Violence Detection in Surveillance Videos

Index

1. Project Overview

Topic: Violence Detection in Surveillance Videos

Description: This project involves the creation of an intelligent surveillance system powered by artificial intelligence and machine learning. The primary focus is on automatically identifying violent actions through video feeds in real-time, enhancing the efficiency and reliability of traditional security setups.

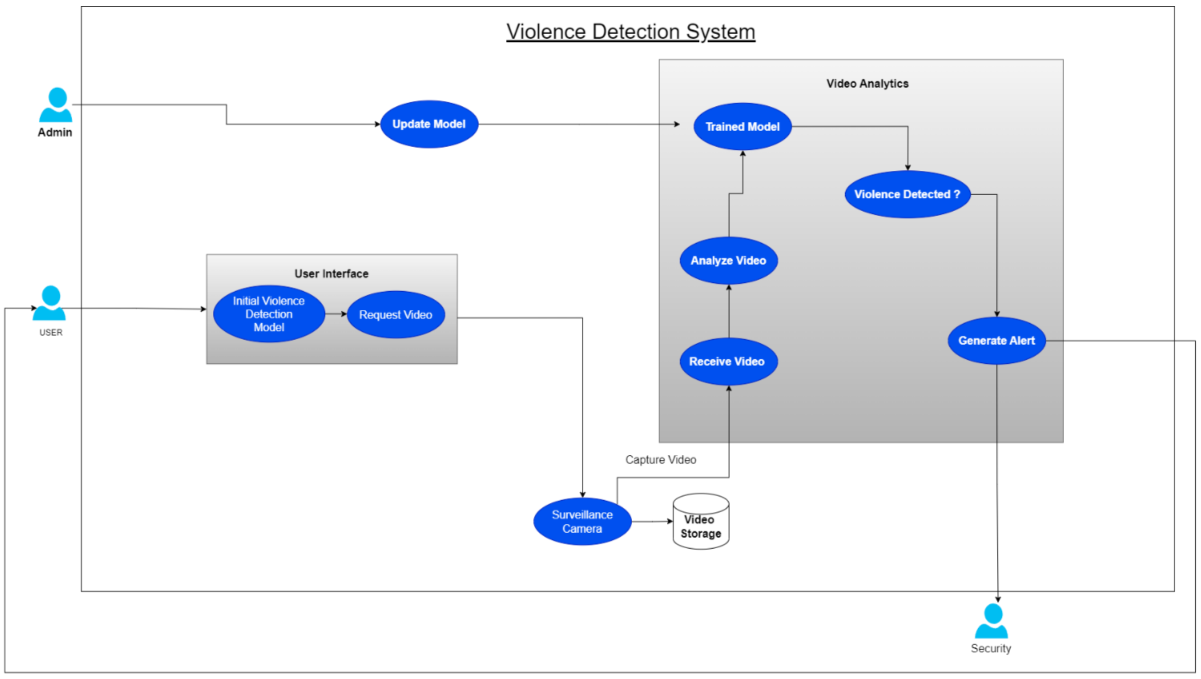
Statement: The system will leverage advanced deep learning models to analyze live video inputs from surveillance cameras and generate immediate alerts upon detecting violent behavior.

Scope: This solution is intended for integration with existing surveillance infrastructures in public and private security sectors. It targets improving incident response times, reducing manual monitoring errors, and providing a scalable and automated method for real-time violence detection.

2. System Architecture

This section provides a detailed breakdown of the design diagrams that form the core of the system's structure and data flow logic.

a) Use Case Diagram



Explanation: The diagram illustrates the interaction of various system actors and the core components.

