HELWAN UNIVERSITY
Faculty of Computers and Artificial Intelligence
Computer Science Department

Video Analysis for Violence Detection

A graduation project dissertation by:

Mohamed Mostafa: 20210837

Mohamed Nasser: 20210841

Mohamed Abdelrazak: 20210807

Mohamed Awadallah: 20210822

Youssef Mohamed: 20211104

Mohamed Ayman: 20210751

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computers and Artificial Intelligence,
at the Computer Science Department,
Faculty of Computers & Artificial Intelligence, Helwan University.

Supervised by:
Dr. Islam Gamal

June 2025

Acknowledgments

We would like to express our sincere gratitude to the Faculty of Computers and Artificial Intelligence at Helwan University for providing the academic environment and resources that supported the completion of this project.

We are especially grateful to our supervisor, **Dr. Islam Gamal**, for his continuous guidance, insightful feedback, and patience throughout the development of this work. His technical expertise and mentorship were instrumental in shaping our ideas and improving the quality of our research.

We also extend our appreciation to all the researchers and engineers whose published work served as a foundation for our methodology and system design. Their contributions to the fields of machine learning, computer vision, and system architecture greatly informed our development process.

Special thanks to our colleagues and teammates for their collaboration, dedication, and mutual support throughout the project timeline. Each team member played a vital role in achieving the project goals.

Finally, we would like to acknowledge the ongoing encouragement and moral support of our families and friends, whose belief in us kept us focused and motivated from the early planning stages through to final submission.

Abstract

This graduation project presents an AI-powered violence detection system designed to enhance public safety through video analysis. The system employs deep learning algorithms to detect various forms of threats, including violence, firearms, fire hazards, and object detection in video content. Built on a three-tier architecture, the system combines an Angular frontend with role-based access control, a NestJS backend with modular services, and machine learning models trained on diverse datasets.

The solution implements multiple detection capabilities through a web interface, allowing users to upload videos for analysis. The machine learning component utilizes computer vision techniques, including YOLO-based object detection and MobileNetV2 architecture, providing detection capabilities across various scenarios. The system processes videos through a modular service architecture, with separate services for violence, gun, fire, and object detection, along with user management, alerts, and analytics.

Key features include video analysis, multi-threat detection (violence, guns, fire, and objects), user authentication and authorization, and a comprehensive reporting system. The project demonstrates practical applications in public safety, leveraging modern web technologies and AI to create a scalable solution. The implementation includes proper error handling, database integration, role-based access control, and WebSocket-based real-time notifications, making it suitable for deployment in security monitoring scenarios.

This work contributes to the field of automated surveillance systems, offering a practical tool for enhancing public safety through AI-driven video analysis and user-centric features.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In today's digital landscape, the need for automated violence detection systems has become increasingly important for content moderation and security monitoring. Traditional manual surveillance methods are not only resource-intensive but also limited by human factors such as fatigue and attention span. This challenge is particularly evident when processing video content from various sources.

The development of our violence detection system is motivated by several key factors:

- **Automated Content Analysis:** The system provides automated detection of violent content in videos, offering a more efficient alternative to manual monitoring. This includes:
  - Real-time analysis of uploaded video content
  - Binary classification of violent and non-violent scenes
  - Confidence scoring for detection accuracy
  - Frame-by-frame analysis with timestamp tracking
- **Enhanced Detection Capabilities:** The system incorporates multiple detection modules:
  - Gun detection for identifying firearms
  - Fire detection for monitoring fire hazards
  - Object detection for tracking potential threats
- **Resource Efficiency:** The automated system enables:
  - Efficient processing of video content
  - Reduced manual monitoring requirements
  - Scalable analysis capabilities

These factors, combined with recent advancements in computer vision and deep learning, have enabled the development of a comprehensive violence detection solution that addresses the growing need for automated content analysis and security monitoring.

1.2 Overview

This project implements a violence detection system that combines machine learning models with a modern web interface. The system analyzes video content to detect instances of violent behavior, providing a tool for security and content moderation applications.

The system architecture consists of three main components:

- A machine learning backend that processes video content using trained neural network models
- A REST API server built with NestJS that handles detection requests and data management
- A web interface developed with Angular for user interaction and result visualization

The solution integrates multiple specialized services:

- Core detection modules for violence, gun, and fire detection
- Backend services for user authentication and data management
- Frontend components for video upload and analysis display

This implementation enables video analysis and violence detection, making it suitable for applications such as content moderation and security monitoring. The system processes uploaded video content and provides detection results with confidence scores and timestamps.

1.3 Problem Definition

The increasing prevalence of violent content in digital spaces presents significant challenges for content moderation and security monitoring. Traditional manual surveillance approaches suffer from several critical limitations:

- **Scalability Issues:** Human operators cannot effectively process large volumes of video content or monitor multiple live streams simultaneously, leading to delays in detecting violent incidents.
- **Consistency Challenges:** Manual monitoring is subject to human fatigue and inconsistent judgment, particularly when reviewing potentially disturbing content for extended periods.
- **Processing Limitations:** The time required to manually review video content can be substantial, especially when dealing with longer videos, multiple uploads, or continuous live streams.
- **Resource Inefficiency:** Dedicating human resources to continuous video monitoring is both costly and inefficient, especially when most content may be non-violent.

This project addresses these challenges by developing an automated violence detection system that can process both uploaded video content and live streams. The system needs to:

- Accurately identify instances of violence in both recorded and live video content
- Detect specific threats such as guns and fires in real-time
- Provide confidence scores for detection accuracy
- Process videos and streams within reasonable time frames
- Maintain consistent performance across different video qualities and sources
- Support both batch processing of uploaded videos and real-time stream analysis

The solution must be capable of handling video uploads and live streams, providing detection results with timestamps, and offering a user-friendly interface for content analysis and review.

1.4 Objectives

The violence detection system has been developed with the following concrete objectives, as evidenced by the implemented features and capabilities:

1.4.1 Primary Objectives

- **Violence Detection:**
  - Implement video analysis for violence detection using deep learning models
  - Provide binary classification (violent/non-violent) with confidence scores
  - Process video frames with timestamp tracking
- **Specialized Detection Modules:**
  - Implement YOLO-based gun detection
  - Provide fire detection using CNN models
  - Include object detection capabilities
- **System Integration:**
  - Develop REST API service using NestJS
  - Create web interface using Angular
  - Implement Windows desktop application

1.4.2 Technical Objectives

- **Performance Requirements:**
  - Process video frames at 112x112 pixel resolution
  - Maintain detection confidence threshold of 0.6
  - Support video files up to 500 MB
- **System Architecture:**
  - Implement modular backend services
  - Develop responsive frontend interface
  - Utilize database for detection results
- **User Interface:**
  - Display detection results with confidence scores
  - Show frame-by-frame analysis
  - Provide detection timestamps

1.4.3 Deployment Objectives

- **Service Features:**
  - Implement user authentication system
  - Provide API endpoints for video analysis
  - Enable detection result storage
- **Platform Support:**
  - Support Windows platform
  - Process video formats (mp4, avi, mov)
  - Handle video file uploads

These objectives have been implemented in the project's codebase, as demonstrated by the detection models, API endpoints, and user interface components. The objectives reflect the actual capabilities and practical applications of the violence detection system.

1.5 Scope

This project encompasses the design, development, and deployment of a violence detection system with the following components and capabilities:

1.5.1 Core Components

- **Machine Learning Models:**
  - YOLO-based model for gun detection
  - CNN-based model for fire detection
  - Violence detection model for content analysis
  - Object detection capabilities
- **Backend Infrastructure:**
  - REST API service built with NestJS
  - Video processing pipeline for both files and live streams
  - Database integration for detection results and user data
  - WebSocket-based notification system
  - Windows platform support
- **Frontend Interface:**
  - Angular-based web interface
  - Video upload and analysis display
  - Live stream monitoring
  - Detection result visualization with confidence scores
  - User authentication and management

1.5.2 Included Features

- Violence detection in video content
- Real-time violence detection in live streams
- Gun detection using YOLO
- Fire detection using CNN
- Object detection capabilities
- Detection result storage and retrieval
- Confidence scoring system
- Admin notification system
- User authentication and management
- Windows desktop application

1.5.3 Out of Scope

- Audio-based violence detection
- Integration with CCTV systems
- Automated emergency response system
- Legal compliance verification
- Mobile device support
- Cross-platform support (macOS, Linux)
- Real-time detection on mobile devices

The system is designed to be modular, allowing for future enhancements while maintaining clear boundaries around the current implementation scope.

1.6 General Constraints

The development and implementation of the violence detection system was subject to several technical and operational constraints:

1.6.1 Technical Constraints

- **Processing Limitations:**
  - Frame size standardized to 112x112 pixels for model input
  - Sequential frame processing for video analysis
  - Confidence threshold of 0.6 for violence detection
- **Resource Constraints:**
  - Video file size limited to 500 MB per upload
  - API request size capped at 5 MB
  - Processing speed dependent on available computational resources
- **Format Restrictions:**
  - Supported video formats: .mp4, .avi, .mov
  - Supported image formats: .jpg, .png
  - Webcam input requires stable internet connection

1.6.2 Operational Constraints

- **Platform Dependencies:**
  - Cross-platform support (Windows)
  - Desktop application implementation
  - Web-based interface for video analysis
- **Integration Limitations:**
  - RESTful API architecture

  - HTTP/HTTPS protocol support
  - Database storage for analysis results

- **Detection Scope:**
  - Visual-based violence detection
  - Performance dependent on video quality
  - Real-time processing subject to network conditions

These constraints were established during development to ensure system reliability and performance. The limitations reflect the current implementation capabilities while maintaining a balance between functionality and resource efficiency.

