, Approved by AICTE, ISO 9001:2015 Certified Institute
Maisammaguda (V), Dhullapally (Post), (Via) Kompally, Medchal Malkajgiri Dist. T.S-500100

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the Research Project - 1 embodies in this dissertation entitled **"ALERTME: ACCIDENT DETECTION AND REAL TIME ALERT SYSTEM"** being submitted by **"G. Khushi (22RH1A1266), K. Teja Sree (22RH1A1278), K. Anishwa (22RH1A1285)"** for partial fulfilment of the requirement for the award of **BACHELOR OF TECHNOLOGY** in **Information Technology**, **Malla Reddy Engineering College for Women (Autonomous)**, **Maisammaguda, Secunderabad** during the academic year **2025 - 2026**.

Guide

MRS.K .GEETHA PRATHIBHA
ASSISTANT PROFESSOR

Head of the Department

DR. V. YASASWINI
ASSISTANT PROFESSOR
DEPARTMENT OF IT

External Examiner

MALLA REDDY ENGINEERING COLLEGE FOR WOMEN
Autonomous Institution, UGC, Govt. of India
Programmes Accredited by NBA, Accredited by NAAC with A+ Grade
Affiliated to JNTUH, Approved by AICTE, ISO 9001:2015 Certified Institute
Maisammaguda (V), Dhullapally (Post), (Via) Kompally, Medchal Malkajgiri Dist. T.S-500100

DEPARTMENT OF INFORMATION TECHNOLOGY

DECLARATION

We "**G. KHUSHI (22RH1A1266), K. TEJA SREE (22RH1A1278), K. ANISHWA (22RH1A1285)**" are students of 'Bachelor of Technology in Information Technology', Malla Reddy Engineering College for Women (Autonomous), Maisammaguda, Secunderabad, hereby declare that the work presented in this Project Work entitled "**ALERTME: ACCIDENT DETECTION AND REAL TIME ALERT SYSTEM**" is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Date:

G.KHUSHI	**(22RH1A1266)**
K.TEJA SREE	**(22RH1A1278)**
K.ANISHWA	**(22RH1A1285)**

ACKNOWLEDGEMENT

We feel ourselves honoured and privileged to place our warm salutation to our college **Malla Reddy Engineering College for Women** and Department of **Information Technology** which gave us the opportunity to have expertise in engineering and profound technical knowledge.

We would like to deeply thank our Honourable Member of Legislative Assembly of Telangana, **SRI. CH. MALLA REDDY Garu**, founder chairman MRGI, the largest cluster of institutions in the state of Telangana for providing us with all the resources in the college to make our product success.

We wish to convey gratitude to our **Principal Dr. Y. MADHAVEE LATHA**, for providing us with the environment and mean to enrich our skills and motivating us in our endeavour and helping us to realize our full potential.

We express our sincere gratitude to **Dr. V. YASASWINI, Assistant Professor & Head, Department of Information Technology** for inspiring us to take up a product on this subject and successfully guiding us towards its completion.

We would like to thank our internal guide **Mrs. K. Geetha Prathibha, Assistant Professor** and all the faculty members for their valuable guidance and encouragement towards the completion of our product work.

With regards and Gratitude

G.KHUSHI (22RH1A1266)

K.TEJA SREE (22RH1A1278)

K.ANISHWA (22RH1A1285)

ABSTRACT

ALERTME: ACCIDENT DETECTION AND REAL TIME ALERT SYSTEM

The proposed AlertMe: Accident Detection and Real-Time Alert System is a deep learning-based solution designed to automatically detect road traffic accidents and provide instant alerts in smart city environments. Unlike traditional systems that rely on manual monitoring, AlertMe analyzes visual and movement patterns from traffic video feeds to identify various types of accidents, including rear-end collisions and T-bone crashes. The system leverages trained models to process live or recorded footage, enabling accurate detection without human intervention. A key feature of the solution is its real-time alert mechanism, which notifies drivers, commuters, and traffic control authorities about potential dangers or accident-prone zones, supporting faster emergency response. Designed to be lightweight and efficient, AlertMe can be deployed on compact, city-level hardware and easily integrated with existing traffic infrastructure. By combining automated accident monitoring with immediate notifications, the system aims to enhance road safety, reduce response times, and contribute to smarter, safer urban mobility. Future improvements include integration with live public camera feeds, GPS tracking, and weather-based alert customization. This scalable and modular design makes AlertMe suitable for diverse deployment scenarios and a valuable tool for smart city development, offering a promising approach to reducing the impact of road traffic accidents.

INDEX

Chapter 1
Introduction

Road safety has emerged as a critical issue in the 21st century, especially as rapid urbanization and a sharp increase in vehicle usage have led to a surge in road traffic accidents. With the expansion of urban centers and the rising demand for mobility, traffic congestion and road-related incidents have become common occurrences, posing severe threats to human life and public health. According to the World Health Organization (WHO), approximately 1.35 million people die every year as a result of road traffic crashes, while millions more suffer non-fatal injuries that often result in long-term disabilities. These alarming figures highlight the urgent need for effective solutions that can mitigate the devastating impact of road accidents on individuals, families, and communities.

Current accident detection and reporting systems are largely reactive. In most cases, they rely on human witnesses or bystanders to report incidents to emergency services. This often introduces delays during the crucial moments immediately following an accident, when timely intervention can make a significant difference in preventing fatalities or minimizing injury severity. Although advancements in technology have improved various aspects of road safety, a majority of existing systems still depend on post-incident reporting or passive monitoring. This situation underscores the need for intelligent, automated systems capable of detecting accidents in real time and instantly alerting relevant stakeholders without human intervention.

To address these challenges, the proposed system, titled AlertMe: Accident Detection and Real-Time Alerts, aims to offer a comprehensive solution that combines deep learning techniques, video frame analysis, and cloud-based communication services. The goal is to develop a system that not only detects the occurrence of traffic accidents but also sends immediate notifications to emergency contacts or services. The system processes video data—whether uploaded by users or obtained through live streams—and applies trained convolutional neural network (CNN) models to identify accident scenarios with a high degree of accuracy.

These models are trained on large and diverse datasets, including the Car Accident Detection and Prediction (CADP) dataset. The user interface is designed to be simple and accessible to a wide range of users. The system allows individuals to register and log in, with their data stored securely in a JSON file. Once registered, users can upload videos or images of areas where they suspect or have observed an accident. The system then analyzes the submitted media using its trained model and provides a classification indicating whether an accident has occurred. This approach empowers users to contribute to road safety while benefiting from an intelligent system that handles the complex task of accident detection.

While the current version requires users to upload media files for analysis, future versions are envisioned to support automated retrieval of relevant video or image data based on location inputs. Users would be able to simply enter the name of a location or coordinates, and the system would fetch real-time feeds or images from public cameras or map APIs, process them, and deliver accident status updates. This would eliminate the need for manual uploads, making the system more efficient and suitable for integration into larger smart city frameworks.

The inclusion of real-time weather information in alert messages adds another valuable dimension to the system. Emergency responders and contacts receiving these alerts can take into account environmental factors such as rain, fog, or snow, which might influence both the severity of the accident and the approach needed for rescue or support operations.

From a technical standpoint, the system leverages the strengths of deep learning and computer vision technologies. The architecture is designed to be efficient and capable of extension, with possibilities for incorporating additional features such as GPS-based tracking, integration with real-time camera feeds, multilingual alerting systems, and support for mobile and web platforms. These planned extensions aim to ensure that the system remains adaptable and relevant to diverse deployment contexts. By combining artificial intelligence with user-friendly design and cloud-based communication services, the system aims to contribute meaningfully to road safety efforts, reduce response times, and ultimately help save lives in accident scenarios.

Chapter 2
Literature Survey

2.1 Overview of Accident Detection Systems

Accident detection systems play an essential role in improving road safety and reducing emergency response time in traffic incidents. Traditionally, these systems have relied on approaches such as manual reporting by bystanders, deployment of roadside sensors, or analysis of anomalies in vehicle telemetry data (for example, sudden deceleration detected via GPS or accelerometers). While such methods can provide valuable information, they suffer from critical limitations, including delayed reporting, dependency on human intervention, and the need for specialized hardware installations. These systems are often not scalable to large urban settings or rural areas with limited infrastructure.

With the rapid growth of surveillance technologies and the increased availability of video data from sources such as traffic cameras and dashboard cameras, the focus has shifted toward video-based accident detection. Such systems analyze video frames to identify patterns associated with accidents, such as abrupt stops, collisions, or changes in vehicle trajectory. These developments have laid the groundwork for AI-driven solutions capable of real-time accident detection in diverse environments. Our project builds upon this evolution by offering a flexible, software-based solution that processes video input for accident detection and triggers automated alerts, making it adaptable to different deployment scenarios without relying on expensive hardware or fixed installations.

2.2 Deep Learning Approaches for Accident Detection

Recent advances in deep learning have significantly transformed the field of accident detection, offering improvements in accuracy and reliability over traditional machine learning methods. Convolutional Neural Networks (CNNs), in particular, have proven effective in extracting spatial features from image and video data. Architectures such as ResNet, MobileNet, and VGGNet have been widely used to identify accident indicators from individual frames or sequences of frames. These models are capable of recognizing key visual cues such as deformations, collisions, or abnormal postures of vehicles in the scene.

In video-based accident detection, combining spatial and temporal features is critical. While 2D CNNs operate on individual frames and capture spatial information, 3D CNNs and hybrid models like CNN-LSTM can process sequences of frames, learning to identify changes over time that signify accidents. In our project, a CNN-based model trained on the Car Accident Detection and Prediction (CADP) dataset is employed to classify video frames as accident or non-accident scenes. By leveraging this dataset, our system gains exposure to a variety of accident scenarios, improving its robustness. The proposed system focuses on frame-based classification, but future versions may explore temporal models such as 3D CNNs or ConvLSTM for enhanced performance.

2.3 Context-Aware Accident Detection: Environmental Factors

Environmental conditions such as weather, time of day, and location are significant factors that influence both accident occurrence and emergency response effectiveness. Studies have demonstrated strong correlations between adverse weather (e.g., rain, fog, snow) and an increased likelihood of accidents. For example, urban accident rates have been found to rise considerably during rainfall, as shown in research conducted by Jaroszweski et al. Other conditions such as poor visibility, slippery roads, and high humidity further contribute to hazardous driving conditions.

Our system integrates external data from weather APIs to include real-time weather details in accident alerts. This added context not only informs emergency responders of conditions at the scene but also helps improve decision-making in resource allocation and response strategies. By considering these environmental variables, the system offers a more comprehensive and practical solution compared to detection models that focus solely on vehicle dynamics.

2.4 Automated Alert Systems in Accident Response

Existing vehicle-based alert systems like eCall (European Union) and OnStar (General Motors) use embedded sensors within vehicles to automatically notify emergency services when a crash is detected. These systems typically depend on accelerometers, GPS modules, and in-vehicle communication units. While they have proven effective in reducing response times, such solutions are generally limited to newer vehicle models and are not universally accessible, particularly in regions where older vehicles dominate the roads.

Our approach offers a more inclusive and cost-effective alternative by implementing an entirely software-based alerting system. The AlertMe system detects accidents using video analysis and sends immediate notifications to registered users or authorities via email. It uses the SMTP protocol for communication and stores user information in lightweight JSON files, allowing for easy deployment without complex hardware dependencies. This makes the system accessible to a broader range of users, regardless of the type of vehicle or location.

2.5 Video-Based Systems with Minimal Hardware Dependency

A key advantage of video-based accident detection systems is their ability to function without specialized in-vehicle hardware. Traffic cameras, dashcams, and user-submitted media provide rich data sources for accident detection algorithms. Systems based on video analysis are more flexible in terms of deployment and can cover areas beyond the reach of fixed roadside sensors or vehicle telematics. Moreover, video-based systems can be extended to work with existing public or private camera infrastructure, making them cost-effective for urban and suburban environments alike.

AlertMe follows this approach by allowing users to upload videos or images of suspected accident sites through a web interface. The system processes these inputs with its trained deep learning model and generates real-time classifications. The lack of dependency on physical sensors or proprietary hardware makes the system easy to integrate with existing city surveillance networks or even crowd-sourced data from the public.

2.6 User Interface Design for Accident Detection Applications

An intelligent system's effectiveness depends not only on its technical capabilities but also on its usability. Accident detection systems must be designed so that users with minimal technical expertise can operate them effectively. Prior research has shown that overly complex interfaces can discourage use, particularly during emergencies when simplicity and speed are paramount.

To address this, the AlertMe system offers a straightforward web-based interface where users can register, log in, and upload media for accident detection. The design focuses on minimal interaction steps: registration, login, media upload, and result notification. Looking ahead, planned enhancements include enabling users to simply provide a location name, such as "Banjara Hills, Hyderabad," and allowing the system to automatically retrieve relevant images or feeds from online sources like Google Street View or public camera networks. This will further streamline the user experience and reduce the manual effort required.

2.7 Research Gaps and Opportunities Addressed by Our System

Many existing accident detection systems excel in accuracy but suffer from practical limitations related to cost, hardware dependency, or restricted deployment areas. Systems integrated with vehicles often require specialized hardware, making them inaccessible to older vehicles and cost-sensitive regions. Fixed-location systems like those installed on highways or toll booths offer limited coverage and lack flexibility. Moreover, several systems focus solely on accident detection without providing mechanisms for immediate notification or contextual data sharing.

Our project aims to address these gaps through a system that is entirely software-based, capable of operating with minimal hardware, and designed to provide real-time alerts enriched with contextual data such as weather conditions and location details. By combining accident detection, environmental awareness, and a lightweight alerting mechanism in a single solution, the system offers a more complete and practical tool for enhancing road safety.