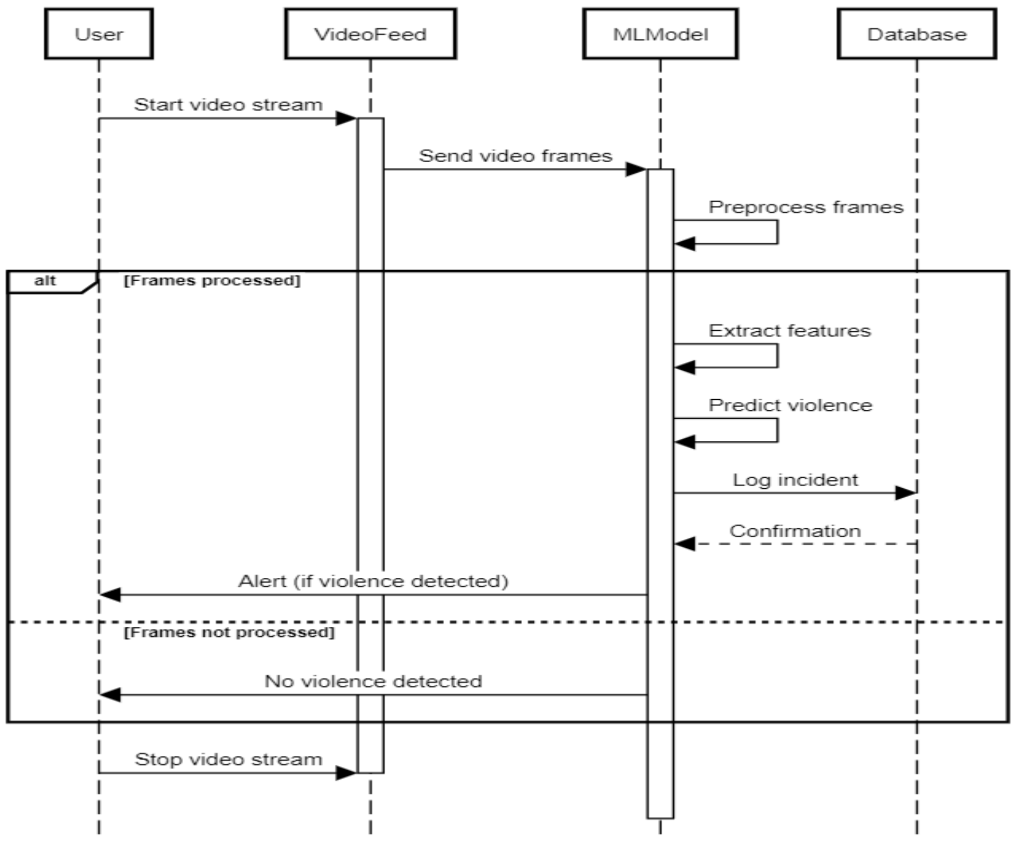
Actors:

1. **Admin** – Updates and maintains the violence detection model.
2. **User Interface (UI)** – Allows users to request video analysis and receive alerts.
3. **Surveillance Camera** – Captures and forwards video footage.
4. **ISFR (Image/Video Frame Reader)** – Reads frames from stored video.
5. **Video Analytics** – Analyzes frames using ML models.
6. **Security System** – Receives alerts based on detected incidents.

Use Cases:

1. Capture video
2. Request and receive video
3. Analyze video using ML models
4. Detect violence
5. Generate and send alerts
6. Update and retrain the model

b) Sequence Diagram



Explanation:

Outlines the flow of information and actions from the moment a user initiates a video stream to violence detection and system feedback.

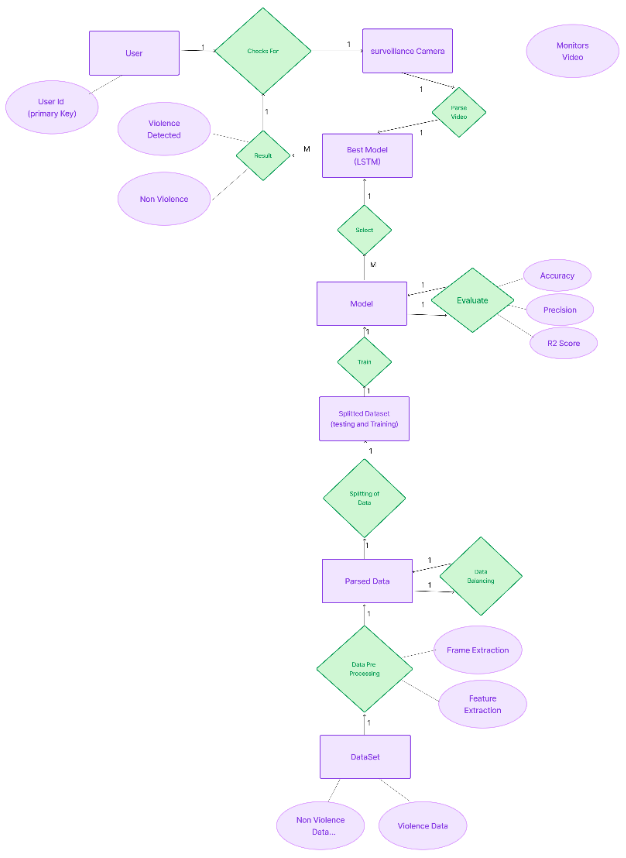
Main Components:

1. User
2. Video Feed
3. ML Model
4. Database

Process Flow:

- Video stream is initialized
- Video frames are processed
- ML model detects violence or not
- Incidents logged in database
- User receives notification
- Stream is terminated

c) ER Diagram



Explanation: Defines entities and their relationships within the system.