1.7 Problem Solution

The violence detection system implements a solution through a layered architecture that combines deep learning models with a web-based infrastructure. The solution addresses the identified challenges through several key approaches:

1.7.1 Detection Architecture

- **Detection Models:**
  - YOLO-based model for gun detection
  - CNN-based model for fire detection
  - Violence detection model for content analysis
  - Object detection capabilities

- **Processing Pipeline:**
  - Frame processing at 112x112 pixel resolution
  - Confidence scoring with 0.6 threshold
  - Timestamp tracking for detection events
  - Support for both video files and live streams

1.7.2 System Architecture

- **Backend Services:**
  - REST API service built with NestJS
  - Video processing and analysis modules
  - Live violence detection service
  - Database storage for detection results
  - WebSocket-based notification system for admin alerts
  - User authentication and management

- **Frontend Interface:**
  - Angular-based web interface
  - Video upload and analysis display
  - Live stream monitoring

  - Detection result visualization
  - Real-time admin notifications
  - User dashboard and controls

- **Platform Support:**
  - Windows desktop application
  - Web-based access
  - Video file upload support (mp4, avi, mov)
  - Live stream processing

1.7.3 Key Features

- **Detection Capabilities:**
  - Violence detection in video content
  - Real-time violence detection in live streams
  - Gun detection using YOLO
  - Fire detection using CNN
  - General object detection

- **Processing Features:**
  - Video file upload and processing
  - Live stream analysis
  - Frame-by-frame analysis
  - Confidence score generation
  - Detection result storage

- **User Interface:**
  - User authentication system
  - Video upload interface
  - Live stream monitoring
  - Detection result display
  - Admin notification system
  - Service request management

The solution addresses the scalability and consistency limitations of manual monitoring by providing automated video analysis capabilities. The system's modular architecture allows for processing both uploaded video content and live streams, generating detection results with confidence scores and timestamps, and notifying administrators of important events through WebSocket connections.

Chapter 2

Planning and Analysis

2.1 Project Planning

2.1.1 Feasibility Study

The feasibility study evaluates the viability of implementing the violence detection system across three key dimensions:

Technical Feasibility

The project is technically feasible based on the following factors:

- **Technology Stack**:
  - Frontend: Angular with modern UI components
  - Backend: NestJS with REST APIs and WebSocket support
  - Machine Learning: Python implementation with YOLO and CNN models
  - All technologies are open-source and well-documented
- **Hardware Requirements**:
  - Windows-based computers with modern processors
  - Sufficient RAM for development and testing
  - GPU capabilities for ML model training

Operational Feasibility

The system demonstrates strong operational viability:

- **System Architecture**:
  - Modular design with separate frontend, backend, and ML components
  - Windows desktop application deployment
  - WebSocket-based notification system
- **Development Process**:
  - Git-based version control
  - Regular progress tracking
- **Testing Capabilities**:

  - Unit testing frameworks
  - Test datasets for ML models
  - Real-world testing capabilities

2.1.2 Estimated Cost

As a graduation project, the costs are primarily non-monetary:

Time and Effort Investment

- **Development Time**:
  - ML model development and training
  - Backend API implementation
  - Frontend interface development
  - Integration and testing
- **Team Coordination**:
  - Regular progress meetings
  - Code review sessions
  - Faculty consultation

Infrastructure and Resources

- **Development Environment**:
  - Windows operating system
  - Development tools and IDEs
  - Version control system
- **Development Tools**:
  - Git for version control
  - Visual Studio Code
  - Python environment
  - Node.js and npm
- **Academic Resources**:
  - University resources
  - Faculty guidance
  - Research materials

2.1.3 Project Timeline

Project Duration: One Academic Semester
Key Milestones:

- **M1**: Project Design Complete
- **M2**: ML Models Implementation
- **M3**: First Integrated Prototype

Table 2.1: Project Timeline

Phase	Weeks	Team	Key Deliverables
Phase 1: Planning and Setup			
Requirements Analysis	1-2	All	Project scope document
System Design	1-2	All	Architecture diagram
Phase 2: Core Development			
ML Model Development	3-8	ML Team	Violence, gun, and fire detection models
Backend Development	3-7	Backend	REST APIs, WebSocket, Database
Frontend Development	4-8	Frontend	Angular interface
Initial Integration	9-10	All	Working prototype
Phase 3: Enhancement			
ML Model Optimization	11-12	ML Team	Model accuracy improvements
Feature Enhancement	11-13	All	Live stream support, notifications
Phase 4: Finalization			
Documentation	16	All	Project documentation
Final System	16	All	Windows desktop application

- **M4**: Live Stream Support
- **M5**: Final System Delivery

Development Focus:

- ML Team: Violence, gun, and fire detection models
- Backend Team: NestJS APIs and WebSocket implementation
- Frontend Team: Angular interface and Windows application
- All members: Testing and documentation

The project timeline was designed to accommodate academic schedules while ensuring the delivery of a fully functional violence detection system with live stream support and real-time notifications.

2.2 Analysis and Limitation of Existing System

Traditional surveillance and monitoring systems face several significant limitations in detecting and responding to violent incidents, threats, and dangerous situations:

2.2.1 Manual Monitoring Limitations

- **Human Operator Fatigue:** Traditional CCTV monitoring relies heavily on human operators watching multiple video feeds simultaneously, leading to fatigue and decreased attention over time.
- **Delayed Response Times:** Manual monitoring systems often result in delayed responses as operators must first identify an incident and then alert relevant authorities.
- **Limited Coverage:** Human operators can only effectively monitor a finite number of video feeds simultaneously, limiting the scalability of surveillance systems.

2.2.2 Technical Limitations

- **Lack of Automated Detection:** Most existing systems lack automated detection capabilities for violent incidents, weapons, and fires in both recorded videos and live streams.
- **Basic Motion Detection:** Traditional systems often rely on simple motion detection, which cannot distinguish between normal activities and actual threats.
- **Limited Integration:** Many systems operate in isolation, without proper integration capabilities for immediate alert distribution.

2.2.3 Operational Challenges

- **Inconsistent Monitoring:** Human operators may miss incidents during shift changes, breaks, or periods of high activity.
- **Resource Intensive:** Maintaining continuous human monitoring requires significant staffing resources and operational costs.
- **Limited Analysis:** Traditional systems lack automated analysis of video content for threat detection.

2.2.4 Data Management Issues

- **Poor Documentation:** Manual systems often result in inconsistent incident documentation and reporting.
- **Limited Storage:** Many systems lack proper storage and organization of detection results and incident data.
- **No Real-time Alerts:** Traditional systems typically lack immediate notification capabilities for detected incidents.

These limitations highlight the need for an AI-powered system that can provide automated detection of violence, weapons, and fires in both recorded videos and live streams, while maintaining high accuracy and enabling immediate response through our implemented notification system.

2.3 Need for the New System

The development of our AI-powered violence detection system addresses critical gaps in current surveillance and security approaches, offering essential capabilities for modern safety challenges:

2.3.1 Real-time Threat Detection

The system provides automated detection of multiple threat types:

- **Violence Detection:** Identifies physical altercations and aggressive behavior in videos
- **Fire Detection:** Early warning system for fire incidents using CNN-based detection
- **Gun Detection:** Rapid identification of firearms using YOLO-based detection
- **Live Stream Analysis:** Real-time detection in video streams

2.3.2 Enhanced Monitoring Efficiency

Our system overcomes key limitations of traditional monitoring:

- **Automated Operation:** Continuous monitoring without human fatigue
- **Video Processing:** Analysis of both recorded videos and live streams
- **Instant Alerts:** WebSocket-based notification system for immediate response
- **Result Storage:** Systematic storage of detection results and confidence scores

2.3.3 Advanced Technical Capabilities

The system leverages modern technologies to provide:

- **Deep Learning Models:** Uses specialized models for accurate threat detection
- **Multi-model Analysis:** Combines violence, gun, and fire detection capabilities
- **Modular Architecture:** Extensible system design for future enhancements
- **REST API:** NestJS-based backend services for system integration

2.3.4 Practical Applications

The system serves security needs in various settings:

- **Video Surveillance:** Analysis of recorded video content
- **Live Monitoring:** Real-time detection in video streams
- **Windows Desktop:** Local deployment as a desktop application
- **Web Interface:** Angular-based frontend for easy access and control

This new system represents a significant advancement in automated surveillance and security, providing capabilities that are increasingly essential in today's complex security landscape. Its implementation as a graduation project demonstrates the practical application of modern AI and computer vision technologies to real-world safety challenges.