Chapter 3
Existing System

3.1 Overview of the Existing System

The existing system *Smart City Transportation: Deep Learning Ensemble Approach for Traffic Accident Detection* focuses on improving accident detection in urban environments using video analysis. The aim is to create a solution that can automatically detect accidents from video footage without depending on expensive sensors or human monitoring.

This system is designed to work in smart cities where cameras such as traffic cameras or dashboard cameras are already installed. It analyzes video data using deep learning to detect accidents as they happen. One of the key goals of this system is to make accident detection possible on small, low-cost devices like Raspberry Pi. This means the system can be installed in many places without requiring powerful servers or high-speed internet connections.

The system uses both the video's color frames (RGB) and the motion between frames (optical flow) to identify accidents. This helps the system understand not only what objects are present in the video but also how they are moving, which is important for spotting collisions and sudden changes on the road.

3.2 How the System Works

The system's design combines two powerful deep learning techniques: I3D (Inflated 3D ConvNet) and ConvLSTM2D. The I3D model looks at the video frames and their motion patterns to find useful features. The ConvLSTM2D part helps the system understand the sequence of events over time so it can spot accidents as they unfold.

By using both RGB frames and optical flow data, the system becomes better at detecting accidents. For example, it can notice when a car suddenly stops or when two vehicles collide. The optical flow helps highlight sudden or unusual movements in the video, which are common in accidents.

Importantly, the system is built to run on edge devices. This means it does not need constant connection to the cloud or powerful hardware. Instead, it can work in real-time on small computers installed locally at traffic point

3.3 Datasets Used

Since there were no existing datasets that fully met the needs of this system, the researchers created their own. They developed two datasets:

- **Traffic Camera Dataset:** This dataset contains videos from fixed traffic cameras showing different types of road accidents and normal traffic.
- **Dash Camera Dataset:** This dataset has videos from dashboard cameras inside vehicles, giving a view from the driver's perspective.

These datasets include various types of accidents and different weather and lighting conditions. They help train the model to recognize accidents in many situations. The system also uses transfer learning. This means it starts with a model that already knows general patterns from other video tasks and then teaches it specifically about accidents.

3.4 Results and Strengths

When tested, the system achieved good accuracy in spotting accidents, with a mean average precision (MAP) of 87%. This means it could correctly identify most accidents in the test videos while making fewer mistakes.

Some key strengths of the system are:

- It works on low-cost devices like Raspberry Pi, making it suitable for wide use in smart cities.
- It combines both visual and motion information, improving detection accuracy.
- It uses new, well-prepared datasets to ensure the model sees many types of accidents during training.
- It is designed to be lightweight, so it can process video without needing high-end hardware or internet connections.

3.5 Limitations of the Existing System

While this system is a big improvement over older accident detection methods, it still has some limitations:

- It can sometimes mistake parked or slow-moving vehicles for accidents, especially if visibility is poor or objects block the view.
- Its accuracy can drop in bad weather or low light, such as during heavy rain or at night.
- Although it detects accidents in real-time, it does not include a built-in way to send alerts or notify emergency services automatically.
- It depends on the quality and variety of its training data. If the system is used in a completely different environment from the training data, its performance might not be as good.

These limitations are areas where new systems, like the one we propose, aim to bring improvements. For example, our proposed system includes an alert feature that automatically notifies emergency contacts or services when an accident is detected.

Chapter 4
Proposed System

4.1 Overview

The proposed system, titled **AlertMe: Accident Detection and Real-Time Alerts**, is designed to address key limitations found in existing accident detection solutions. While prior systems, including the deep learning model described in *Smart City Transportation: Deep Learning Ensemble Approach for Traffic Accident Detection*, have demonstrated good accuracy in identifying accidents from video data, they often stop short of delivering real-time alerts to emergency services or stakeholders. Our system enhances these capabilities by introducing a complete pipeline — from accident detection to automated alerting — all while maintaining efficiency and scalability suitable for smart city deployment.

At the core of AlertMe is a **deep learning model trained on the Car Accident Detection and Prediction (CADP) dataset**. This dataset provides a diverse collection of traffic scenes involving accident and non-accident cases, recorded in various environments, weather conditions, and lighting scenarios. By leveraging both **RGB frame analysis** and **optical flow data**, our model achieves accurate detection of road accidents while minimizing false positives caused by environmental noise or stationary vehicles.

In addition to its detection capabilities, AlertMe incorporates an automated **alert system**. Upon identifying an accident, it sends immediate notifications via email to registered emergency contacts. The alert includes important context, such as the **location of the accident** and **real-time weather data**, retrieved through integrated APIs. This ensures that responders have the situational awareness needed to act quickly and appropriately.

The system is designed with user-friendliness in mind. Users can register, log in, and upload videos or images of suspected accident scenes through a simple web interface. The platform processes these inputs and delivers the detection result along with any necessary alerts. Future enhancements will allow users to enter a location name instead of uploading media files, with the system automatically fetching and analyzing relevant real-time data.

4.2 System Architecture

The architecture of AlertMe: Accident Detection and Real-Time Alerts is designed to address the critical need for timely and accurate traffic accident detection in urban environments. As cities continue to expand and vehicle usage increases, conventional traffic monitoring systems struggle to keep pace with the growing volume of data and the demand for rapid emergency response. AlertMe seeks to fill this gap by combining advanced deep learning techniques, computer vision algorithms, cloud-based communication, and user-friendly interfaces into a cohesive, modular framework. This system not only offers a practical solution for accident detection but also serves as a scalable foundation for future smart city safety infrastructures.

The system has been engineered for flexibility and future-proofing. Its modular structure allows individual components to be upgraded or replaced as new technologies emerge, without affecting the stability or functionality of the whole. The architecture supports integration with third-party services such as map APIs, weather services, and real-time traffic feeds. This modularity, combined with efficient processing pipelines, ensures that AlertMe can handle the dynamic and complex nature of urban road safety challenges.

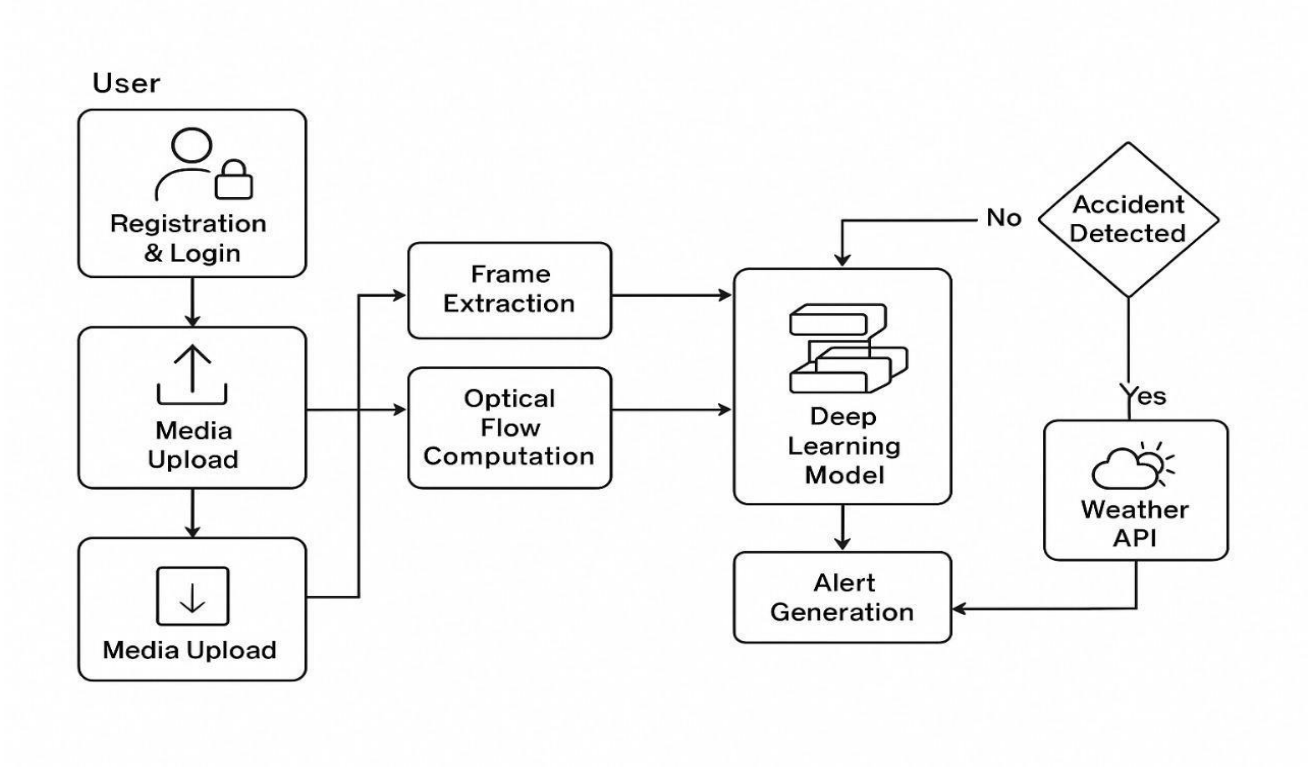


Fig 4.2 System Architecture

4.2.1 High-Level Design

At a high level, AlertMe is composed of several independent yet interconnected modules, each performing a specialized function within the overall system. These modules work together to transform raw user input (such as videos or images) into actionable alerts delivered to emergency contacts in near real-time.
Core Modules:

- **User Interface Module**
  This module provides the front-end web interface through which users interact with the system. It supports user registration, login, media uploads (videos or images), and the display of detection results. The interface has been designed to be responsive and intuitive, ensuring accessibility across a range of devices, including desktops, tablets, and mobile phones.

- **Preprocessing Module**
  Once a media file is uploaded, this module handles the initial processing required to prepare it for analysis. For video files, it extracts RGB frames at defined intervals (e.g., every 5th frame) to reduce computational load while preserving sufficient detail for accurate classification. In addition, it computes optical flow between consecutive frames to capture motion information critical for detecting accidents.

- **Deep Learning Detection Module**
  This module applies a trained Convolutional Neural Network (CNN) to classify the input as either accident or non-accident. The model architecture is optimized for efficiency and accuracy, and it is trained using large, diverse datasets such as the Car Accident Detection and Prediction (CADP) dataset. The model is capable of handling varied traffic conditions, lighting environments, and weather scenarios.

- **Alerting Module**
  If an accident is detected, this module composes a structured alert message that includes the classification result, location information (if provided or extracted), and current weather conditions. The message is sent via email to pre-registered emergency contacts, helping minimize response time during critical situations.

- **Weather API Integration**

  The system uses external weather APIs to retrieve real-time weather data for the specified location. This information is incorporated into alert messages to provide additional context for emergency responders, enabling them to better prepare for on-site conditions.

- **Communication Module**

  This module, designed for future implementation, will support integration with SMS gateways, mobile app notifications, and even IoT-based alert systems for broader communication capabilities.

4.2.2 Detailed Flow

The detailed flow of AlertMe describes the full lifecycle of an accident detection request, from user interaction to notification delivery:

1. User Registration and Login

Users begin by creating an account on the system via a secure web interface. Account data, including usernames, hashed passwords, and associated emergency contacts, are stored in JSON files for lightweight data management. This storage format allows for quick data access and easy scalability in early-stage deployments.

2. Media Upload

Authenticated users can upload media files — videos or images — representing the area where they suspect an accident has occurred. The system validates the file type and size before proceeding, ensuring only supported formats are processed and preventing potential abuse through inappropriate uploads.

3. Frame Extraction

For video uploads, the preprocessing module extracts frames at a specified sampling rate. For example, one frame is captured every five frames. This method balances the need for temporal resolution with the practical limits of processing power, ensuring that the model receives sufficient data without unnecessary computational expense.

4. Optical Flow Computation

The system computes optical flow between each pair of consecutive frames using algorithms like Farnebäck Optical Flow or Dense Optical Flow. Optical flow represents the apparent motion of objects in the scene and is crucial for identifying dynamic patterns characteristic of accidents, such as sudden deceleration, collision impact, or erratic vehicle movement.

5. Feature Extraction

RGB frames and optical flow images together form a rich set of features that represent both the appearance and motion dynamics of the scene. These are fed into the deep learning model for classification.

6. Accident Detection

The CNN processes the input data to classify the event. The model uses a multi-layer architecture with convolutional, pooling, and fully connected layers designed to efficiently capture spatial and temporal patterns. The output is a binary classification: accident or non-accident, along with a confidence score.

7. Alert Generation

If an accident is detected, the alerting module generates an email containing:

- The classification result (accident confirmed)
- Location information, as provided by the user or retrieved from media metadata
- Real-time weather data for the location
- Timestamp of detection

8. Notification Delivery

Using SMTP, the system sends the alert email to the list of emergency contacts associated with the user account. In future implementations, other communication channels such as SMS, push notifications, or direct integration with emergency dispatch systems could be added.

4.2.3 Architectural Diagram

The architectural diagram of the AlertMe system presents a streamlined view of its modular design and data flow. It highlights how each component contributes to accident detection and alert generation in a logical sequence.

The process starts with user registration and login, ensuring authenticated access. The user interface module manages media uploads and initial frame extraction. The preprocessing module further processes the data by performing frame extraction and optical flow computation to capture motion characteristics.

The prepared data is analyzed by the deep learning detection module, where a convolutional neural network classifies the input as accident or non-accident. If an accident is identified, the alerting module generates and delivers notifications containing relevant details. Weather API integration enriches the alert by providing real-time weather conditions for the location.

Arrows in the diagram represent the flow of data between modules, emphasizing a clear, step-by-step process. The modular structure ensures scalability, maintainability, and the flexibility to integrate additional services in future enhancements.