Entities:

- User (with unique User ID)
- Surveillance Camera
- Monitor's Video
- Model
- Preload Data
- Dataset

Relationships:

- Users initiate detection
- Cameras stream to the ML model
- Model detects and logs violence/non-violence
- Preprocessed data and datasets are cyclically used for training and tuning
- Evaluation metrics help improve model accuracy and performance

3. Tech Stack

- **Languages & Libraries**: Python, OpenCV, Pandas, NumPy, Scikit-learn, TensorFlow, Keras

- **ML Models**: CNN, LSTM
- **Tools**: GitHub for version control, Google Colab for training
- **Storage**: CSV for lightweight storage; scalable to SQL/NoSQL

4. Project Implementation Status

- **Completed**:
  - Video capturing
  - ML model training
  - Basic UI for displaying alerts
- **Ongoing**:
  - UI optimization
  - Real-time inference improvements
- **Pending**:
  - Integration of authentication
  - Final validation and performance tuning

5. Challenges Faced The development of a real-time violence detection system presented several technical and operational challenges. These challenges were both anticipated and discovered during different phases of implementation and testing.

- **Video Quality Variability**: Video frame inconsistencies due to changing lighting conditions, occlusions, and varying camera angles impacted detection accuracy.
- **Hardware Limitations**: Processing live video streams with ML models required significant computational resources, and the limited hardware available posed constraints on model complexity and speed.
- **Model Accuracy**: Reducing false positives (non-violent actions flagged as violent) and false negatives (missing actual violent actions) was a significant challenge requiring fine-tuning and iterative model training.
- **System Integration**: Ensuring all system components—video input, processing, alert generation, and logging—worked together smoothly was complex and required rigorous testing.

Solutions Considered To mitigate these challenges, the following strategies and technologies were explored:

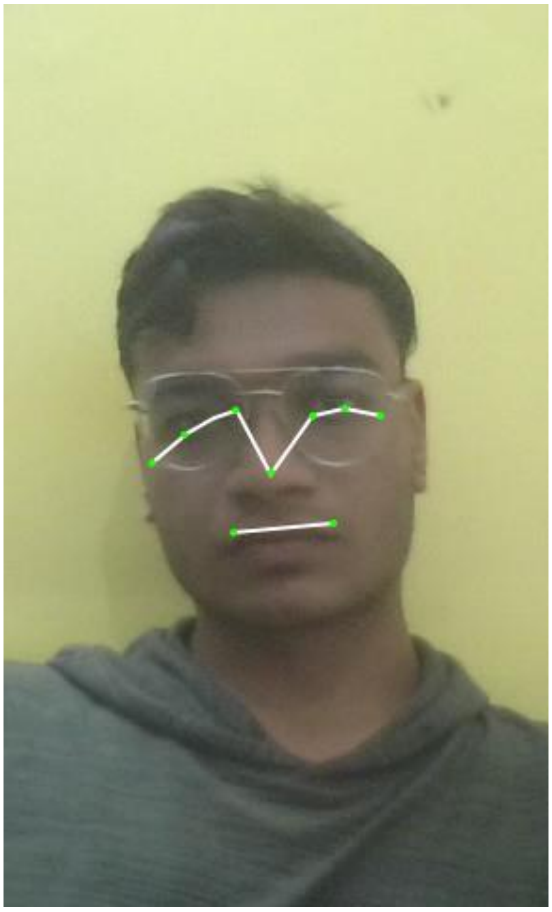
- Applied adaptive preprocessing techniques to normalize video input regardless of lighting or angle.
- Utilized lightweight, optimized versions of CNN and LSTM models for faster inference.
- Introduced confidence thresholds and ensemble model approaches to improve prediction reliability.