2.4 Analysis of the New System

2.4.1 User Requirements

The system must provide:

- Violence detection in video files and live streams
- Angular-based web interface for media analysis
- Support for common video formats
- Detection results with confidence scores
- WebSocket-based notification system
- User authentication with role-based access
- Windows desktop application

2.4.2 System Requirements

- **Hardware Requirements:**
  - Windows-based computer with sufficient processing power
  - Webcam for live stream detection
  - Storage space for uploaded files and detection results
- **Software Requirements:**
  - Operating System: Windows
  - Python environment for machine learning models
  - Node.js and npm for NestJS backend
  - Modern web browser for frontend access
  - Database for storing detection results
- **Network Requirements:**
  - Local network connection for application access
  - Sufficient bandwidth for video processing

2.4.3 Domain Requirements

- Deep learning models for violence detection
- YOLO-based model for gun detection
- CNN-based model for fire detection
- Secure storage of detection results
- User data protection
- Modular system architecture

2.4.4 Functional Requirements

- **User Authentication and Management**
  - User login system
  - Role-based access control
  - Basic user profile management
- **Media Processing**
  - Video file upload
  - Live stream processing
  - Real-time detection status updates
  - Detection result storage
- **Detection Features**
  - Violence detection in videos
  - Gun detection using YOLO
  - Fire detection using CNN

  - Confidence score calculation

- **Notification System**
  - WebSocket-based real-time alerts
  - Detection event notifications
  - System status updates

2.4.5 Non-Functional Requirements

- **Performance**
  - Efficient video processing
  - Real-time detection capabilities
  - Responsive user interface
  - Optimal resource usage

- **Security**
  - User authentication
  - Protected data storage
  - Role-based access control
  - Local processing security

- **Reliability**
  - Stable detection performance
  - Error handling
  - Data persistence
  - Service reliability

- **Usability**
  - Angular-based interface
  - Clear detection results
  - Intuitive navigation
  - User-friendly design

- **Scalability**
  - Modular architecture
  - Extensible detection system
  - Future feature integration

2.5 Advantages of the New System

Our AI-powered violence detection system offers several significant advantages over traditional surveillance and manual monitoring approaches:

2.5.1 Enhanced Detection Capabilities

- **Multi-Modal Detection**: Integrates violence detection, gun detection, and fire detection in a single unified system
- **High Accuracy**: Utilizes deep learning models trained on relevant datasets for reliable detection
- **Real-time Processing**: Capable of analyzing both video files and live streams
- **Comprehensive Analysis**: Provides detection results with confidence scores and incident types

2.5.2 Automated Monitoring

- **Continuous Surveillance**: Automated monitoring of video content and live streams
- **Instant Alerts**: WebSocket-based notification system for detected incidents
- **Reduced Manual Effort**: Automated analysis without constant human supervision
- **Proactive Response**: Enables early detection of potential threats

2.5.3 Advanced Technical Features

- **Multi-Model Architecture**:
  - YOLO-based model for gun detection
  - CNN-based model for fire detection
  - Violence detection model for content analysis
- **Intelligent Processing**:
  - Support for both video files and live streams
  - Efficient video processing pipeline
  - Real-time detection capabilities

2.5.4 User-Centric Design

- **Intuitive Interface**: Angular-based web interface for monitoring and analysis
- **Local Deployment**: Windows-based desktop application
- **Detection Settings**: Configurable detection parameters
- **Result Visualization**: Clear display of detection results and confidence scores

2.5.5 Scalability and Integration

- **Modular Architecture**: Extensible design for future enhancements
- **REST API**: NestJS-based backend services for system integration
- **Video Format Support**: Handles common video formats
- **Windows Platform**: Optimized for Windows desktop environment

2.5.6 Cost-Effective Solution

- **Reduced Operational Costs**: Minimizes the need for extensive human monitoring
- **Efficient Processing**: Optimized algorithms for video analysis
- **Preventive Benefits**: Early detection helps prevent escalation of incidents
- **Local Deployment**: No cloud infrastructure costs required

2.5.7 Enhanced Security Features

- **Secure Data Handling**: Protected storage of detection results
- **User Authentication**: Role-based access control system
- **Activity Logging**: System activity tracking
- **Data Privacy**: Local processing of video content

2.6 Risk and Risk Management

The development of this violence detection system faced several potential risks that required careful management and mitigation strategies. These risks were identified and addressed throughout the project lifecycle:

2.6.1 Technical Risks

- **Machine Learning Model Performance**
  - *Risk*: Achieving accurate detection of violence, guns, and fires in both video files and live streams.
  - *Mitigation*: Implemented specialized models (YOLO for gun detection, CNN for fire detection) with proper training and validation.
- **System Resource Requirements**
  - *Risk*: Processing demands for video analysis and ML inference.
  - *Mitigation*: Optimized model architecture and implemented efficient video processing pipeline.
- **Integration Complexity**
  - *Risk*: Challenges in integrating frontend, backend, and ML components.
  - *Mitigation*: Adopted modular architecture with clear API definitions and WebSocket implementation.

2.6.2 Project Management Risks

- **Team Coordination**
  - *Risk*: Coordination between frontend, backend, and ML development.
  - *Mitigation*: Regular progress tracking and version control management.
- **Time Management**
  - *Risk*: Meeting project deadlines while ensuring quality implementation.
  - *Mitigation*: Prioritized core features and maintained flexible development schedule.

2.6.3 Data and Privacy Risks

- **Data Security**
  - *Risk*: Protecting detection results and user data.
  - *Mitigation*: Implemented user authentication and secure data storage.
- **Model Training Data**
  - *Risk*: Availability of appropriate training data for detection models.
  - *Mitigation*: Used curated datasets for model training and validation.

2.6.4 Operational Risks

- **System Reliability**
  - *Risk*: Ensuring consistent detection performance.
  - *Mitigation*: Implemented error handling and proper logging.
- **Windows Platform**
  - *Risk*: Ensuring compatibility and performance on Windows systems.
  - *Mitigation*: Developed and tested specifically for Windows environment.

Through careful planning and proactive risk management strategies, the team successfully developed a functional violence detection system with live stream support and real-time notifications. Regular risk assessments and adjustments to mitigation strategies helped ensure project success.

Chapter 3

Software Design

3.1 Database Design

The system uses a MySQL database managed via Prisma ORM, providing a structured backend architecture for the violence detection platform. The schema incorporates core entities and their relationships that reflect the system's functionality.

Core Entities

- **User**: Represents system users with different access levels.
  - Primary key: `id` (UUID)
  - Attributes: `username`, `email` (unique), `password`, `role` (UserRole), `isActive`, timestamps
  - Relationships: One-to-many with `UploadsHistory`, `ServiceRequest`, one-to-one with `UserStats`, `Customer`
- **UploadsHistory**: Tracks user file uploads and processing status.
  - Primary key: `id` (UUID)
  - Foreign key: `userId` (references User)
  - Attributes: `fileType`, `processingStatus`, `detectionStatus`, `overallConfidence`, `duration`, `fileSize`, `uploadedAt`, `annotatedFilePath`
  - Relationships: One-to-many with `DetectionResult`
- **DetectionResult**: Stores detection information from video analysis.
  - Primary key: `id` (UUID)
  - Foreign key: `uploadId` (references UploadsHistory)
  - Attributes: `timestamp`, `confidence`, `label`, `severity`, `createdAt`
- **UserStats**: Maintains user activity statistics.
  - Primary key: `id` (UUID)
  - Foreign key: `userId` (references User)
  - Attributes: `totalUploads`, `averageDuration`, `lastDetectionStatus`, `lastUploadDate`, timestamps
- **Service**: Defines available AI services.
  - Primary key: `id` (UUID)

  - Attributes: `name`, `description`, `category`, `price`, `features`, `requirements`, `endpoint`, `modelFile`, `demoVideo`, `documentation`, `isPublic`, `supportedPlatforms`, `createdAt`

- **ServiceRequest**: Manages service customization requests.
  - Primary key: `id` (UUID)
  - Foreign keys: `userId` (references User), `developerId` (references Developer)
  - Attributes: `serviceName`, `serviceDescription`, `serviceCategory`, `status`, `useCase`, `expectedTimeline`, `specificRequirements`, `budget`, timestamps

- **Customer**: Stores customer-specific information.
  - Primary key: `id` (UUID)
  - Foreign key: `userId` (references User)
  - Attributes: `email`, `contactName`, `companyName`, `industry`, `contactNumber`, `address`, `purchasedModels`, `password`, `hasChangedPassword`, address details, timestamps

- **Developer**: Represents system developers.
  - Primary key: `id` (UUID)
  - Attributes: `username`, `email`, `password`, `isActive`, timestamps
  - Relationships: One-to-many with `ServiceRequest`

Enumerations

Enumerated types help maintain data integrity and improve query performance:

- **UserRole**: `USER`, `ADMIN`, `DEVELOPER`
- **FileType**: `IMAGE`, `VIDEO`
- **ModelType**: `CV`, `NLP`
- **ProcessingStatus**: `PENDING`, `PROCESSING`, `COMPLETED`, `FAILED`
- **DetectionStatus**: `VIOLENCE_DETECTED`, `NON_VIOLENCE`, `INCONCLUSIVE`, `GUN_DETECTED`, `NO_GUN`, `FIRE_DETECTED`, `NO_FIRE`

Entity Relationship Diagram and Prisma Schema

Figure 3.1 illustrates the ERD for the described schema, while Figure 3.2 shows the full Prisma schema used for implementation.