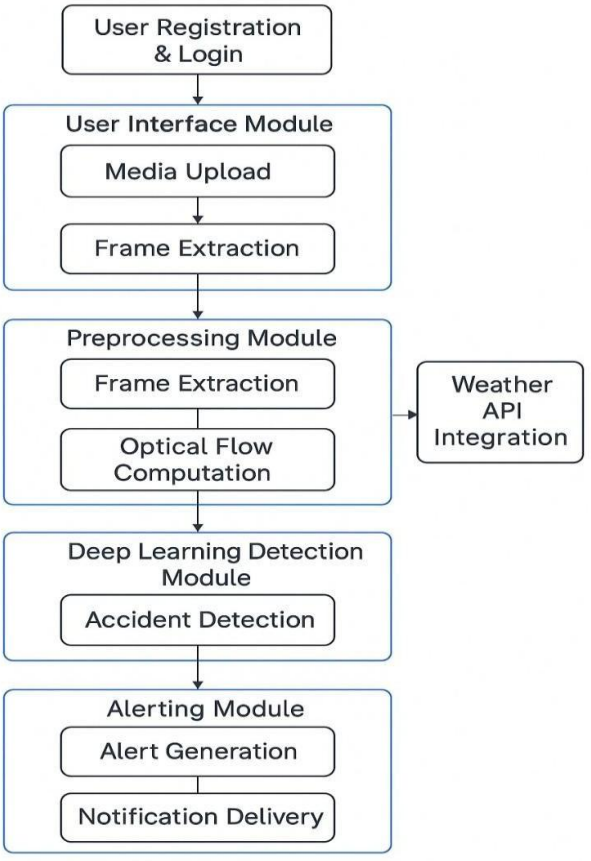


Fig 4.2.3 Architecture Diagram

4.2.4 Design

The design of AlertMe focuses on the following key principles:

- Modularity:

  Each system component is self-contained and interacts with others through well-defined interfaces, allowing for easier debugging, maintenance, and future upgrades.
- Efficiency:

  By sampling frames and computing optical flow selectively, the system reduces processing time while maintaining detection reliability. This makes the solution viable even on resource-constrained environments or edge devices.
- Scalability:

  The architecture can be expanded to handle live video feeds, cloud deployment, and integration with city-wide camera networks. The use of JSON files for data storage in the initial version provides simplicity, while future versions can migrate to databases as needed.
- Extensibility:

  Future enhancements such as multilingual support, mobile app integration, GPS auto-tagging, and AI-driven prediction models can be incorporated without altering the core framework.

4.2.5 UML Diagrams

Use Case Diagram

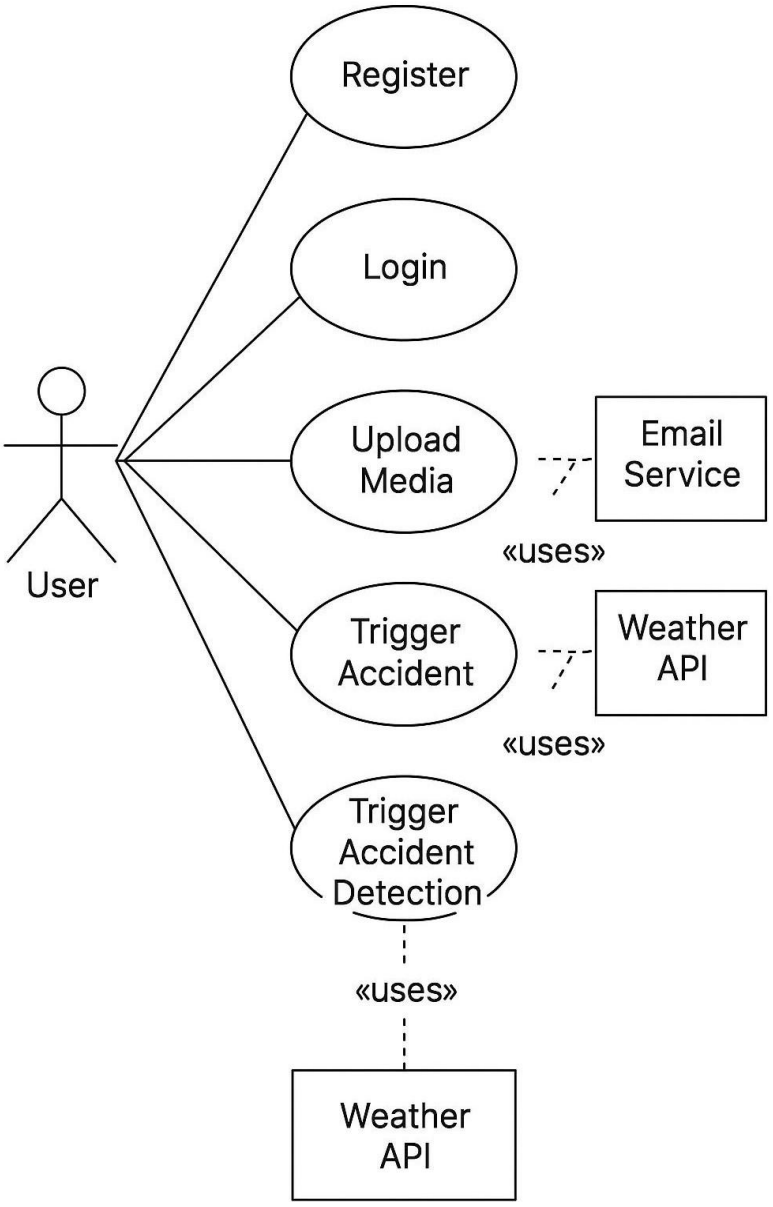


Fig 4.2.5.1 Use Case Diagram

Class Diagram

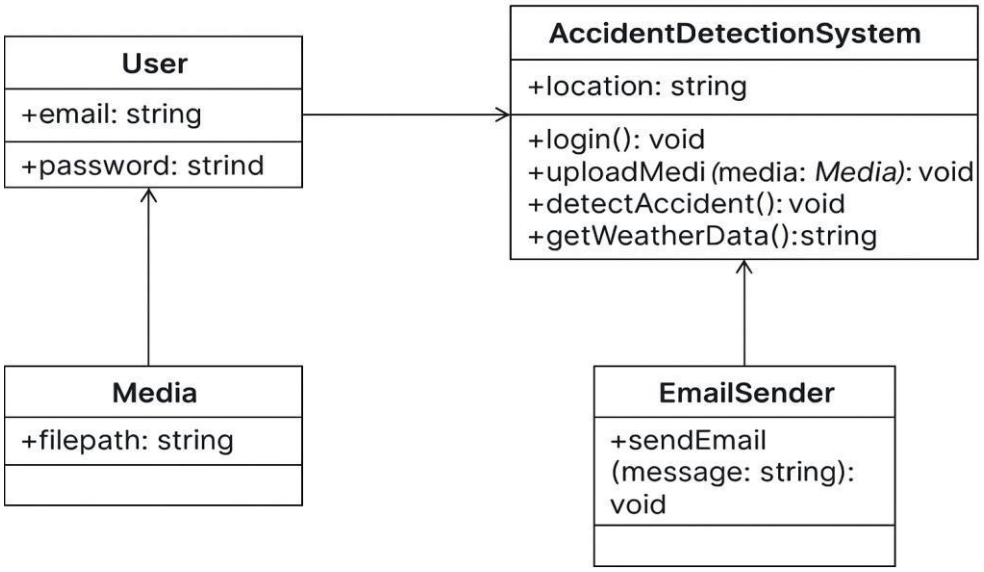


Fig 4.2.5.2 Class Diagram

Component Diagram

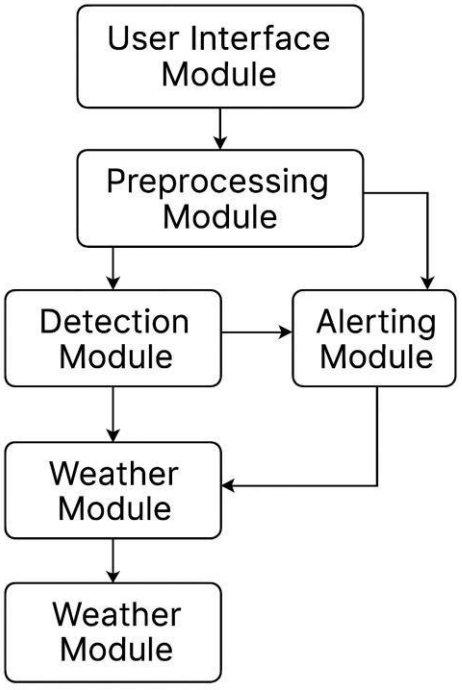


Fig 4.2.5.3 Component Diagram

Sequence Diagram

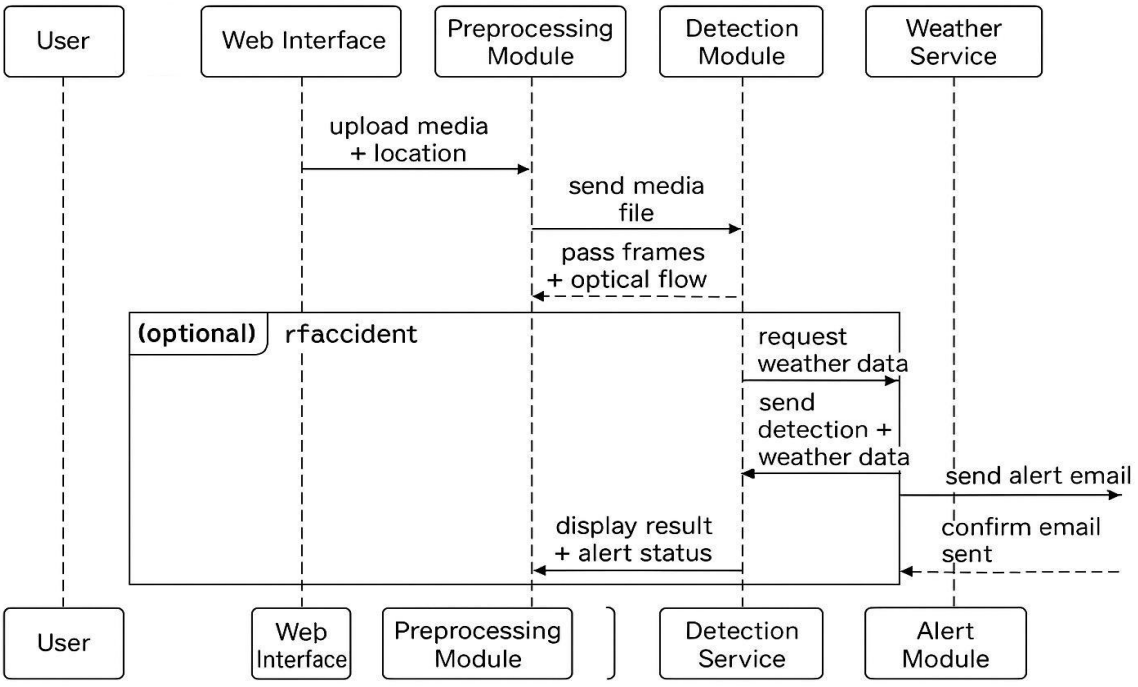


Fig 4.2.5.4 Sequence Diagram

Deployment Diagram

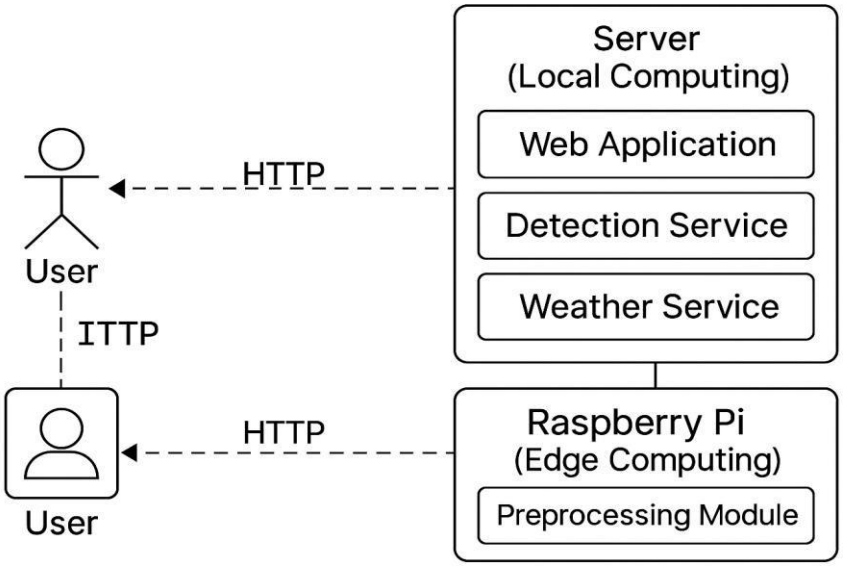


Fig 4.2.5.5 Deployment Diagram

4.2.6 Scalability and Future Enhancements

AlertMe's architecture allows for multiple future enhancements that will improve system functionality and broaden its impact:

- Integration with real-time camera feeds:
  Public CCTV or traffic surveillance cameras can feed directly into the system, enabling automated, continuous monitoring of key traffic points.
- Cloud-based deployment:
  Moving processing to the cloud will enable distributed, large-scale monitoring with centralized management and analytics.
- Mobile platform integration:
  Dedicated mobile applications will allow users to submit reports on-the-go and receive instant notifications.
- GPS and map integration:
  Automatically tagging accident reports with GPS coordinates will enhance accuracy and help responders locate incidents faster.
- Smart city ecosystem compatibility:
  The system can interface with other smart city services, such as emergency response units, traffic management systems, and disaster management frameworks.