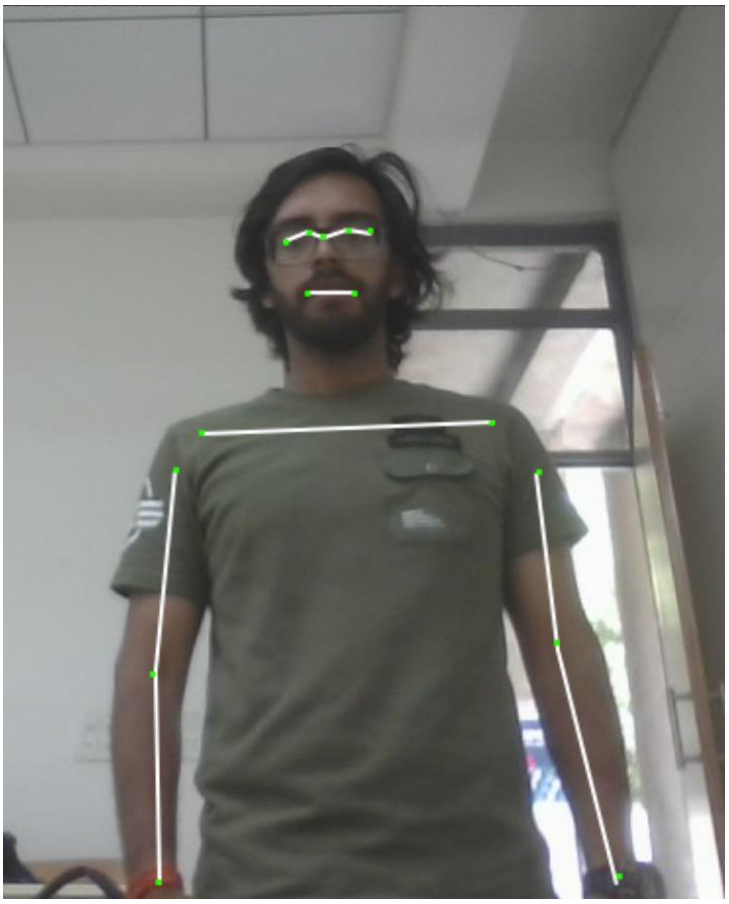
- Designed modular code architecture to simplify integration and debugging across components.

6. Testing and Quality Analysis Ensuring the quality and reliability of the system was a multi-faceted process. Different layers of testing and review were conducted throughout the project lifecycle to ensure robustness and performance.

Interfaces (Testing)

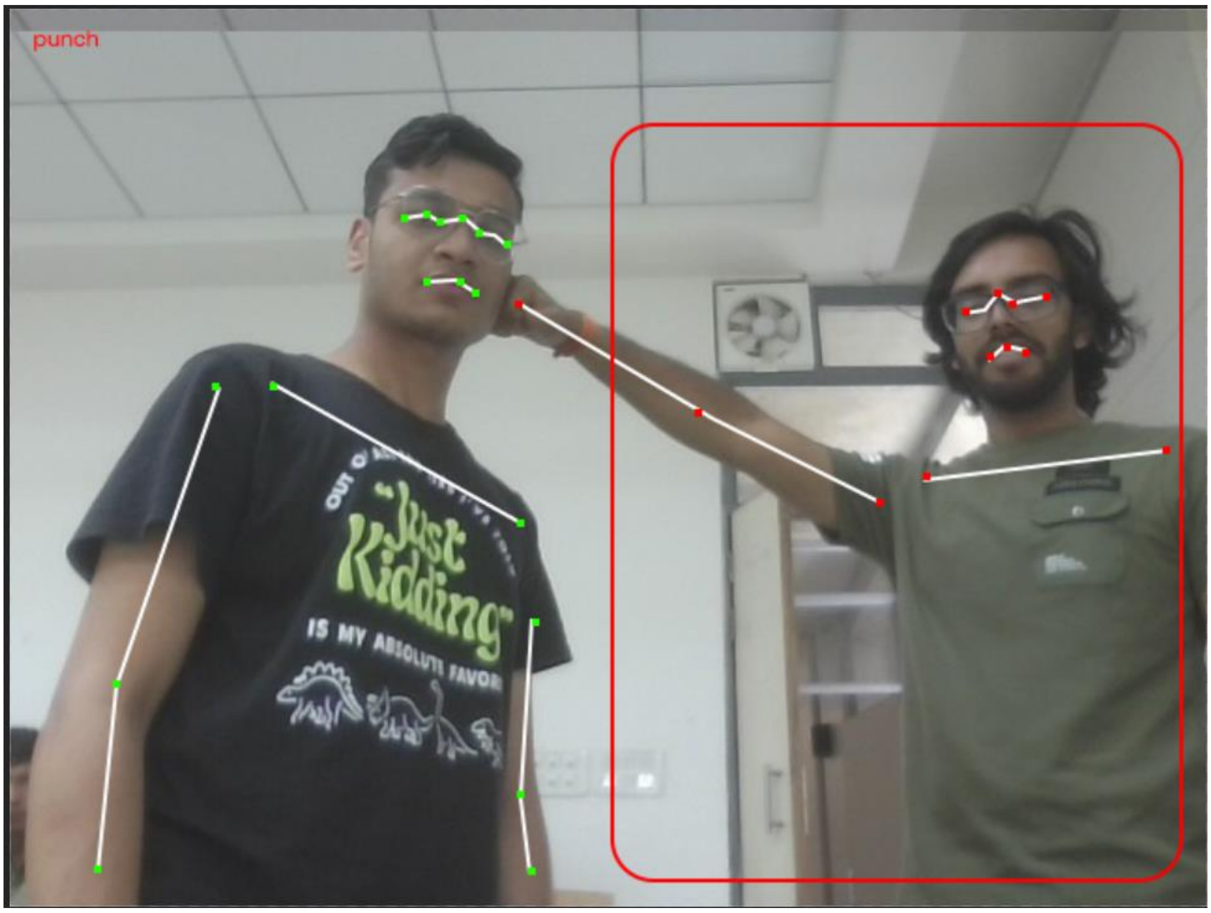
i. Interface with Device The script captures video frames from the device's camera using OpenCV (cv2.VideoCapture) and processes them to detect and analyze poses using the MediaPipe library (mpPose.Pose()). This interface ensures seamless interaction between the software and the camera device.