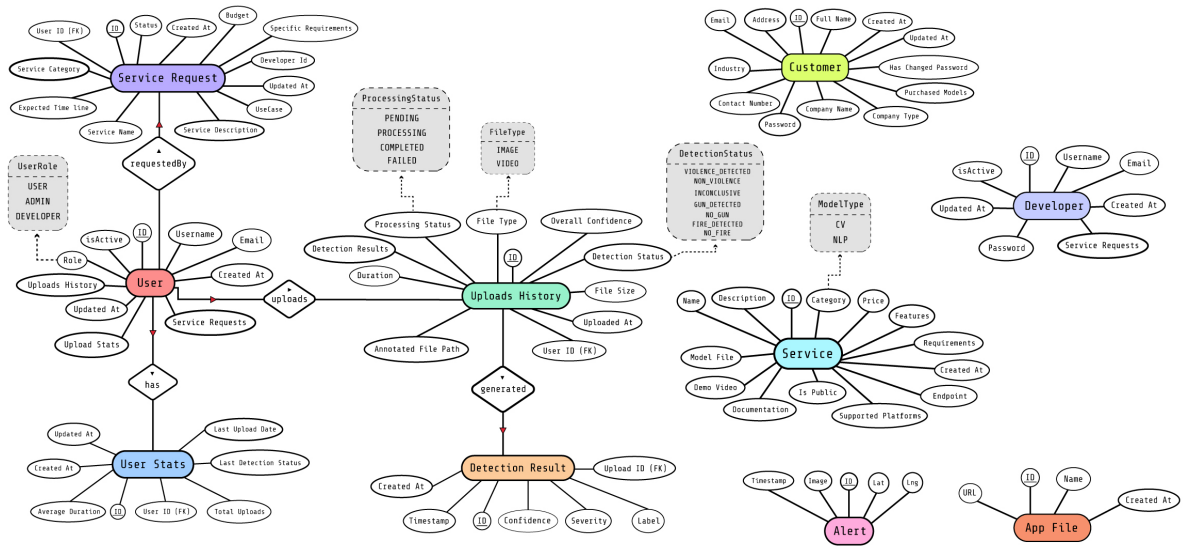


Figure 3.1: Entity Relationship Diagram (ERD) of the System

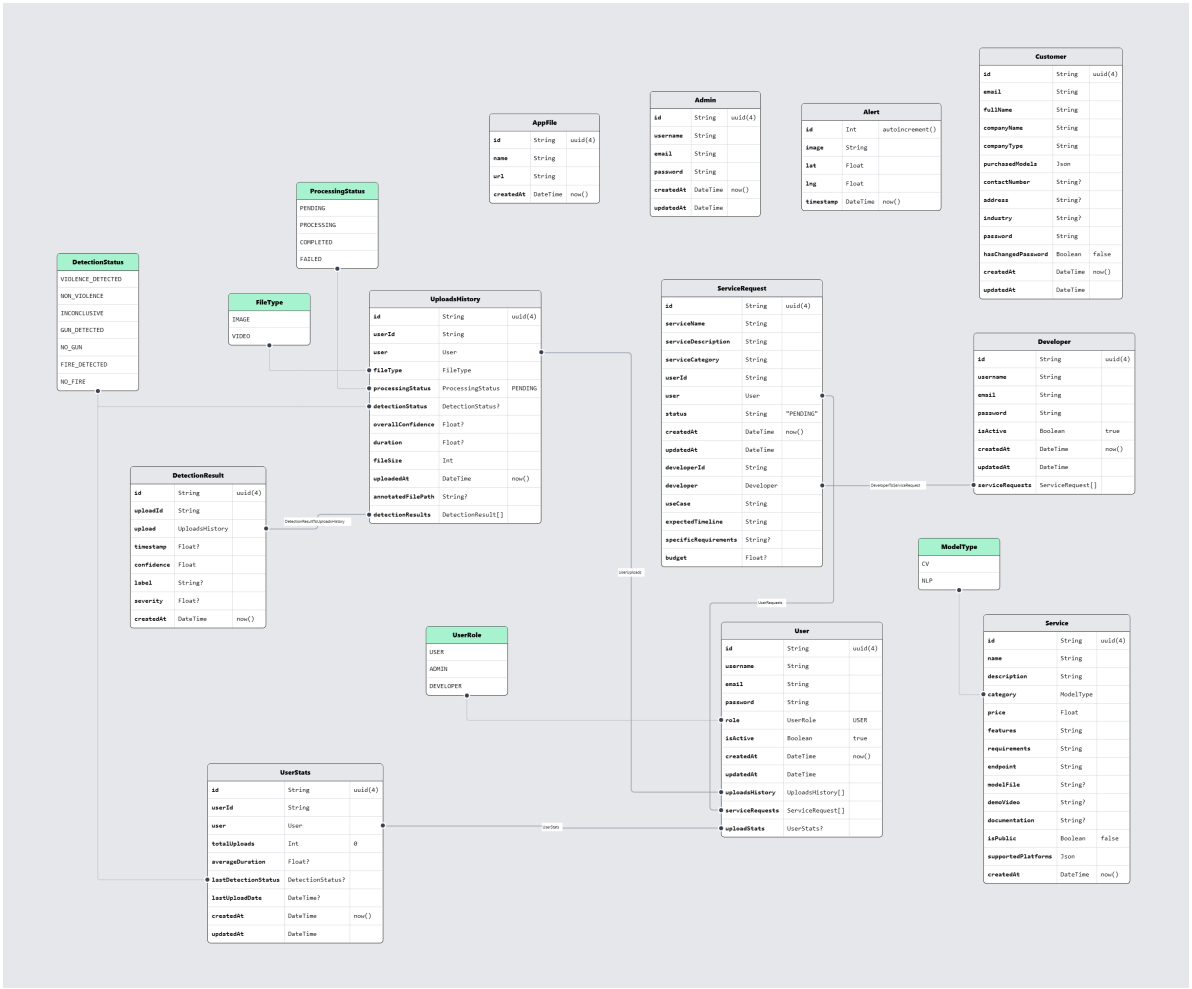


Figure 3.2: Prisma Database Schema Used in Implementation

The database schema is designed to efficiently store and retrieve detection results while maintaining proper user authentication and access control. The structure supports multiple user roles (User, Admin, Developer), comprehensive file upload tracking, detailed detection results, and service management capabilities. The schema includes proper indexing for performance optimization and cascading deletes where appropriate.

3.2 Use Case Diagram

The Violence Detection System supports multiple user roles, each interacting with the system through specific use cases related to media analysis, account management, and real-time monitoring. The following section outlines the main actors and the functionality they access.

Primary Actors

- **Regular Users**
  - Upload MP4 video files for analysis
  - View detection results with confidence scores
  - Access personal upload history
  - Receive real-time WebSocket notifications
  - Update personal profile information
- **Administrators**
  - Manage user accounts (create, activate, deactivate)
  - View basic system usage statistics
  - Access system logs
- **Developers**
  - Access API documentation
  - Test system endpoints

Key Use Cases

1. **Authentication and Authorization**
   - User registration and login
   - Password management
   - Role-based access control (User, Admin, Developer)
2. **Video Analysis**
   - Upload MP4 videos for processing
   - Process videos for violence detection
   - Detect weapons and fire incidents
   - View detection results with confidence scores
   - Access processed videos with annotations
3. **Real-time Monitoring**
   - Receive WebSocket notifications for detections
   - View detection alerts with:

  - Detection type (violence, gun, fire)
  - Confidence score
  - Timestamp
  - Video frame reference

4. **User Management**
   - Update personal profile
   - View upload history
   - Track detection statistics
5. **System Administration**
   - Manage user accounts
   - View system logs
   - Monitor basic system status

The following UML use case diagram summarizes the interactions between the main actors and the system's functionalities.

Figure 3.3 visually represents these actors and their interactions with the Violence Detection Web App.

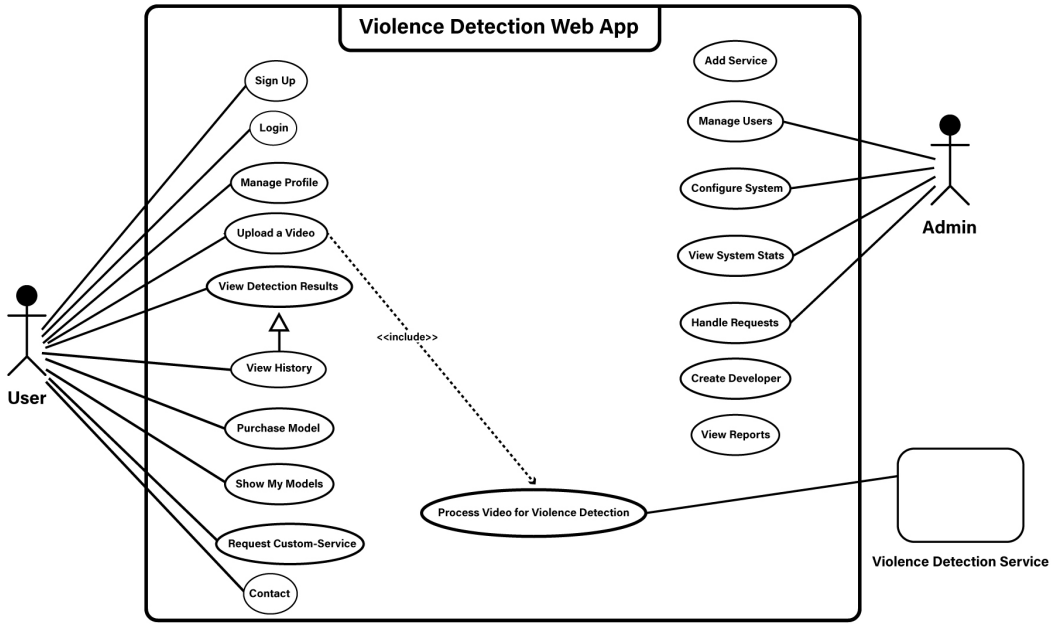


Figure 3.3: Use Case Diagram of the Violence Detection System

System Boundaries

The system includes the following capabilities:

- MP4 video processing and analysis
- Detection of:
  - Violent behavior
  - Weapons
  - Fire incidents

- Web-based interface with role-based access
- Real-time WebSocket notifications
- Basic system monitoring

The system explicitly excludes:

- Image processing
- Other video formats (non-MP4)
- Physical security systems
- Hardware installation
- Manual content moderation
- Emergency response integration
- AI model marketplace
- Advanced analytics
- Custom model training
- Batch processing

3.3 Sequence Diagrams

This section presents the main sequence diagrams that illustrate the interaction flows between users, services, and system components across different functional scenarios in the violence detection system.