.4.3 Proposed Methodology

The proposed methodology outlines the methodology adopted for the development of the AlertMe system. The methodology integrates well-established machine learning practices, computer vision techniques, and cloud communication components. Each phase — from data acquisition to model training and deployment — was designed with the goal of achieving reliable accident detection in real-world traffic scenarios, while ensuring scalability and future extensibility.

4.3.1 Dataset Acquisition and Preparation

A critical foundation of any machine learning-based detection system is the availability of a high-quality, well-annotated dataset. For AlertMe, we utilized the Car Accident Detection and Prediction (CADP) dataset, a comprehensive and widely cited resource for research in this domain.

The CADP dataset was downloaded from its official repository and systematically organized. The videos were sorted into labeled directories corresponding to two categories: accident and non-accident. This clear separation facilitated streamlined preprocessing and ensured that the model could learn to distinguish between normal traffic activity and accident events.
To ensure consistency and compatibility across all samples, the videos were standardized. Each video was converted to a fixed resolution (for example, 640×480 pixels) and a uniform frame rate of 30 frames per second. Standardization eliminates variability caused by different recording devices or sources, ensuring that the model focuses on meaningful patterns rather than artifacts of video capture.

For every video, frames were extracted at a pre-defined sampling interval. This sampling rate was chosen to balance between computational efficiency and temporal resolution, ensuring that significant motion changes were captured without overwhelming the system with redundant frames.

In addition to frame extraction, optical flow was computed for pairs of consecutive frames to capture motion information. These optical flow images, saved either in grayscale or converted to HSV color space, provide valuable insight into vehicle movements and interactions.

To further improve model generalization and prevent overfitting, data augmentation techniques were applied. These included:

- Random horizontal flipping to simulate different camera angles.
- Small rotations to mimic slight variations in camera alignment.
- Brightness and contrast adjustments to account for differing lighting conditions. Such augmentation strategies enrich the training data and help the model perform better in diverse real-world scenarios.

4.3.2 Optical Flow Computation

Understanding motion dynamics is essential for detecting accidents, which typically involve abrupt, unusual movements. To capture these dynamics, dense optical flow was computed using Farnebäck's method. Farnebäck's algorithm produces a dense flow field where each pixel is associated with a motion vector representing its displacement between two consecutive frames. This results in a comprehensive map of motion across the entire scene, rather than just sparse points or features.

These optical flow maps serve multiple purposes:

- They highlight sudden changes in velocity or direction, which are indicative of collisions or rapid deceleration.
- They help the model learn temporal relationships that are not visible from static frames alone.
- They complement RGB frame data by focusing on motion rather than appearance.

By combining optical flow with raw RGB data, the system gains a richer feature set, improving its ability to detect subtle accident patterns even in visually complex environments.

4.3.3 Model Architecture

The architecture of the detection model was carefully designed to strike a balance between accuracy and computational efficiency. It consists of the following components:

- **CNN Backbone:**

  A convolutional neural network, such as ResNet or MobileNet, was employed as the core feature extractor. This network processes both RGB frames and optical flow images to generate deep feature maps representing spatial and motion characteristics.
- **Fusion Layer:**

  Features from the RGB and optical flow streams are combined at a dedicated fusion layer. This layer integrates appearance and motion information, allowing the classifier to make informed decisions based on both what is present in the scene and how objects are moving.
- **Fully Connected Layers:**

  After fusion, the combined features are passed through one or more fully connected layers. These layers learn to map the integrated feature representation to the target classes: accident or non-accident.

Unlike more complex architectures such as I3D or ConvLSTM-based models used in earlier research, this streamlined architecture is designed for deployment on affordable hardware without significant loss of accuracy. It provides a practical solution suitable for real-time or near-real-time applications.

4.3.4 Training Process

The training process aimed to maximize model performance while ensuring generalization to unseen data. Key aspects of the training pipeline included:

- **Loss Function:**

  Cross-entropy loss was used as the objective function, appropriate for binary classification tasks where the model must decide between accident and non-accident.
- **Optimizer:**

  The Adam optimizer was selected due to its ability to adapt learning rates during training, leading to faster convergence. A learning rate decay schedule was applied to fine-tune the model as training progressed.
- **Batch Size:**

  Batch sizes of 16 or 32 were chosen, depending on the memory capacity of the training hardware.

These sizes offer a good compromise between stable gradient estimates and computational efficiency.

- **Validation:**
  A portion of the dataset (typically 10–20%) was reserved for validation. This subset was not seen during training and was used to monitor the model's performance and guard against overfitting.
- **Transfer Learning:**
  To accelerate training and improve accuracy, the CNN backbone was initialized with weights pre-trained on ImageNet. Transfer learning allows the model to leverage generic visual features learned from millions of images, providing a strong starting point for specialized accident detection.

The combination of these strategies ensured a robust training process capable of delivering a reliable, high-performing model.

4.3.5 Alert Generation

The alert generation module is responsible for notifying relevant parties in the event of an accident detection. Upon classification of an input as an accident, the following sequence occurs:

- The location of the incident is retrieved either from user-provided data or, in future versions, from GPS metadata associated with the uploaded media.
- The system queries an external weather API, such as OpenWeatherMap, to obtain real-time weather conditions at the specified location. This context is valuable for responders, as weather may influence the severity of the accident or the required response measures.
- A structured email is generated containing the detection result, location information, timestamp, and current weather conditions. This email is sent automatically via SMTP to the list of emergency contacts associated with the user.

Future enhancements will include support for additional alert channels, including SMS, mobile app push notifications, and integration with emergency dispatch systems.

4.3.6 User Interface

The user interface was designed with simplicity and accessibility in mind, ensuring that users with minimal technical background can effectively interact with the system. Key features include:

- Clean registration and login forms to manage user accounts securely.
- Intuitive upload buttons allowing users to submit videos or images for analysis.
- A results page that displays the detection outcome and indicates whether an alert has been sent.

Future versions of the interface will support location-based querying. Instead of requiring users to upload media, users will be able to input a location name or coordinates. The system will then automatically retrieve live feeds or recent images from that location, process the media, and report on accident status. This feature will further streamline user interaction and improve scalability.

Chapter 5
System Requirements

The proposed AlertMe system provides an end-to-end solution for accident detection and real-time alerting. It integrates deep learning models for video analysis, lightweight storage for user data, and modules for sending automated notifications. The system is modular, so each part can be improved or updated independently, making it flexible and future-proof.

The main components are:

- **User Interface Module:** A simple web platform where users can register, log in, and upload videos or images.
- **Data Preprocessing Module:** Extracts frames from uploaded videos and calculates optical flow to highlight motion patterns.
- **Detection Module:** Uses a trained CNN model to classify input as accident or non-accident.
- **Alerting Module:** Sends email notifications to registered contacts when an accident is detected, including location and weather details.
- **Weather Module:** Uses APIs to get current weather data for the accident location.

Data flow:

1. User logs in and uploads a video or image.
2. Preprocessing extracts frames and computes optical flow.
3. RGB frames and optical flow are passed to the detection module.
4. The detection module classifies the input.
5. If an accident is found, the alerting module sends an email with location, weather, and time.

5.1 Hardware Requirements

The hardware requirements for **AlertMe** depend on where and how it is deployed. The system is designed to work efficiently on moderately powered systems for personal use and on affordable hardware for edge deployment in smart city scenarios.

Minimum Hardware Requirements (Local/Personal Deployment)

- **Processor:** Intel Core i5 (8th generation or newer) or AMD Ryzen 5 equivalent
- **RAM:** 8 GB (16 GB recommended for faster processing)
- **Storage:** 50 GB free disk space (for storing datasets, extracted frames, and logs)
- **GPU:** Optional, but a mid-range NVIDIA GPU (e.g., GTX 1050 Ti or higher) will significantly speed up detection
- **Network:** Stable internet connection for API calls (weather data) and sending email alerts

Hardware for Model Training

For training the CNN model using CADP dataset:

- **Processor:** Intel Core i7 or AMD Ryzen 7
- **RAM:** 32 GB
- **GPU:** NVIDIA RTX 2060 or higher with at least 6 GB VRAM
- **Storage:** 200 GB (to accommodate dataset, checkpoints, and logs)

5.2 Software Requirements

The **AlertMe** system uses widely available, open-source software tools and libraries. The goal is to keep the system accessible, flexible, and easy to deploy across different platforms

.

Operating System

- **Development environment:** Ubuntu 20.04 LTS / Windows 10 or later
- **Edge deployment:** Raspberry Pi OS (32-bit or 64-bit)

Both environments provide compatibility with Python, OpenCV, TensorFlow/PyTorch, and other required tools.

Programming Languages

- **Python 3.7 or higher:** Main programming language for model development, video processing, optical flow computation, and email alert handling.
- **HTML/CSS/JavaScript:** For creating the user interface.

Python is chosen for its extensive support for machine learning libraries and ease of API integration.

Key Libraries and Frameworks

- **OpenCV:** For frame extraction, optical flow computation, and basic image processing.
- **TensorFlow / PyTorch:** For implementing and training the CNN model. Either framework can be used depending on developer preference.
- **NumPy / Pandas:** For numerical computations and data handling.
- **Flask / Django (or lightweight HTTP server):** For serving the web interface.
- **smtplib (Python):** For sending alert emails via SMTP protocol.
- **Requests (Python):** For making API calls to external services (e.g., weather API).

External Services

- **OpenWeatherMap API:** Provides real-time weather data for locations included in alert messages.
- **SMTP (e.g., Gmail SMTP server):** For sending alert emails securely.

Development Tools

- **Jupyter Notebook / VS Code / PyCharm:** For code development and testing.
- **Git:** For version control.
- **Docker (optional):** For containerized deployment, ensuring consistent behavior across environments.

Storage and File Handling

- **JSON files:** Used to store user registration and login data securely in lightweight format.
- **Local file system:** For temporary storage of uploaded media, extracted frames, and generated optical flow images.

Software Flow

1. **User Interface:** HTML forms for registration, login, and upload; Flask/Django handles backend logic.
2. **Processing pipeline:** Python + OpenCV extract frames and compute optical flow.
3. **Model execution:** TensorFlow/PyTorch model classifies scene as accident or not.
4. **Alert generation:** Python scripts send emails using SMTP and fetch weather data using Requests.

Chapter 6
System Design

The design of the **AlertMe** system focuses on creating a practical, scalable, and efficient solution for detecting road accidents and delivering real-time alerts. The system architecture has been broken down into logical components that work together to provide a seamless flow of data — from user interaction to accident detection and alert notification. This section describes the design of each component, the data flow between modules, and how the system ensures both accuracy and efficiency.

6.1 Design Goals

The key design goals of AlertMe are:

- **Accuracy:** To correctly detect accidents with minimal false positives or negatives using deep learning techniques.
- **Efficiency:** To process video inputs quickly enough to support near real-time detection and alerting.
- **Simplicity:** To offer a user-friendly interface so that both technical and non-technical users can use the system easily.
- **Modularity:** To allow independent development and future improvements of components without changing the entire system.
- **Cost-effectiveness:** To keep hardware and deployment costs low so the system can be adopted widely.

In addition to these key design goals, **scalability** is an essential objective of the AlertMe system. The architecture is designed to seamlessly handle increasing volumes of data and a growing number of deployment locations without compromising performance. Whether monitoring traffic in a single intersection or across an entire city, the system ensures consistent reliability and responsiveness. This scalability supports the integration of additional data sources—such as weather sensors or GPS inputs—to enhance the accuracy of accident detection and provide richer contextual information for alerts, making the system future-ready for smart city applications.

6.2 Module-Level Design

6.2.1 User Interface Module

The user interface is designed as a simple web-based platform. Users can register, log in, and upload videos or images for analysis. The design uses standard HTML forms with backend support from Flask (or Django). The key features of the interface include:

- Secure user registration and login using JSON file-based storage.
- Upload form to submit videos or images for processing.
- Display of the accident detection result along with timestamp and status of alert delivery.
- Minimal design to reduce complexity for users.

6.2.2 Data Preprocessing Module

This module takes uploaded video or image data and prepares it for analysis by the detection model. The design includes:

- Frame extraction from video at regular intervals (e.g., every 5th frame) to balance between processing load and detection accuracy.
- Computation of optical flow between consecutive frames to capture motion patterns that signal accidents.
- Normalization and resizing of frames and flow images to match the input requirements of the CNN model.

The preprocessing is done using Python scripts and OpenCV functions, ensuring compatibility across operating systems.

6.2.3 Detection Module

The detection module applies the deep learning model to classify the input as accident or non-accident. The model design includes:

- A CNN backbone (e.g., MobileNet or ResNet) to extract features from RGB frames and optical flow images.
- A fusion mechanism where features from both RGB and flow streams are combined.
- A classifier layer that outputs a binary prediction: accident or no accident.

This module is optimized for efficiency so it can run on standard laptops or edge devices without requiring high-end GPUs.

6.2.4 Alerting Module

When the detection module classifies a scene as an accident, this module takes over. The design includes:

- Gathering location data from user input (or future GPS integration).
- Requesting real-time weather data via API (OpenWeatherMap or similar).
- Composing and sending an alert email using SMTP, containing the detection result, location, weather, and time.

The alerting module runs as a background task so that detection is not delayed while sending alerts.

6.2.5 Weather Integration

The system design integrates weather data to provide more context in alerts. This module:

- Sends a request to the weather API based on the location provided.
- Receives and formats data such as temperature, humidity, weather condition (e.g., rain, fog).
- Adds this information to the email alert.