ii. Interface with OS The script runs on an operating system and interacts with it to access camera resources (cv2.VideoCapture) and perform file operations (e.g., reading/writing CSV files using Pandas). This interface ensures compatibility and communication with the host environment.

iii. Interface with Database Although no dedicated database is used, the script employs the Pandas library for reading and writing to CSV files, serving as a lightweight data storage layer.



iv. Interface with User The script displays real-time video frames with pose landmarks overlaid for feedback. Annotations (e.g., "neutral" or "punch") are added using OpenCV (cv2.putText) to improve interaction.

- **Unit Testing**: Each functional module, such as video capture, model loading, and alerting, was tested independently to verify correctness.
- **Integration Testing**: Full end-to-end workflows were tested to confirm smooth interaction between the video stream input, machine learning pipeline, and UI output.
- **Model Quality Checks**: The CNN-LSTM model's accuracy, precision, recall, and F1-score were evaluated using multiple datasets to ensure the model was both generalizable and reliable under different lighting and camera conditions.
- **Backend Validation**: Data logging, alert generation, and response timings were monitored to ensure the backend infrastructure maintained consistency and low-latency performance.
- **UI/UX Usability Testing**: The user interface underwent usability checks for clarity, responsiveness, and ease of navigation to ensure administrators could respond to alerts quickly and effectively.
- **Performance Metrics**: Evaluated model predictions and system throughput using Precision, Recall, and F1-Score.
- **Issue Tracking**: Development bugs and feature enhancements were managed using GitHub Issues for transparent collaboration and timely resolution.

Additional checkpoints like cross-browser UI compatibility, video lag detection, and alert response delay were also reviewed to ensure the system could perform reliably in real-world deployments.

7. Timeline and Milestones

Date	**Milestone**
15-Jan-2025	Project Kickoff & Requirement Analysis
25-Jan-2025	Data Collection & Preprocessing
05-Feb-2025	Model Prototyping
20-Feb-2025	Model Training
05-Mar-2025	Integration & UI Development
20-Mar-2025	Testing Phase
05-Apr-2025	Final Review
18-Apr-2025	Submission

8. Documentation Associated with this Project

- Software Requirement Specification (SRS)
- Design Diagrams (Use Case, Sequence, ER)
- Final Project Report
- User Manual
- GitHub Repository with code and instructions

9. Future Plans Future enhancements are envisioned to extend the capabilities, adaptability, and accessibility of the system across broader security use cases:

- **Expand Detection Scope**: Integrate additional threat detection capabilities such as theft, fire, loitering, and suspicious object tracking.
- **Cloud Migration**: Deploy system components to cloud platforms (e.g., AWS, Azure) to support high-volume video feeds, facilitate load balancing, and enable distributed processing.
- **Mobile & Web Integration**: Develop user-friendly mobile applications and web dashboards to allow remote monitoring and alert management for on-the-go security personnel.
- **Smart City Integration**: Partner with municipal bodies to integrate with smart city surveillance infrastructure, contributing to urban safety analytics.
- **AI Model Enhancement**: Continuously retrain models using feedback loops and diverse datasets to improve detection across various real-world scenarios.
- **Multi-Language Support**: Introduce multilingual support in the UI for global deployment.

- **Automated Reports & Analytics**: Generate downloadable incident reports and trend analyses for administrative and legal documentation.

These future initiatives are aimed at making the system more robust, scalable, and accessible to meet evolving security challenges across different environments.

Amit S Sahu (500123854)

Mradul Lakhotiya (500125405)

Yash (500125397)