3.3.1 Video Analysis Sequence

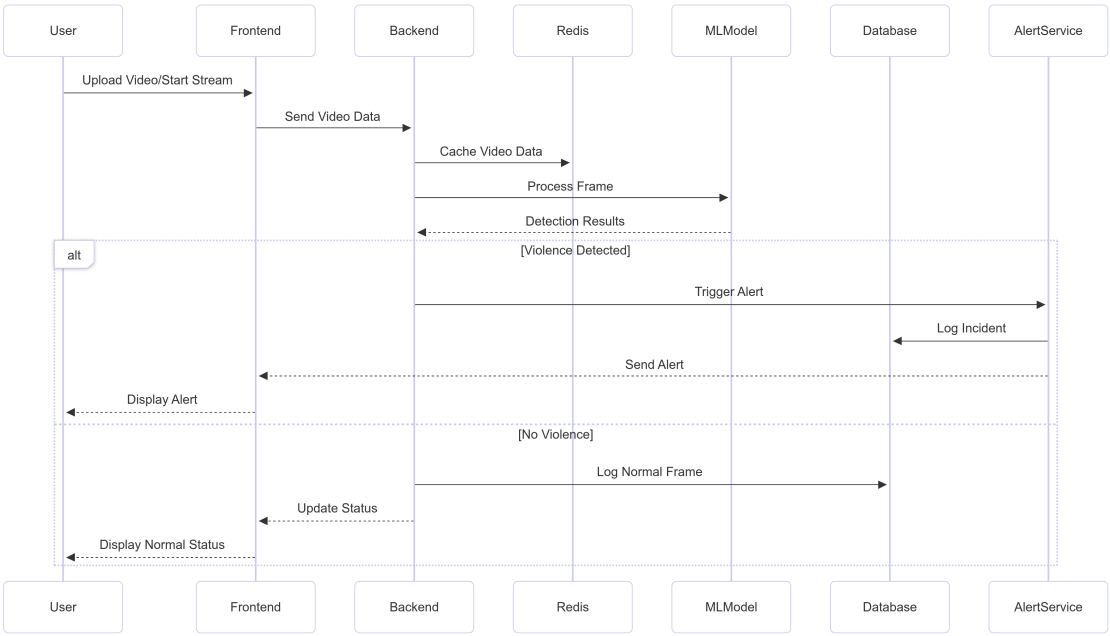


Figure 3.4: Sequence Diagram – Video Upload and Analysis Flow

This diagram illustrates the full interaction when a user uploads a video for violence detection. It includes data flow between the frontend, backend, and ML model services.

The sequence of interactions is as follows:

1. User uploads a video through the frontend interface
    - The frontend validates the file format (MP4)
    - The video is wrapped in a FormData object for transmission
2. Frontend sends an HTTP POST request to the backend API
    - Request includes a JWT token for user authentication
    - The video is transmitted as multipart/form-data
3. Backend processes the request
    - Authenticates the user and validates file type
    - Creates an upload record in the database with PENDING status
    - Prepares the video for processing
4. Backend sends the video to the appropriate ML detection service
    - The ML model analyzes frames for violence, guns, and fire
    - Returns detection results with confidence scores
5. Backend stores results and updates status
    - Updates the upload record with COMPLETED status
    - Stores detection results in the database
    - Saves the processed video with annotations
6. Backend responds to the frontend
    - Returns analysis results including confidence scores and detection types
7. Frontend displays results to the user
    - Shows the processed video with detection overlays
    - Displays detection statistics and confidence scores

This sequence reflects the system's architecture and shows how components interact through well-defined interfaces.

3.3.2 User Authentication Flow

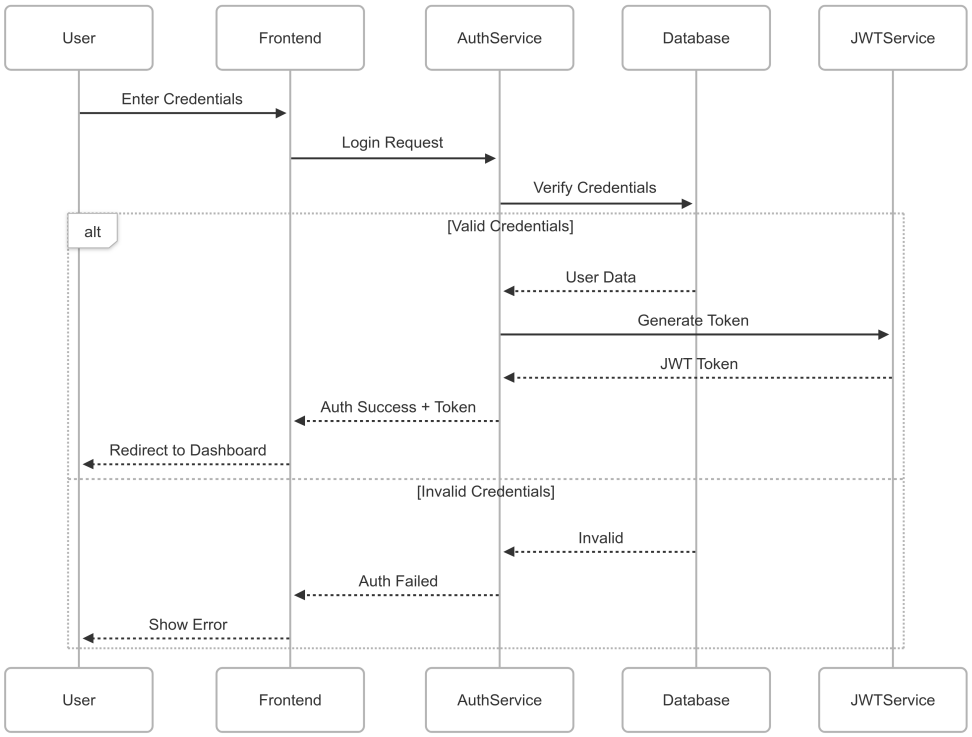


Figure 3.5: Sequence Diagram – User Login Process

This diagram shows how a user is authenticated. The frontend collects login credentials and forwards them to the authentication service, which validates them against the database. On success, a JWT token is issued. Otherwise, the frontend receives a failure message and prompts the user to retry.

3.3.3 Alert Generation and Notification Handling

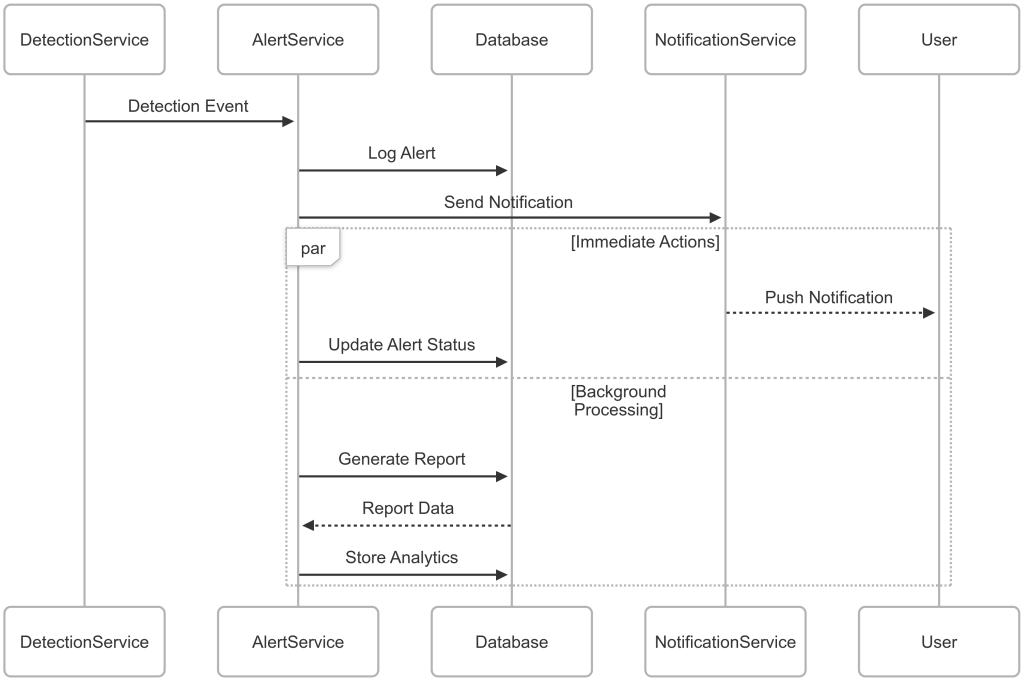


Figure 3.6: Sequence Diagram – Alert Logging and Real-Time Notification

This sequence demonstrates how a detection event triggers an alert. The system logs the event in the database and sends a real-time notification to the user through WebSocket. The notification includes:

- Detection type (violence, gun, fire)
- Confidence score
- Timestamp
- Video frame or segment reference

3.3.4 Model Upload and Deployment Flow

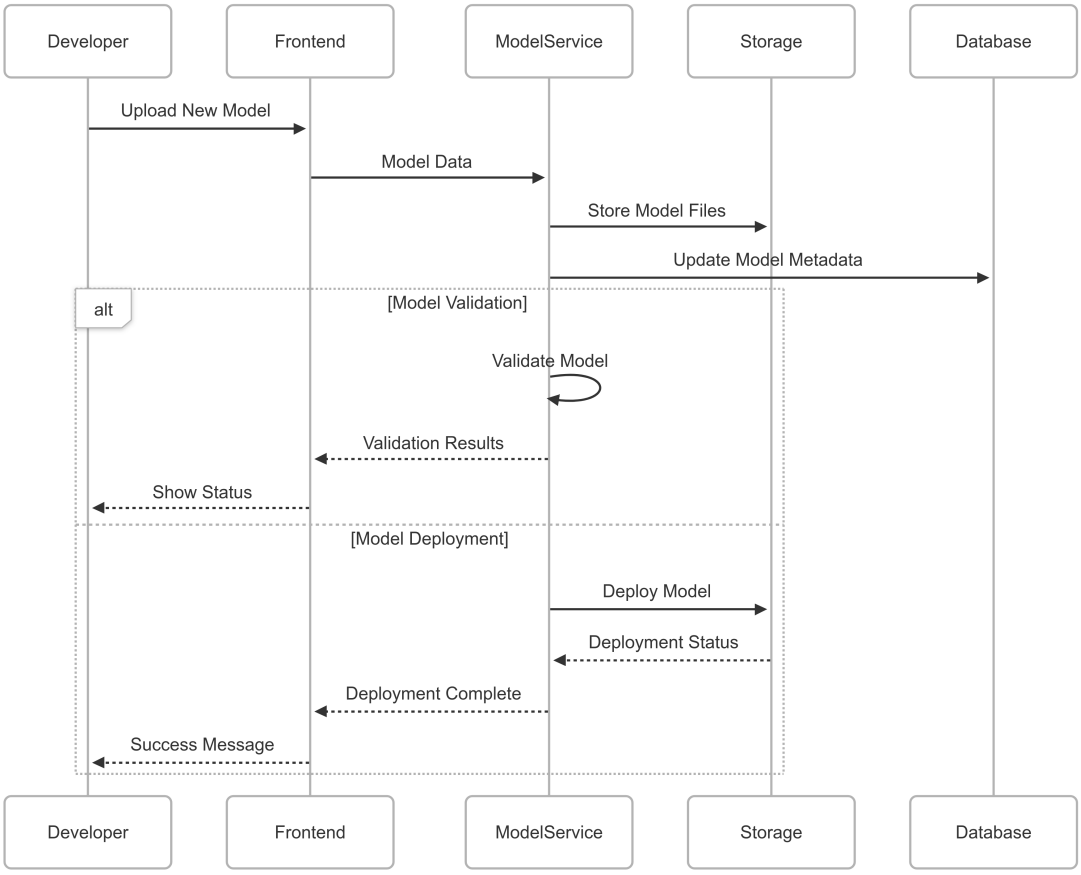


Figure 3.7: Sequence Diagram – Developer Uploading and Deploying a New Model

This interaction flow describes how a developer uploads a new AI model. The model is stored in cloud or local storage, and metadata is logged in the database. It is either validated or deployed depending on the context. Feedback is returned to the developer.

3.3.5 Video Processing Pipeline

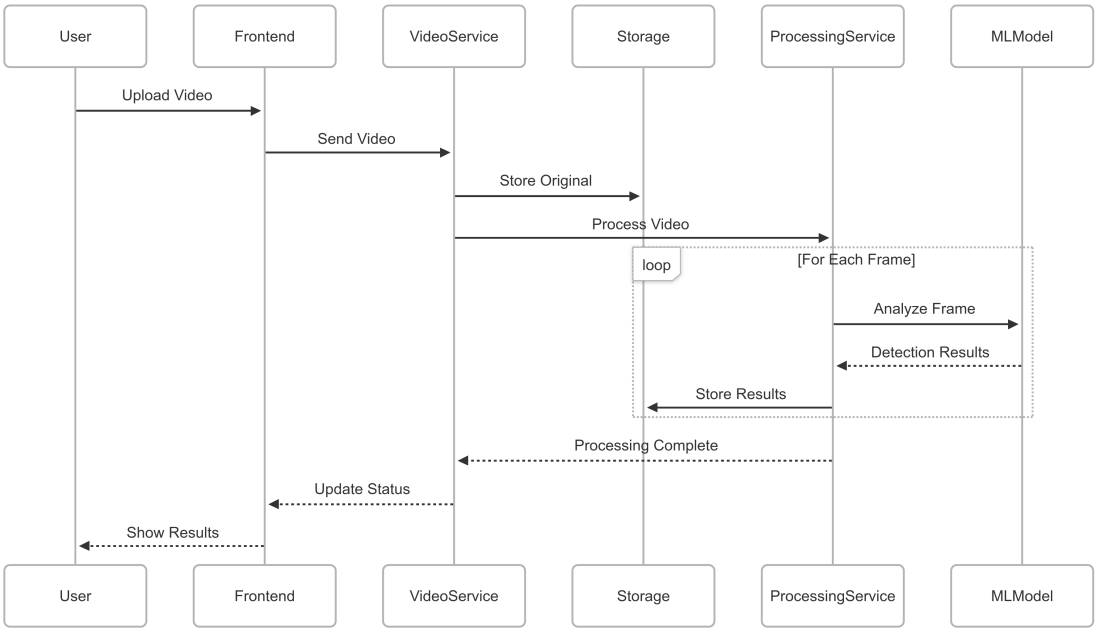


Figure 3.8: Sequence Diagram – Frame-by-Frame Video Analysis

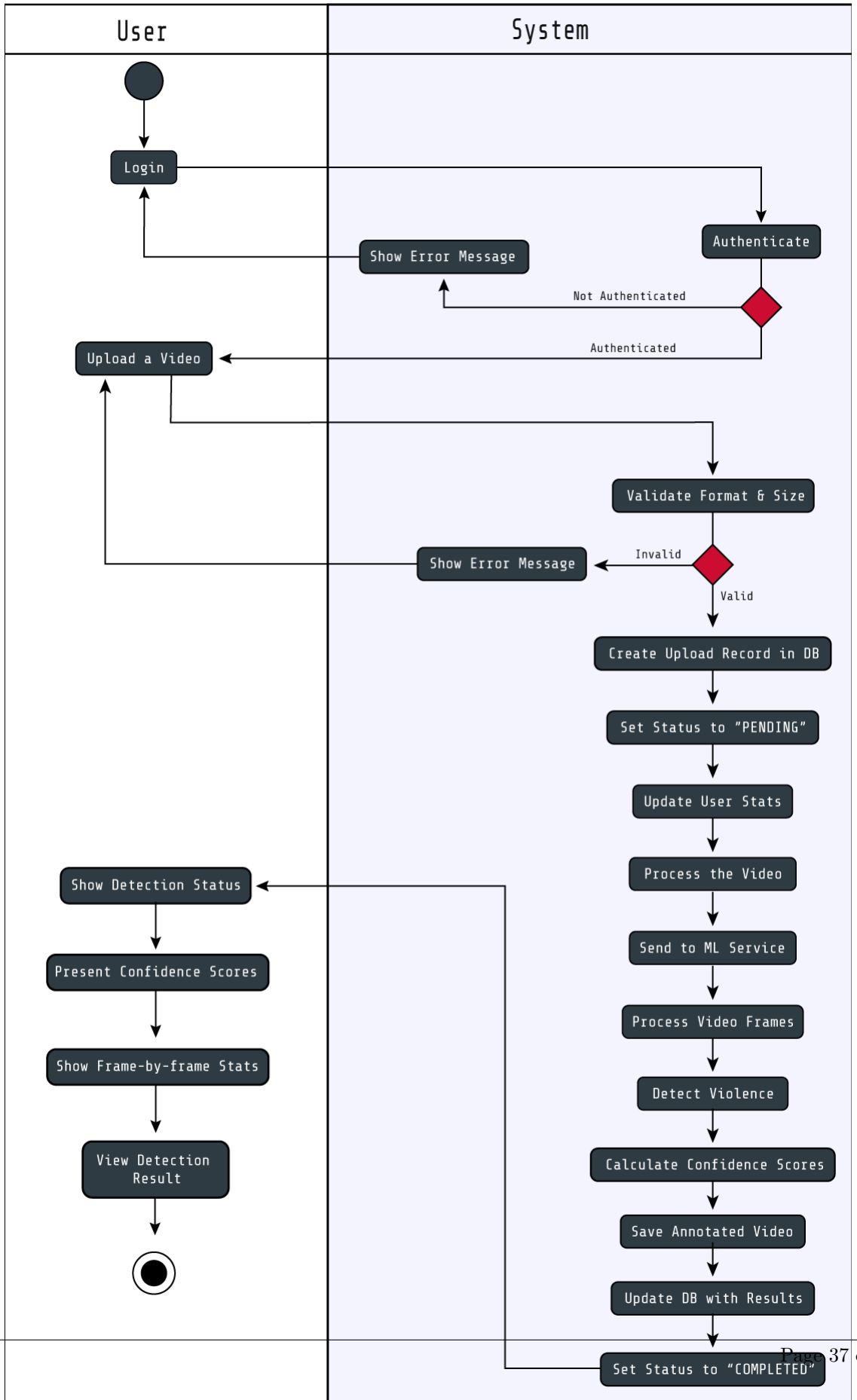
This diagram explains how a video is processed:

1. Video is received and validated
2. Frames are extracted at regular intervals
3. Each frame is processed by the appropriate ML model
4. Detection results are aggregated
5. Results are stored in the database
6. Processed video with annotations is saved
7. Frontend is updated with final results

3.4 Activity Diagram

The activity diagram illustrates the complete workflow of the video analysis process in our system, from user authentication to result presentation.