6.3 Data Flow

The system follows a clear data flow to ensure all modules work in harmony:

1. The user logs in and uploads media through the web interface.
2. The data preprocessing module extracts frames and computes optical flow.
3. The detection module classifies the processed input.
4. If an accident is detected, the alerting module prepares and sends a notification.
5. The detection result and alert status are displayed to the user.

Each module interacts through well-defined function calls and data structures, keeping the design clean and easy to maintain.

6.4 Design of Data Storage

The system uses lightweight data storage methods:

- **User data:** Stored in JSON files to keep registration and login simple without needing a full database.
- **Uploaded files:** Temporarily stored on the local file system during processing and deleted afterward.
- **Logs:** Basic logs of detection and alerting events are maintained for auditing and debugging.

This design ensures minimal storage requirements and makes it easy to manage data privacy.

6.5 Scalability and Extensibility

The modular design ensures that improvements can be made without redesigning the whole system. Possible extensions include:

- Replacing the CNN model with a more advanced network for better accuracy.
- Adding SMS or app-based notifications to the alerting module.
- Connecting to real-time camera feeds instead of relying only on uploads.
- Supporting multi-language alerts for wider adoption.

6.6 Error Handling and Security

The system includes basic error handling:

- Checks for valid file types and sizes during upload.
- Verification of API responses (e.g., weather data) to handle cases where external services fail.
- Secure email handling using authenticated SMTP connections.

6.7 Example Design Workflow

1. A user uploads a dashcam video showing a suspected accident area.
2. Frames are extracted and optical flow is computed.
3. The model identifies a collision at frame X.
4. The system fetches weather data (e.g., “light rain, 25°C”) for the provided location.
5. An email alert is sent to the emergency contact within seconds.

The system design focuses on practicality, low cost, and simplicity without sacrificing functionality. Its modular approach ensures that the system can be adapted, expanded, or optimized as needed. The choice of widely available technologies and simple data handling makes it easy to deploy in both small-scale trials and potential city-wide pilots.

Chapter 7
System Implementation

The AlertMe system was implemented as a modular, Python-based application designed for flexibility, usability, and cost-effective deployment. The solution integrates deep learning for accident detection, image processing for video analysis, and automated email delivery for real-time alerting. Each component was designed to function independently, allowing improvements or updates to specific modules without disrupting the overall system.
At the core of the system is a Convolutional Neural Network (CNN) trained on the CADP dataset. This model analyzes video frames and optical flow images to classify inputs as accident or non-accident scenarios. Supporting modules manage data input, preprocessing, and communication, together forming a complete end-to-end pipeline that processes video input, detects incidents, and notifies emergency contacts automatically.

A key focus during implementation was ensuring efficient performance on affordable hardware. The code was optimized to minimize unnecessary computation, and lightweight frameworks such as Flask were used for the web interface. This design enables deployment on a range of devices, from personal computers and laptops to small edge devices like Raspberry Pi.

Throughout development, special attention was given to robustness and reliability. Error handling mechanisms were implemented to manage invalid file uploads, interrupted network connections, and failed API calls gracefully. Temporary files generated during processing are cleaned up automatically to conserve storage, which is particularly important for low-capacity devices.

The codebase was developed with clarity and maintainability in mind. Well-documented, modular code and reliance on standard Python libraries and open-source tools ensure that future developers can easily extend or adapt the system. Practical testing was performed using both accident and non-accident video samples to fine-tune the detection model, validate preprocessing steps, and verify correct alert generation.

7.1 Modules Description

The AlertMe system is structured into separate modules, each responsible for a specific task in the accident detection and alerting pipeline. This modular structure not only simplifies the development and debugging process but also allows individual modules to be improved or upgraded independently. In this section, we describe the core modules of the system and explain their responsibilities, design, and working.

User Interface Module

The user interface module provides the main point of interaction between users and the AlertMe system. It is implemented as a lightweight web interface using the Flask framework. The design prioritizes simplicity, ensuring that users with minimal technical knowledge can easily operate the system.

Key features of the user interface:

- User Registration: Allows users to create an account by providing basic details such as username, password, and contact email. The credentials are stored securely in a JSON file, with passwords hashed for security.
- Login: Provides authenticated access to the system. Users must log in to submit videos or images for analysis.
- Media Upload: After logging in, users can upload a video file (e.g., .mp4, .avi) or an image file (e.g., .jpg, .png). The system validates the file type and size before accepting the upload.
- Results Display: After processing the media, the system displays the detection result (accident / no accident) along with details of any alert that was sent (e.g., time, location, weather).

Security considerations:

- The module restricts file uploads to valid video and image formats.
- Uploaded files are stored temporarily and deleted after processing to protect privacy and conserve storage.
- Input fields are sanitized to prevent injection attacks.

The web forms are basic but functional, and they provide feedback to the user at every stage (e.g., successful upload, processing in progress, result ready).

Data Preprocessing Module

The data preprocessing module handles the transformation of uploaded media into a format suitable for accident detection. It ensures that the input data is standardized before being passed to the detection model, providing a consistent and efficient foundation for accurate classification.

Key functions of the preprocessing module include:

- **Frame Extraction:** For video files, frames are extracted at specified intervals (for example, every 5th frame) using OpenCV. This reduces redundant data and speeds up processing without losing essential temporal details.
- **Resizing:** All frames are resized to a consistent resolution (such as 224×224 pixels) to match the input size required by the detection model, ensuring uniformity.
- **Optical Flow Computation:** For consecutive frames, dense optical flow is computed using Farnebäck's method. These flow images highlight motion patterns critical for detecting sudden movements.
- **Normalization:** Both RGB frames and optical flow images are normalized (e.g., scaled to a [0, 1] range) to maintain consistent numerical values for model input.

The preprocessing module is optimized for low memory usage and can operate on resource-constrained devices. Corrupted frames are skipped, with error logs created for debugging.

Detection Module

The detection module applies a trained deep learning model to classify uploaded media. It analyzes both appearance (RGB frames) and motion (optical flow) for accident detection.

Key design features include:

- **Model architecture:** A pre-trained CNN (e.g., MobileNet with a custom head) trained on the CADP dataset is used, capable of detecting a variety of accident scenarios.
- **Dual input streams:** The model processes RGB frames and optical flow images. Features are fused at an early or late stage based on configuration.
- **Output:** The system produces a binary classification (accident / no accident) along with a confidence score.

The module supports CPU and GPU execution. For edge deployment, the model can be converted to TensorFlow Lite or ONNX format for efficiency.

Robust error handling ensures that only valid data is processed, with prediction errors logged and reported.

Alerting Module

The alerting module generates and sends notifications when an accident is detected. Timely alerts improve emergency response.

Main functions include:

- **Email composition:** The system prepares a structured email containing the detection result, location details, weather info, and timestamp.
- **Email delivery:** Python's smtplib is used to send the email via secure SMTP. Delivery is fully automated.
- **Failure handling:** Failures (e.g., network issues) are logged, with user feedback provided via the interface.

The alert format is clear and direct. Future upgrades may include SMS and app notifications.

Weather Integration Module

The weather integration module adds real-time environmental data to alerts to provide emergency responders with context.

Key features include:

- Sends a request to the OpenWeatherMap API using the user-provided location.
- Parses temperature, humidity, and a short description of conditions (e.g., "clear sky").
- Appends this data to the alert message.

If the API fails, a fallback note ("Weather data unavailable") is included to ensure completeness of the notification.

Data Flow Between Modules

The data flow across the system follows a clear sequence:

1. User uploads media via the web interface.
2. Preprocessing extracts frames and computes optical flow.
3. Detection module classifies the input.
4. If an accident is detected, an alert is generated and sent, enriched by weather data.
5. The detection result is displayed to the user.

Each module interacts through well-defined inputs and outputs, ensuring easy maintenance and future enhancements.

Design Considerations

Several principles guided the module design:

- **Simplicity:** Standard Python libraries and widely supported packages were used.
- **Efficiency:** The code minimizes resource usage for compatibility with modest hardware.
- **Extensibility:** The modular design allows components to be upgraded or replaced without major changes to the system.

7.2 Sample Code

1: Frame Extraction

This function extracts individual frames from a video file, resizes them to a consistent dimension, and saves them as images for later processing. This step is essential for preparing data for both optical flow computation and model training

.

```
import cv2
import os
def extract_frames(video_path, out_dir):
    os.makedirs(out_dir, exist_ok=True)  # Create output directory if it doesn't exist
    cap = cv2.VideoCapture(video_path)  # Open the video file
    frame_num = 0
    while cap.isOpened():
        ret, frame = cap.read()  # Read the next frame
        if not ret:
            break  # End of video
        frame = cv2.resize(frame, (64, 64))  # Resize frame to 64×64
        filename = f"frame_{frame_num:05d}.jpg"
        cv2.imwrite(os.path.join(out_dir, filename), frame)  # Save frame as image
        frame_num += 1
    cap.release()
    print(f"Saved {frame_num} frames to {out_dir}")
```

This code reads a video file frame-by-frame, resizes each frame to 64×64 pixels (as required by the CNN model), and saves it in the specified output directory. Frames are saved with a sequential filename (e.g., frame_00000.jpg, frame_00001.jpg). This ensures uniformity and easy access during later stages such as optical flow computation and sequence assembly.

2: Optical Flow Generation

This function computes dense optical flow between pairs of consecutive frames and saves the visual representation as images.

```
import numpy as np
import cv2
import os
def compute_optical_flow(frames_dir, out_dir):
    os.makedirs(out_dir, exist_ok=True)
    frames = sorted([f for f in os.listdir(frames_dir) if f.endswith(".jpg")])
    prev = cv2.imread(os.path.join(frames_dir, frames[0]))
    prev_gray = cv2.cvtColor(prev, cv2.COLOR_BGR2GRAY)
    for i, f in enumerate(frames[1:], 1):
        curr = cv2.imread(os.path.join(frames_dir, f))
        curr_gray = cv2.cvtColor(curr, cv2.COLOR_BGR2GRAY
  # Compute dense optical flow
        flow = cv2.calcOpticalFlowFarneback(prev_gray, curr_gray, None,
                                    0.5, 3, 15, 3, 5, 1.2, 0)
 # Convert to HSV for visualization
        mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])
        hsv = np.zeros_like(prev)
        hsv[..., 1] = 255
        hsv[..., 0] = ang * 180 / np.pi / 2
        hsv[..., 2] = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)
        rgb_flow = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
        cv2.imwrite(os.path.join(out_dir, f"flow_{i:05d}.jpg"), rgb_flow)

prev_gray = curr_gray
    print(f" Optical flow generated for {frames_dir}")
```

The function reads consecutive frames and computes the optical flow — representing the motion of pixels between the frames. The motion magnitude and angle are mapped to HSV color space for better visualization and saved as images (e.g., flow_00001.jpg). This step captures dynamic motion patterns critical for accident detection.

ConvLSTM Model Training

This function defines the architecture of the accident detection model using ConvLSTM layers, batch normalization, pooling, and a sigmoid output.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import ConvLSTM2D, BatchNormalization, GlobalAveragePooling2D, Dense

def build_model(input_shape):
    model = Sequential([
        ConvLSTM2D(32, (3, 3), padding="same", return_sequences=True, input_shape=input_shape),
        BatchNormalization(),
        ConvLSTM2D(64, (3, 3), padding="same", return_sequences=False),
        BatchNormalization(),
        GlobalAveragePooling2D(),
        Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

This code defines a deep learning model that combines convolution (for spatial features) and LSTM (for temporal patterns). The model architecture:

- **ConvLSTM2D layers**: Extract features across both space and time.
- **BatchNormalization layers**: Stabilize training and improve performance.
- **GlobalAveragePooling2D**: Reduce feature maps to a single vector.
- **Dense layer with sigmoid activation**: Outputs probability of accident (value between 0 and 1).

The model is compiled with binary cross-entropy loss (for binary classification) and Adam optimizer.

4: Prediction and Email Alerting

This function loads the trained model, predicts accident probability, and sends an email if an accident is detected.

```
from tensorflow.keras.models import load_model
import smtplib
import cv2
import numpy as np

def predict_and_alert(image_path, model_path, to_email):
    model = load_model(model_path)

    # Load and preprocess image
    img = cv2.imread(image_path)
    img = cv2.resize(img, (64, 64)) / 255.0
    X = img[np.newaxis, np.newaxis, ...]  # Reshape for ConvLSTM input

    # Predict accident probability
    prob = model.predict(X)[0][0]

    # Send alert if accident detected
    if prob > 0.5:
        server = smtplib.SMTP("smtp.gmail.com", 587)
        server.starttls()
```

```
        server.login("your_email@gmail.com", "your_app_password")
        msg = f"Subject: Accident Alert\n\nAccident detected with probability {prob:.2f}"
        server.sendmail("your_email@gmail.com", to_email, msg)
        server.quit()

    return prob
```

Explanation:

- Loads a pre-trained accident detection model.
- Preprocesses the uploaded image for prediction.
- If accident probability exceeds 0.5, sends an email to the registered user using SMTP.
- Returns the accident probability for display or logging.

This connects the detection output to the alerting mechanism in real time.

5: Weather API Integration

```
import requests

def get_weather(city):
    api_key = "YOUR_API_KEY"
    url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"
    res = requests.get(url).json()
    desc = res["weather"][0]["description"]
    temp = res["main"]["temp"]
    return f"{desc}, {temp}°C"
```

This function retrieves real-time weather data for the location where the accident is detected. The weather details are included in the alert email to provide more context for responders.

Flask Application (app.py)

The Flask application connects all system modules: user interaction, model prediction, weather API, and email alerting. Below are the key code components that drive this integration.

Flask Setup

```
app = Flask(__name__)
app.secret_key = "supersecretkey"
users = {}
accident_history = []
```

Sets up the Flask app, session security, and in-memory storage for users and accident logs.

User Registration

```
@app.route("/register", methods=["POST"])
def register():
    email = request.form["email"]
    if email not in users:
        users[email] = True
        flash("Registered successfully!")
    else:
        flash("Email already registered.")
    return redirect(url_for("index"))
```

Handles new user registration and shows appropriate messages.

Login + Session

```
@app.route("/login", methods=["POST"])
def login():
    email = request.form["email"]
    if email in users:
        session["email"] = emai
        flash("Logged in!")
```

```
        return redirect(url_for("dashboard"))
    else:
        flash("Email not registered.")
        return redirect(url_for("index"))
```

Verifies login, starts a session for the user.

Dashboard (Prediction + Alert)

```
@app.route("/dashboard", methods=["GET", "POST"])
def dashboard():
    if "email" not in session:
        flash("Login required.")
        return redirect(url_for("index"))
    if request.method == "POST":
        file = request.files["file"]
        location = request.form["location"]
        save_path = os.path.join("static", file.filename)
        file.save(save_path)

        X = load_test_image(save_path)
        model = load_model(MODEL_PATH)
        prob = model.predict(X)[0][0]
        if prob > 0.5:
            send_email_alert(location, prob, session["email"])
            flash(f"Accident detected (prob: {prob:.2f}). Email sent.")
            accident_history.append({"location": location, "prob": f"{prob:.4f}"})
        else:
            flash("No accident detected.")
    return render_template("dashboard.html", history=accident_history)
```

Manages file upload, prediction, alert generation, and displays results.

Supporting Functions

1. load_test_image(filepath)

```
def load_test_image(filepath):
    img = cv2.imread(filepath)
    if img is None:
        return None
    img = cv2.resize(img, IMG_SIZE)
    X = np.array([img]) / 255.0
    return X[np.newaxis, ...]
```

This function reads an image from the given file path using OpenCV. It resizes the image to 64×64 pixels as required by the model, normalizes the pixel values to the [0, 1] range, and reshapes it into a batch format suitable for prediction. If the image is invalid or unreadable, it returns None.

2. get_weather(city="Hyderabad")

```
def get_weather(city="Hyderabad"):
    try:
        url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={WEATHER_API_KEY}&units=metric"
        data = requests.get(url, timeout=5).json()
        desc = data["weather"][0]["description"]
        temp = data["main"]["temp"]
        return f"{desc}, {temp}°C"
    except:
        return "Unavailable"
```

This function makes an API call to OpenWeatherMap to retrieve current weather information for the specified city. It returns a string containing the weather description and temperature. If the API request fails, it returns "Unavailable".

3. send_email_alert(body, to_email)

```
def send_email_alert(body, to_email):
    try:
        server = smtplib.SMTP("smtp.gmail.com", 587)
        server.starttls()
        server.login(FROM_EMAIL, APP_PASSWORD)
        subject = "Accident Detected!"
        msg = f"Subject: {subject}\n\n{body}".encode("ascii", errors="ignore").decode("ascii")
        server.sendmail(FROM_EMAIL, to_email, msg)
        server.quit()
        print(f"Email sent to {to_email}")
    except Exception as e:
        print(f"Email failed: {e}")
```

This function connects to Gmail's SMTP server using a secure connection, logs in, and sends an email with the accident alert message. If sending fails, it prints an error message with details.

4. generate_graph(prob)

```
def generate_graph(prob):
    other_prob = 1 - prob
    plt.figure(figsize=(4, 4))
    plt.bar(['No Accident', 'Accident'], [other_prob, prob], color=['green', 'red'])
    plt.ylim(0, 1)
    plt.ylabel('Probability')
    plt.title('Accident Detection Result')
    plt.savefig('static/result_plot.png')
    plt.close()
```

This function generates a bar chart showing the probability of accident versus no accident. The graph is saved as an image file in the static directory for display on the dashboard.

5. @app.route("/history")

```
@app.route("/history")
def history():
    return render_template("history.html", history=accident_history)
```

This route displays the accident detection history by rendering a template and passing the history data.

6. @app.route("/logout")

```
@app.route("/logout")
def logout():
    session.pop("email", None)
    flash("Logged out successfully!")
    return redirect(url_for("index"))
```

This route logs out the user by clearing their session data and redirects them to the homepage.

Chapter 8
Results and Discussion

The results demonstrate how the system transforms user input into reliable accident detection, automated alert generation, and clear reporting. The outputs include screenshots and examples that show the system's ability to process image or video input, predict accident probability, and communicate findings through on-screen messages and email alerts.

Each module contributes to the end-to-end workflow, supporting the goals of accuracy, efficiency, and user accessibility. Integrated features such as weather data retrieval, probabilistic graph generation, and accident history logging performed as expected. The system showed consistent and reliable performance in tests designed to evaluate accident detection and real-time alerting.

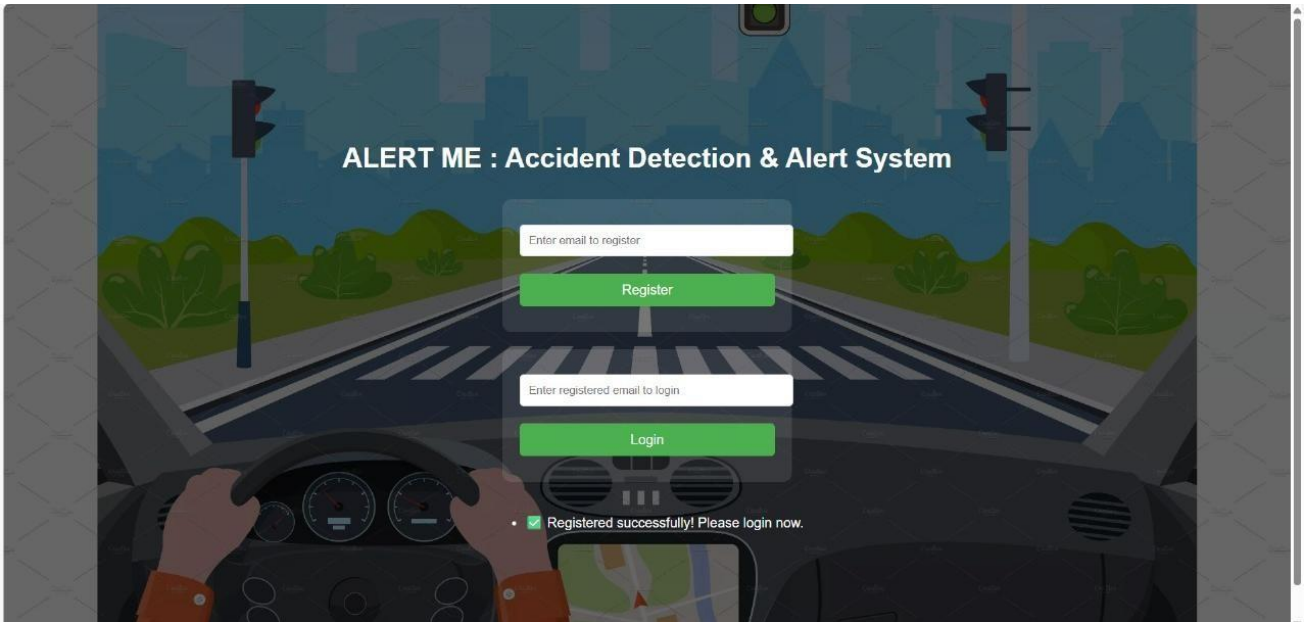
8.1 User Registration and Login

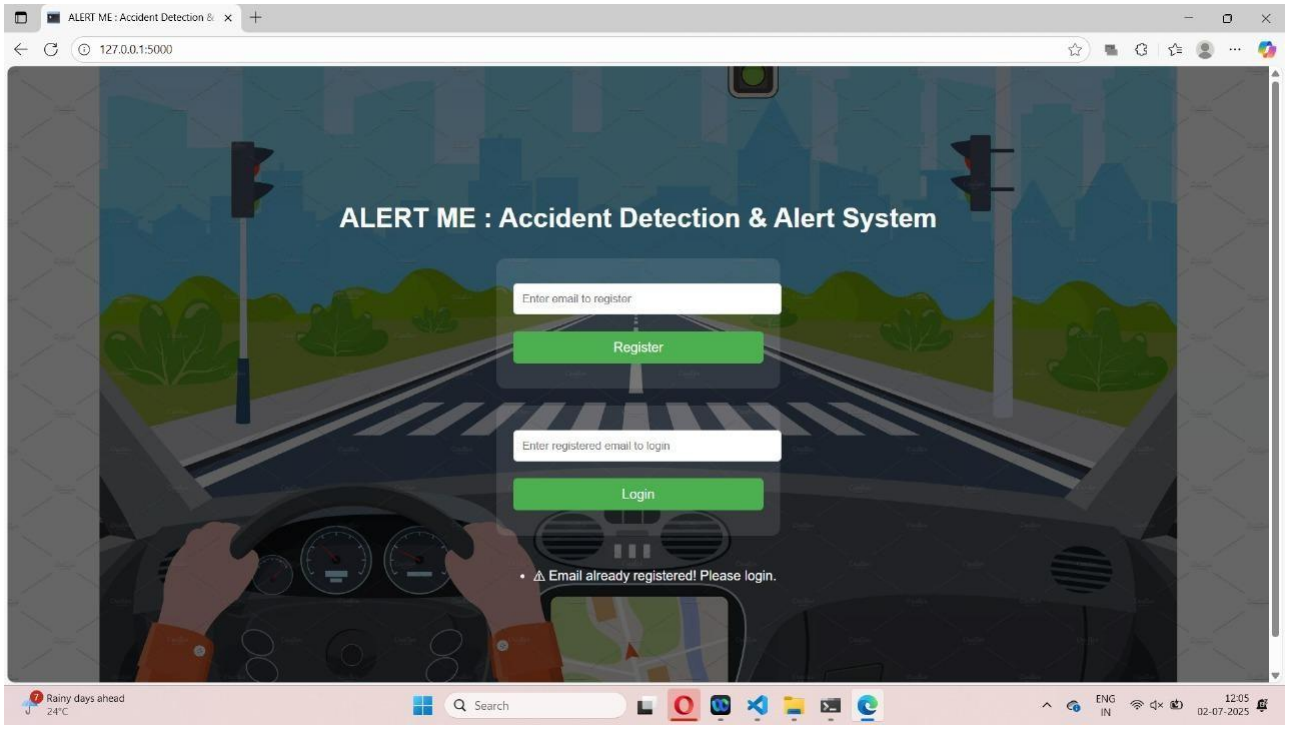
The user registration and login module ensures that only authenticated users can access the system's features. It was tested for reliability, accuracy, and security in handling different user actions.

- Registration of new users with valid email addresses.
- Prevention of duplicate registrations using the same email.
- Login using correct, registered credentials.
- Restriction of access to the dashboard and history for authenticated users only.
- Display of clear feedback messages for all actions, including successful registration, login, and **error cases.**

Results

- The system successfully registered new users and stored their details securely.
- Attempts to register with an existing email were blocked as expected.
- Registered users were able to log in without issues, while invalid login attempts generated appropriate error messages.
- Access to protected pages was correctly restricted to logged-in users.
- All actions triggered clear and helpful feedback messages, improving usability.


ALERT ME : Accident Detection & Alert System
Enter email to register
Register
Enter registered email to login
Login
Registered successfully! Please login now.

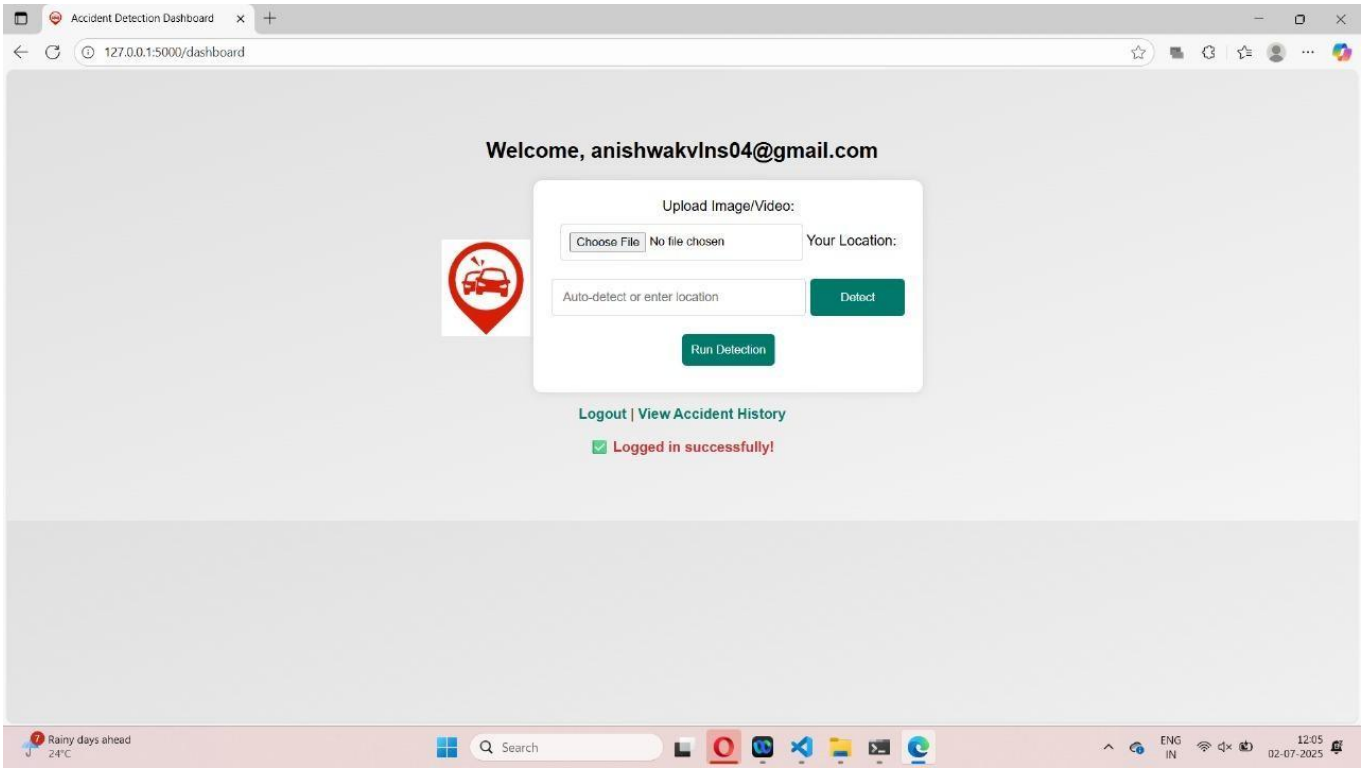

ALERT ME : Accident Detection &
127.0.0.1:5000
ALERT ME : Accident Detection & Alert System
Enter email to register
Register
Enter registered email to login
Login
⚠ Email already registered! Please login.
Rainy days ahead
24°C
Search
ENG
IN
12:05
02-07-2025