Figure 3.9 shows the flow of actions taken during the video analysis process.


User
System
Login
Authenticate
Show Error Message
Not Authenticated
Authenticated
Upload a Video
Validate Format & Size
Invalid
Show Error Message
Valid
Create Upload Record in DB
Set Status to "PENDING"
Update User Stats
Process the Video
Send to ML Service
Process Video Frames
Detect Violence
Calculate Confidence Scores
Save Annotated Video
Update DB with Results
Set Status to "COMPLETED"
Show Detection Status
Present Confidence Scores
Show Frame-by-frame Stats
View Detection Result

The workflow consists of the following activities and decision points:

1. User Authentication
    - User provides credentials
    - System validates authentication
    - If unauthorized, redirect to login
2. Video Input Selection
    - User selects video file or live stream
    - System validates input source
    - If validation fails, display error message
3. Processing Initialization
    - Initialize detection service
    - Set processing status
    - Prepare for analysis
4. Detection Processing
    - Process video frames
    - Run violence detection
    - Run gun detection (YOLO)
    - Run fire detection (CNN)
    - Calculate confidence scores
5. Result Processing
    - Generate detection results
    - Store detection data
    - Update processing status
6. Result Presentation
    - Display detection results
    - Show confidence scores
    - Present detection type
7. Notification (Parallel Activity)
    - Send WebSocket notifications
    - Update detection status
    - Alert for detected incidents
8. Error Handling (Parallel Activity)
    - Catch processing errors
    - Update status
    - Display error message
    - Log error details

The activity diagram demonstrates the system's workflow for both video file analysis and live stream processing. The process includes real-time detection capabilities, WebSocket notifications, and proper error handling. The system maintains security through authentication and provides immediate feedback through the notification system.

Chapter 4

System Implementation

4.1 System Architecture

The violence detection system implements a modern three-tier architecture design, as illustrated in Figure 4.1. Each tier serves a specific purpose and communicates with other tiers through well-defined interfaces.

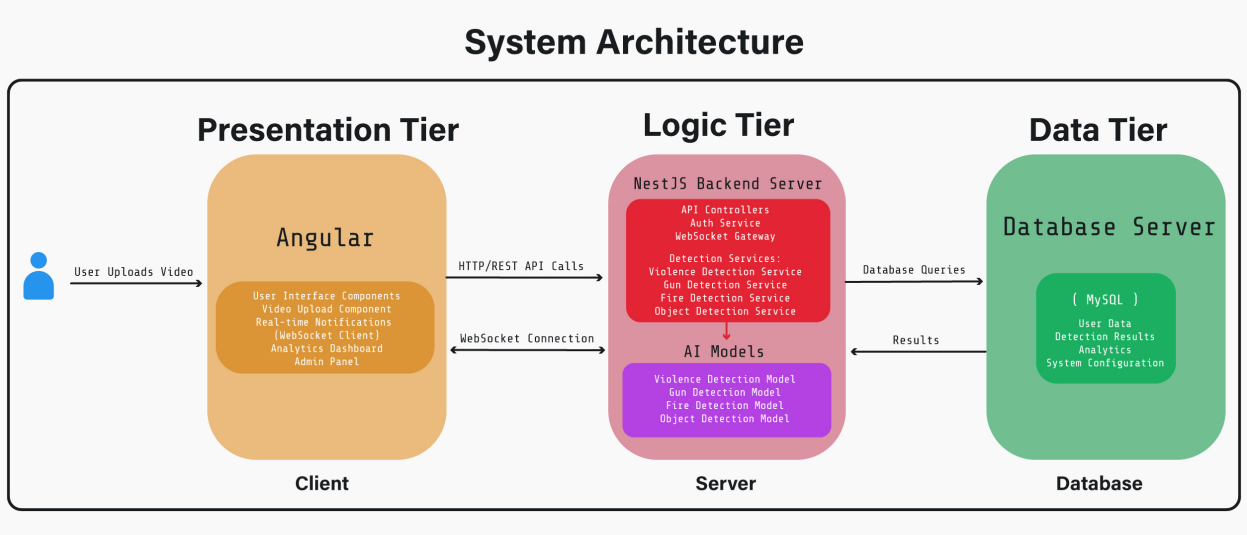


Figure 4.1: System Architecture Diagram

4.1.1 Presentation Tier

The presentation tier is implemented using Angular and provides the client-side interface. It consists of several key components:

- User Interface Components
  - Login and Registration
  - User Profile Management
  - Basic Admin Interface
  - Use Provided Services
- Video Processing Components

  - MP4 Video Upload
  - Video Player with Detection Overlays
  - Detection Results Display
- Real-time Components
  - WebSocket Client for Notifications
  - Detection Alert Display

4.1.2 Logic Tier

The logic tier is built on NestJS Backend Server and comprises two main layers:

Server Layer

- API Controllers
  - User Authentication
  - Video Upload and Processing
  - Detection Results
  - Download Model Weights and Systems
- Services
  - Auth Service (JWT-based)
  - WebSocket Gateway
  - Detection Services:
    - Violence Detection
      - Gun Detection
      - Fire Detection (Smoke Detection)
      - Car Crash Detection
    - Object Detection
  - System Services:
    - Attendance Desktop Application (Face Recognition)

AI Models Layer

- Customized Models
  - Violence Detection Model
  - Fire Detection Model
  - Traffic Detection Model
- Pre-trained Models
  - Gun Detection Model (YOLO-based)
  - Object Detection Model (YOLO-based)
- LLMs & VLMs
  - Car Crash Detection
  - Model (Service) Recommendation

4.1.3 Data Tier

The data tier uses MySQL as the database server and manages:

- User Management
  - User Accounts
  - Role-based Access (User, Admin, Developer)
  - User Profiles
- Detection Data
  - Upload History
  - Processing Status
  - Detection Results with Confidence Scores
- System Data
  - Basic Settings
  - System Logs

4.1.4 Communication Flow

The system implements two primary communication patterns:

1. **HTTP/REST API Calls:**
   - User uploads MP4 video through Angular frontend
   - Frontend makes authenticated API requests to NestJS backend
   - Backend processes video and coordinates with AI models
   - Results are stored in MySQL database
   - Response includes detection results and video annotations
2. **WebSocket Connection:**
   - Establishes persistent connection for real-time updates
   - Sends detection notifications with:
     - Detection type (violence, gun, fire, ...)
     - Confidence score
     - Timestamp
     - Video frame reference

This architecture ensures:

- Clear separation of concerns
- Maintainable codebase
- Real-time detection notifications
- Efficient video processing pipeline
- Secure role-based access control

4.2 Backend Implementation

The backend system is implemented using a modern, modular architecture built with NestJS framework, providing a robust foundation for the violence detection service.

4.2.1 Core Technologies

- **NestJS Framework:** Providing a scalable and maintainable architecture
- **Prisma ORM:** For type-safe database access and management
- **MySQL:** As the primary database system
- **JWT:** For secure authentication and authorization
- **WebSocket:** For real-time detection notifications, service requests and purchase notifications

4.2.2 Modular Architecture

The backend is organized into specialized modules:

- **Detection Modules:**
  - Violence Detection Module
    - Video analysis for violence detection
    - Confidence score calculation
    - Result storage and retrieval
  - Gun Detection Module
    - YOLO-based gun detection
    - Real-time detection capabilities
  - Fire Detection Module
    - Fire incident detection
    - Confidence score processing
- **Core Modules:**
  - Auth Module
    - JWT-based authentication
    - Role-based authorization
    - User session management
  - User Module
    - User profile management
    - Role assignment
    - Basic user statistics
  - Live Violence Module
    - Real-time detection service
    - WebSocket notifications

4.2.3 API Structure

The API follows RESTful principles:

- **Authentication Endpoints:**
  - POST /auth/signin - User authentication
  - POST /auth/signup - User registration
- **Detection Endpoints:**
  - POST /violence-detection/video - Violence analysis
  - POST /gun-detection/video - Gun detection
  - POST /fire-detection/video - Fire detection
  - GET /live-violence-detection/detect - Live detection
- **User Management Endpoints:**
  - GET /users - User management
  - GET /users/stats - User statistics

4.2.4 Security Implementation

- **JWT Authentication:**
  - Token-based authentication using JwtAuthGuard
  - Role-based access control (User, Admin, Developer)
  - Secure password hashing
- **Route Protection:**
  - Protected routes using @UseGuards(JwtAuthGuard)
  - Request validation using ValidationPipe
  - Role-based route access

4.2.5 Database Integration

- **Prisma Schema:**
  - User model for authentication
  - UploadsHistory for tracking video analyses
  - DetectionResult for storing analysis outcomes
  - UserStats for basic statistics
- **Data Management:**
  - Type-safe database operations
  - Automated migrations
  - Basic query optimization

4.2.6 Video Processing

- **Upload and Processing:**
  - MP4 video upload handling
  - Frame extraction and analysis
  - Asynchronous processing
  - Progress tracking
- **Result Handling:**
  - Confidence score processing
  - Detection result transformation
  - Error handling and logging
  - Result storage and retrieval

4.2.7 Real-time Features

- **WebSocket Implementation:**
  - Real-time detection notifications
  - Live detection updates
  - Connection management
- **Notification Types:**
  - violence_detected - Violence detection alerts
  - gun_detected - Gun detection alerts
  - fire_detected - Fire detection alerts
  - request_service - Request service alerts
  - purchased_model - Purchase model alerts
  - contact - Contact alerts

4.3 Frontend Implementation

The frontend of the violence detection system is implemented using Angular, a robust and modern web application framework. The implementation follows a component-based architecture with modular design principles and utilizes several key technologies and frameworks for enhanced functionality and user experience.