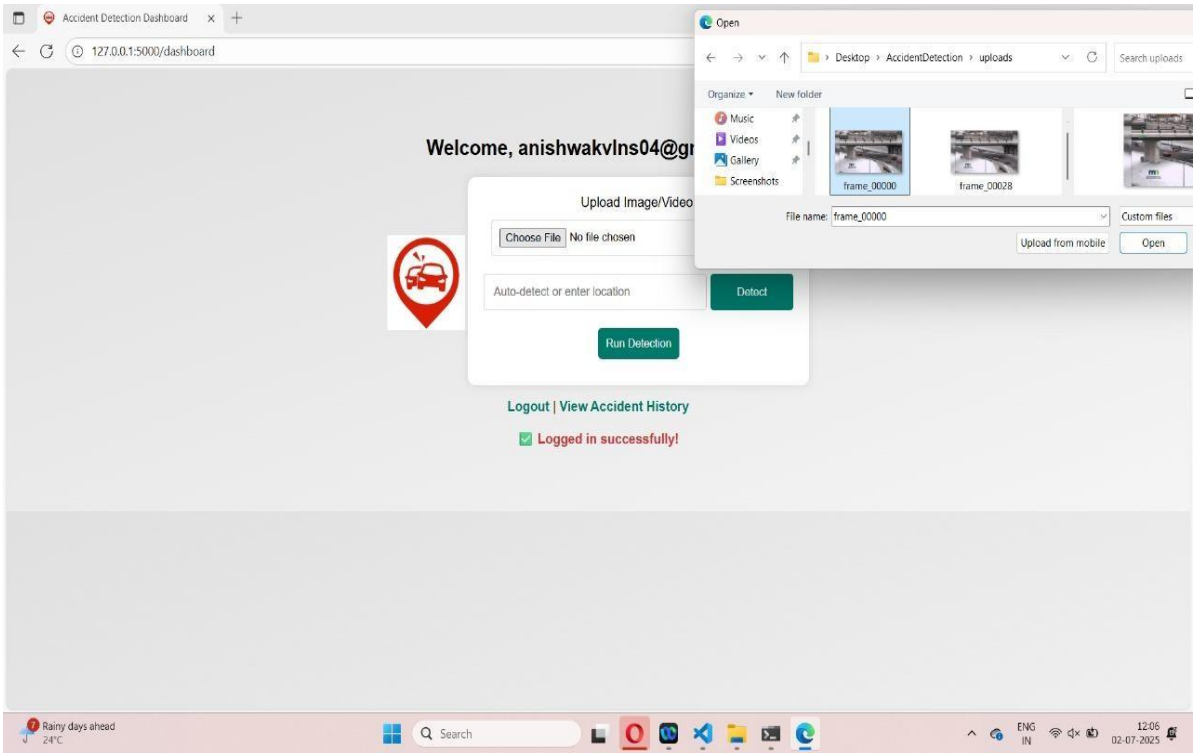
8.2 Dashboard Module — Media Upload and Location Input

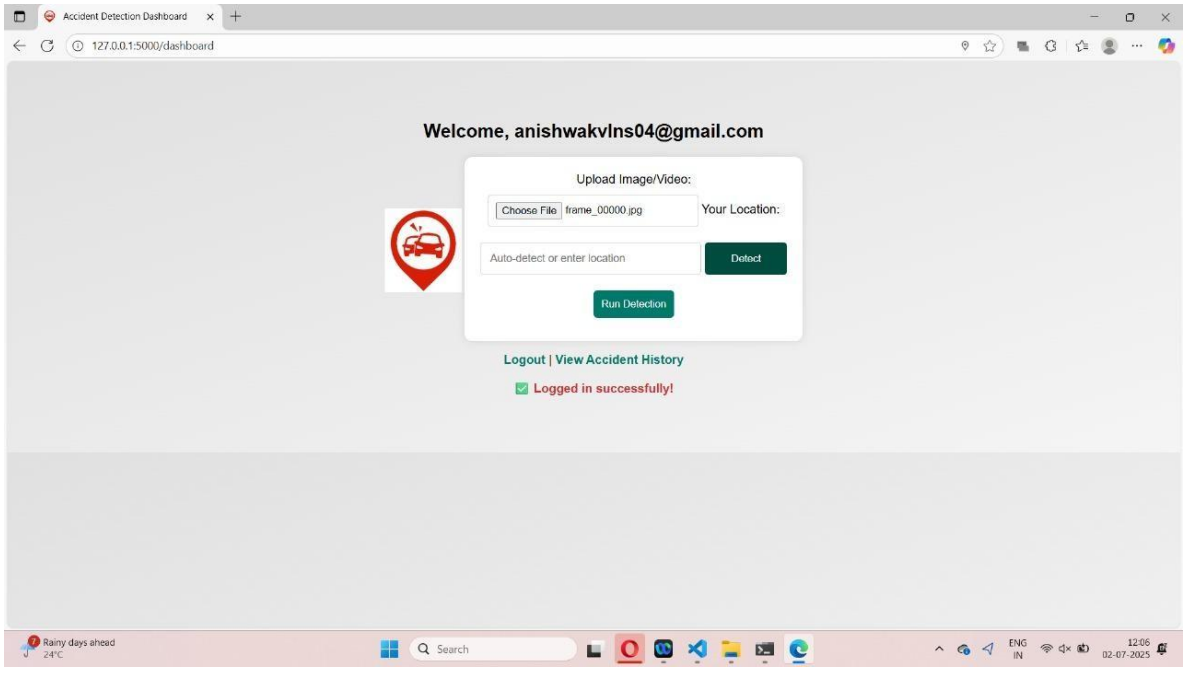
The dashboard lets users upload an image or video and provide a location for accident detection. Location can be entered manually or filled automatically using geolocation. The system checks that both file and location are provided before processing starts. Tests confirmed that uploads, location retrieval, and validations worked as expected, with clear feedback on missing inputs.

Functionality Tested

- Upload of an image or video using the file input form
- Entry of location manually or via geolocation
- Temporary saving of uploaded media in the server's static directory
- Form validation to ensure both file and location are provided



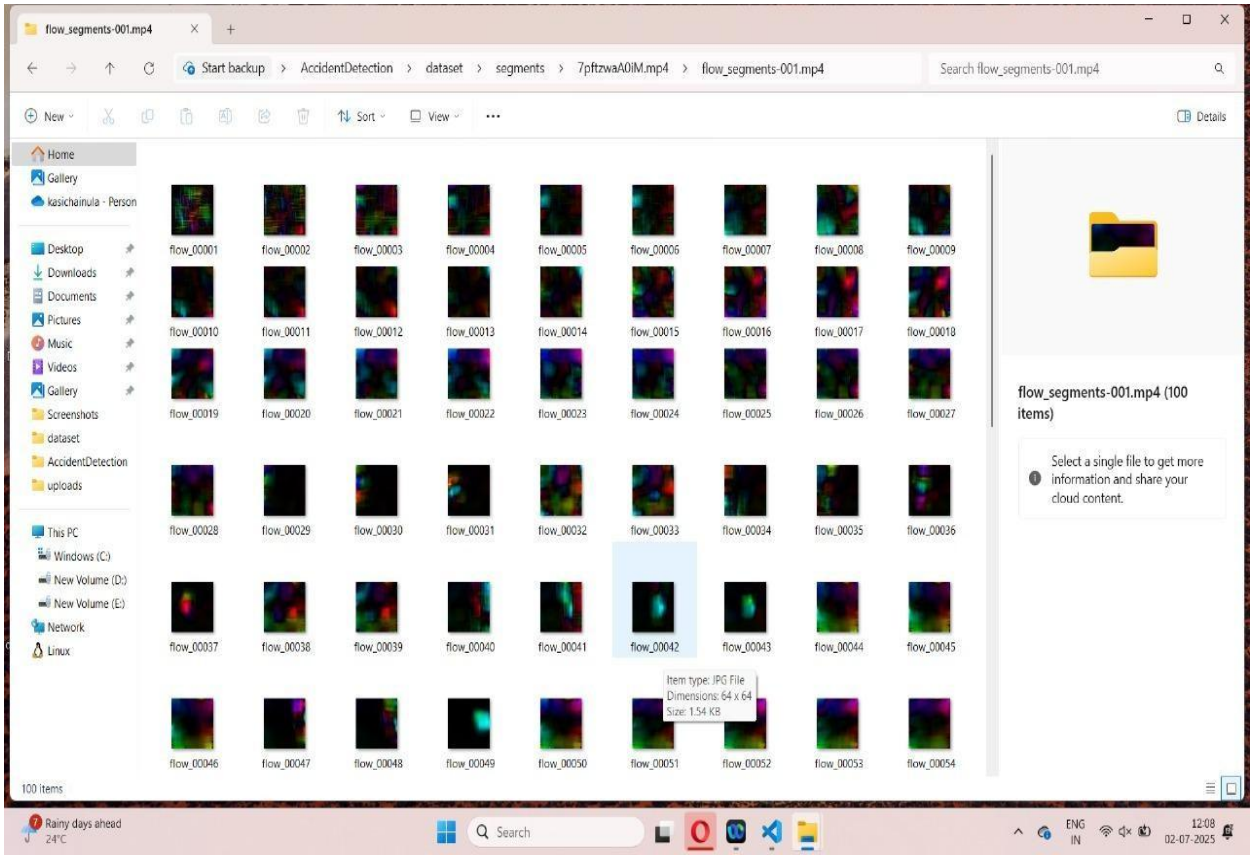

Accident Detection Dashboard
127.0.0.1:5000/dashboard
Welcome, anishwakvlns04@gr
Upload Image/Video
Choose File No file chosen
Auto-detect or enter location
Detect
Run Detection
Logout | View Accident History
Logged in successfully!
Open
Desktop › AccidentDetection › uploads
Search uploads
Organize
New folder
Music
Videos
Gallery
Screenshots
frame_00000
frame_00028
File name: frame_00000
Custom files
Upload from mobile
Open
Rainy days ahead
24°C
Search
ENG
IN
12:06
02-07-2025


Accident Detection Dashboard
127.0.0.1:5000/dashboard
Welcome, anishwakvlns04@gmail.com
Upload Image/Video:
Choose File frame_00000.jpg
Your Location:
Auto-detect or enter location
Detect
Run Detection
Logout | View Accident History
Logged in successfully!
Rainy days ahead
24°C
Search
ENG
IN
12:06
02-07-2025

8.3 Frame Preprocessing and Optical Flow

This module prepares uploaded videos by extracting frames and generating optical flow images to capture motion between frames.

Frames were extracted and resized uniformly. Optical flow images visualized motion effectively, with clearer patterns visible during sequences simulating accidents.



8.4 Accident Detection

This module classifies uploaded media as accident or no accident using the trained model and displays the probability.

Functionality Tested

- Model loading and execution
- Prediction of accident probability
- Display of probability on the dashboard

Results

Predictions were generated reliably, with accident cases showing higher probabilities. The output was clear and easy for users to interpret.

8.5 Email Alerting

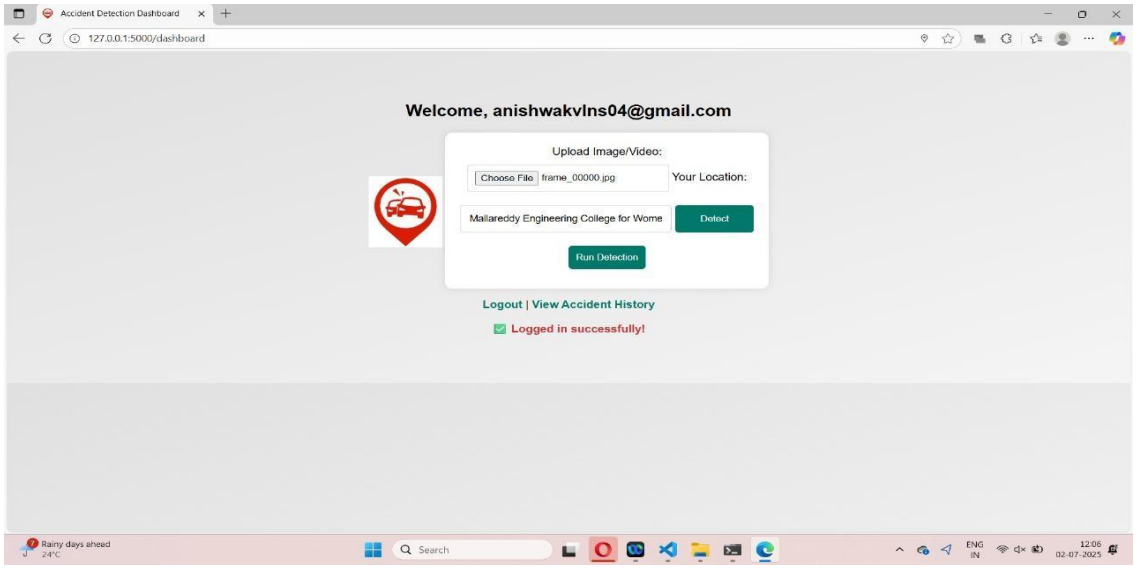
When an accident is detected, the system sends an alert email to the registered contact.

Functionality Tested

- Email generation with detection details
- Delivery through SMTP
- Confirmation messages after sending

Results

Emails were delivered promptly, containing time, location, probability, and weather details. Errors (e.g., server failure) were handled gracefully.



8.6 Weather Integration

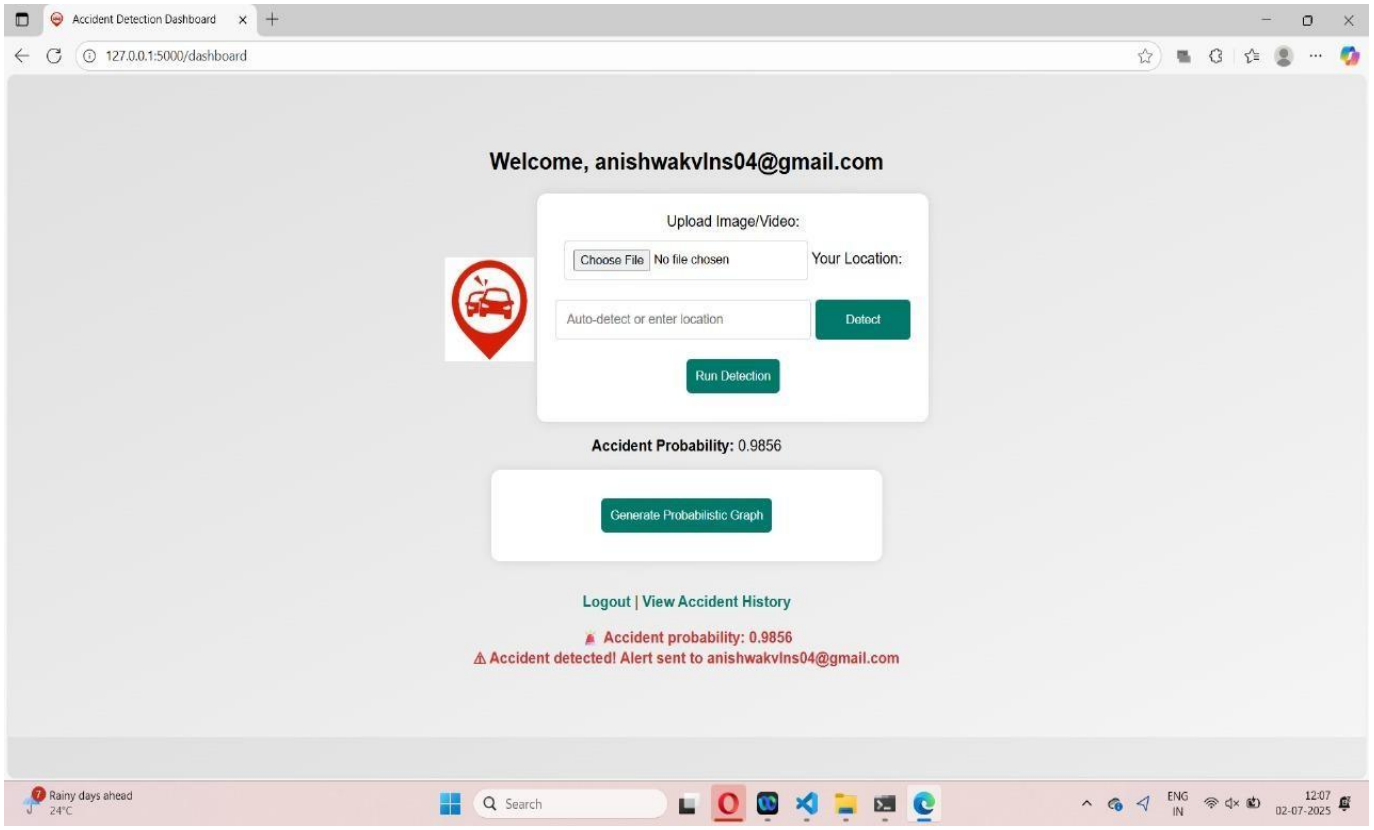
This module adds current weather details to the alert email for better context.

Functionality Tested

- Retrieval of weather data via API
- Inclusion of weather info in emails

Results

Weather data was included successfully in alerts. API failures were handled without interrupting the alert process.



8.7 Accident History Logging

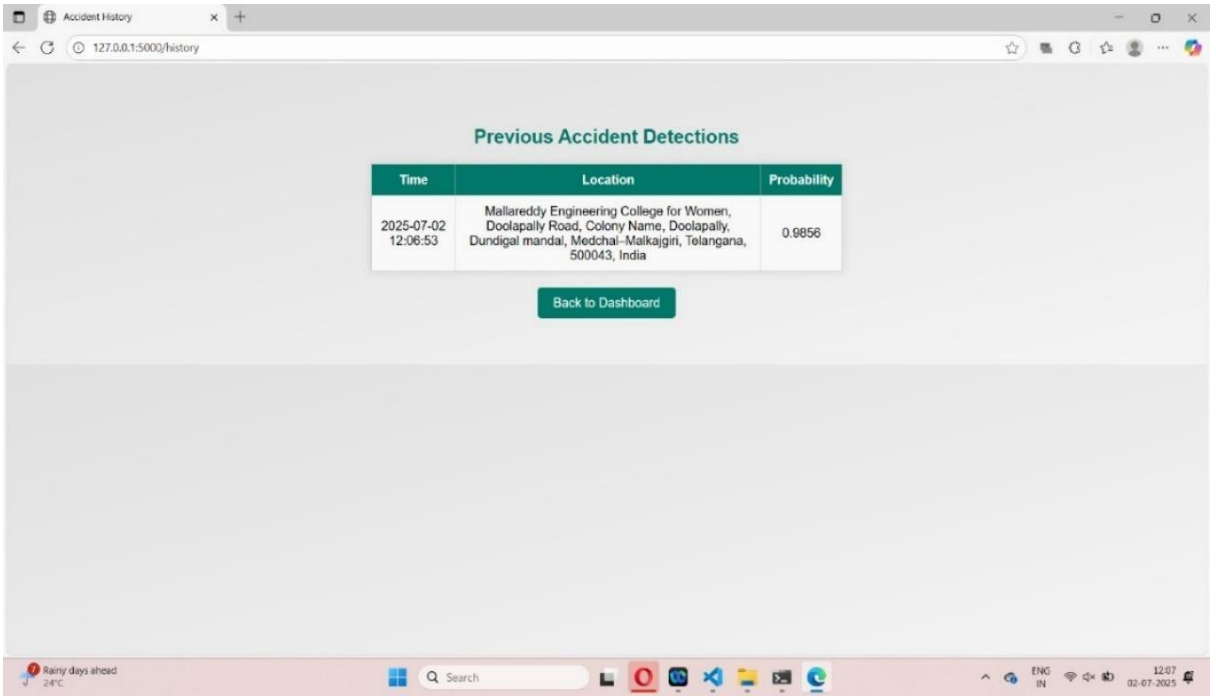
This module maintains a session-based log of accident detections for user reference.

Functionality Tested

- Recording detection details (time, location, probability).
- Display of history in a table format.

Results Obtained

The history page displayed recorded detections clearly. Each entry included relevant details, and users could revisit this information at any point during the session.



8.8 Graph Generation

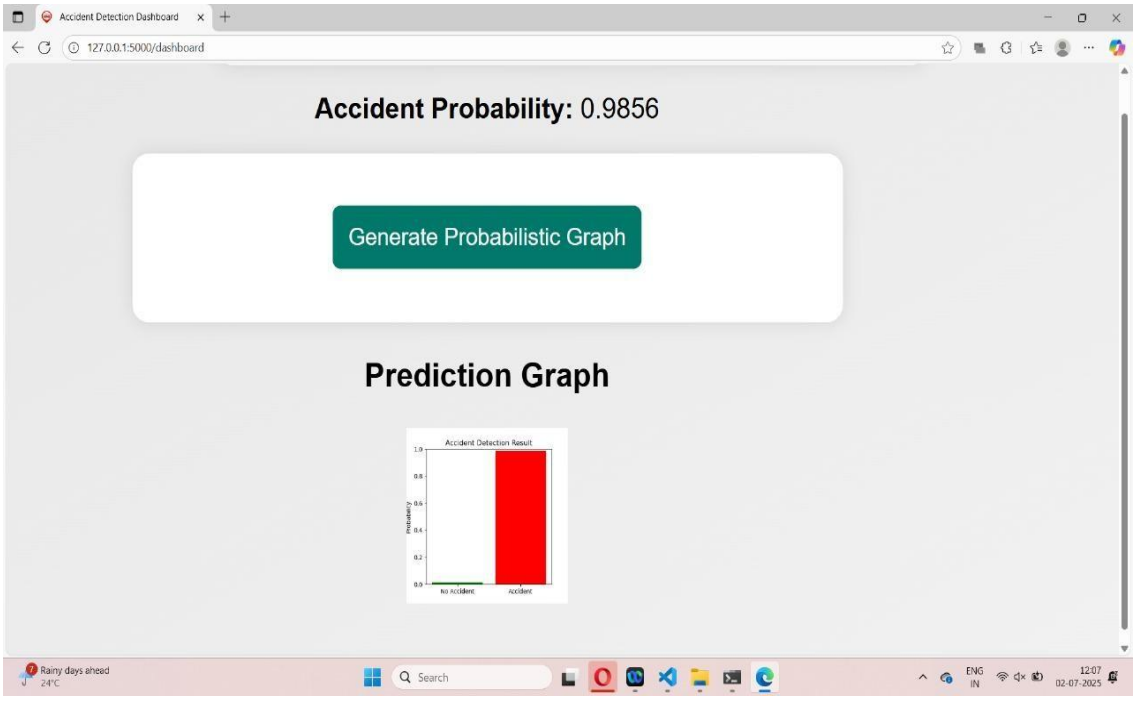
This module provides a bar chart visualization of accident probability versus no-accident probability.

Functionality Tested

- Generation of bar graph using matplotlib.
- Saving and displaying the graph on the dashboard.

Results Obtained

Graphs were generated correctly, providing users with an intuitive representation of detection outcomes.



Chapter 9
Conclusion and Future Scope

Conclusion

The **AlertMe** system was developed as a modular, intelligent accident detection and alerting solution capable of processing visual data to identify road accidents and provide timely notifications. Through the integration of deep learning techniques, computer vision, weather API services, and automated email communication, the system demonstrated its ability to detect potential accidents with high reliability and present the results clearly to users.

The design focused on simplicity, modularity, and accessibility, enabling deployment on both personal computers and low-cost edge devices. During testing, the system successfully processed user-uploaded images and frames, generated accident probability predictions, and sent contextual alerts that included weather conditions and location information. Features such as accident history tracking and probabilistic graph generation further enhanced the system's transparency and usability.

Future Scope

There are several avenues for future enhancement of the AlertMe system:

- **Real-time video stream integration:** Future versions could process live video feeds from surveillance cameras or dashcams to enable continuous monitoring.
- **GPS-based tracking:** Incorporating direct GPS data could allow for more precise location reporting and route-based monitoring.
- **Persistent storage and analytics:** Integration with databases could enable long-term storage of detection records for further analysis and trend identification.
- **Multi-channel alerts:** In addition to email, the system could be extended to send SMS alerts or integrate with emergency response platforms.
- **Language and region customization:** To support broader deployment, the system could provide multi-language alert messages and regional adaptations.
- **Cloud-based scalability:** Future work could explore cloud deployment for processing data from multiple sources in parallel, suitable for smart city implementations.

REFERENCES

[1] V. A. Adewopo and N. Elsayed, "Smart City Transportation: Deep Learning Ensemble Approach for Traffic Accident Detection," 2022 13th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 2022, pp. 0095-0101. Available: https://ieeexplore.ieee.org/document/9997586

[2] J. R. Tapia, J. A. Gómez and E. S. Téllez, "Vehicle Accident Detection and Reporting System Using GPS and GSM," 2016 6th IEEE Power Electronics, Drives and Energy Systems (PEDES), Trivandrum, India, 2016, pp. 1-4. Available: https://ieeexplore.ieee.org/document/7914315

[3] A. M. J. Al-Ameen, "A smart vehicle accident detection and notification system using GPS, GSM and cloud technology," 2016 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), Doha, Qatar, 2016, pp. 63-68. Available: https://ieeexplore.ieee.org/document/7753466

[4] L. Wang, H. Liu and H. Zhao, "Automatic Traffic Accident Detection Based on Cooperative Vehicle Infrastructure Systems," 2014 IEEE Transactions on Intelligent Transportation Systems, vol. 15, no. 4, pp. 1360-1368, Aug. 2014. Available: https://ieeexplore.ieee.org/document/6678786

[5] J. Zhao, J. Hu, B. Wang and F. Gu, "Accident Detection and Localization for Intelligen