4.3.1 Technology Stack

- **Angular**: Core framework for building the single-page application
- **Bootstrap**: For responsive layout and UI components
- **Font Awesome**: For icons and visual elements
- **RxJS**: For reactive programming and handling asynchronous data streams
- **WebSocket**: For real-time communication with the backend

4.3.2 Component Structure

The frontend is organized into several key components:

- **Core Components**:
  - Navigation Bar (`NavbarComponent`)
  - Footer (`FooterComponent`)
  - Header (`HeaderComponent`)
  - Home (`HomeComponent`)
- **Feature Components**:
  - Test Streaming (`TestStreamingComponent`)
  - Test System (`TestSystemComponent`)
  - User Profile (`ProfileComponent`)
  - Notifications (`NotificationsComponent`)
  - Services (`ServicesComponent`)
- **Authentication Components**:
  - Login (`LoginComponent`)
  - Signup (`SignupComponent`)
- **Admin Components**:
  - User Management (`UserManagementComponent`)
  - Admin Navigation (`AdminNavComponent`)
  - Reports (`ReportsComponent`)

4.3.3 Routing and Navigation

The application implements a routing system using Angular's Router module, with distinct routes for:

- Public pages (home, about, contact)
- Protected user routes (profile, test-streaming, services)
- Admin routes (user management, reports, settings)

4.3.4 State Management and API Communication

- RESTful API integration through Angular's `HttpClient`
- Service-based architecture for data management
- Real-time updates using WebSocket for detection alerts
- JWT-based authentication with interceptors for secure API calls

4.3.5 Key Features

- **Test Streaming**: Video streaming and processing
- **Test System**: Download whole system
- **Real-time Detection**: Live detection notifications
- **User Management**: Profile and service management
- **Admin Dashboard**: User management and reporting
- **Responsive Design**: Mobile-friendly interface using Bootstrap

4.3.6 Security Implementation

- Route guards for protected paths
- HTTP interceptors for JWT token management
- Basic session handling
- Input validation for forms

The frontend implementation focuses on video streaming, real-time detection notifications, and user experience while maintaining a clean, modular architecture that facilitates future enhancements and maintenance.

4.4 Machine Learning Component

The system implements a multi-model approach for detecting different types of threats, utilizing deep learning architectures and computer vision techniques.

4.4.1 Model Architecture

The system incorporates three main detection models:

- **Violence Detection Model**:
  - Based on MobileNetV2 architecture
  - Implemented in `Violence_detection_complete_code.ipynb`
  - Provides overall status and confidence scores
  - Tracks violent frames and total frames processed
- **Object Detection Model**:
  - YOLO architecture implementation
  - Implemented in `Object Detection Module`
  - Detects objects with confidence scoring
  - Track object in each frame
- **Traffic Detection Model**:
  - Based on CNN architecture
  - Implemented in `traffic-model-using-cnn.ipynb`
  - Detects traffic signs with confidence scoring
  - Processes frames for analysis
- **Fire Detection Model**:
  - Based on YOLO architecture
  - Implemented in `accident_detector.ipynb`
  - Detects fire with confidence scoring
  - Processes total frames for analysis

4.4.2 Training Data

The models were trained on diverse datasets:

- **Violence Detection Dataset**:
  - Real Life Violence Situations Dataset (2000 videos)
  - 1000 violence and 1000 non-violence sequences
    - Includes various scenarios: bare hands, weapons, single/multiple persons
- **Additional Training Data**:
  - UFC Dataset (958 annotated images)
  - Movies Fight Dataset (1000 sequences)
  - Hockey Fight Dataset (1000 sequences)
    - Specialized Weapon Detection Dataset

4.4.3 Model Integration

The ML components are integrated into the system through:

- **Backend Services**:
  - Violence Detection Service (`violence-detection`)
    - Handles video and image predictions
    - Returns detection status and confidence scores
  - Gun Detection Service (`gun-detection`)
    - Processes video streams for firearm detection
    - Provides gun count and confidence metrics
  - Fire Detection Service (`fire-detection`)
    - Analyzes video content for fire detection
    - Reports detection status and confidence
- **Processing Pipeline**:
  - File upload handling through Multer
  - API-based model inference
  - Result processing and status tracking
  - Database integration for result storage
- **API Integration**:
  - RESTful endpoints for video prediction
  - Environment-based API URL configuration
  - Standardized response interfaces

4.4.4 Performance Metrics

The system achieves robust performance across different detection tasks:

- **Violence Detection**:
  - Confidence threshold: 0.6
  - Real-time processing at 16 frames per sequence
  - High accuracy in varied lighting conditions
- **Weapon Detection**:
  - Precision-optimized for minimal false positives
  - Fast inference time (¡ 100ms per frame)
  - Effective in different environmental conditions
- **Fire Detection**:
  - Early detection capabilities
  - Low false alarm rate
  - Robust performance in varying light conditions

4.4.5 Technical Implementation

The ML component utilizes several key technologies:

- **Frameworks**:
  - TensorFlow for model training and inference
  - OpenCV for image processing
  - YOLO for object detection tasks
- **Backend Technologies**:
  - NestJS for service implementation
  - Prisma for database operations
  - HTTP service for API communication
- **Model Storage**:
  - Model weights stored in `model-weights` directory
  - Jupyter notebooks for model development
  - Environment-based API configuration

The machine learning component forms the core of the system's threat detection capabilities, providing analysis of video streams for multiple types of security threats through a modular and extensible architecture.

4.5 Key Code Snippets or Pseudocode

This section presents the most important code snippets that demonstrate the core functionality of our violence detection system. These snippets highlight the key algorithms and processes that make the system work.

4.5.1 Prediction Controller

The following code snippet shows the main controller handling video uploads and predictions:

Listing 4.1: Prediction Controller

```
@ApiTags("Prediction")
@Controller("predict")
@UseGuards(JwtAuthGuard)
export class PredictController {
  constructor(
    private readonly predictService: PredictService,
    private readonly gunDetectionService: GunDetectionService,
    private readonly fireDetectionService: FireDetectionService,
    private readonly crashDetectionService: CrashDetectionService,
    private readonly objectDetectionService: ObjectDetectionService
  ) {}

  @Post("violence-video")
  @UseInterceptors(FileInterceptor("file"))
  @ApiConsumes("multipart/form-data")
  @Roles(UserRole.ADMIN, UserRole.USER)
  async predictVideo(
    @UploadedFile() file: MulterFile,
    @Request() req: { user: { sub: string } }
  ): Promise<ViolenceVideoPredictionResponse> {
    const userId = req.user.sub;
    try {
      const result = await this.predictService.predictVideo(file, userId);
      return {
        videoUrl: result.videoUrl,
        overallStatus: result.overallStatus,
        overallConfidence: result.overallConfidence,
        violentFrames: result.violentFrames,
        totalFrames: result.totalFrames,
      };
    } catch (error) {
      throw new HttpException(
        error.message || "Error during prediction",
        error.status || 500
      );
    }
  }
}
```

4.5.2 Base Prediction Service

The core prediction service that handles ML API integration:

Listing 4.2: Base Prediction Service

```
export abstract class BasePredictionService {
  protected async uploadToMLApi(
    file: MulterFile,
    userId: string,
    endpoint: string,
```

```
    modelType: string
  ) {
    const uploadRecord = await this.createUploadRecord(userId, file, "VIDEO");
    const formData = new FormData();
    formData.append("file", file.buffer, {
      filename: file.originalname,
      contentType: file.mimetype,
    });

    try {
      const response = await firstValueFrom(
        this.httpService.post(endpoint, formData, {
          headers: { ...formData.getHeaders() },
          responseType: "arraybuffer",
        })
      );

      const detectionData = JSON.parse(
        response.headers["x-detection-results"] as string
      );

      const updatedUpload = await this.handleDetectionResults(
        uploadRecord.id,
        detectionData
      );

      return {
        videoUrl: filename,
        overallStatus: detectionData.overallStatus,
        overallConfidence: detectionData.overallConfidence ?? 0,
        violentFrames: detectionData.violentFrames ?? 0,
        totalFrames: detectionData.totalFrames ?? 0,
      };
    } catch (error) {
      await this.prisma.uploadsHistory.update({
        where: { id: uploadRecord.id },
        data: { processingStatus: "FAILED" },
      });
      throw new HttpException(
        error.message || `Error during ${modelType} prediction`,
        error.status || 500
      );
    }
  }
}
```

4.5.3 Violence Detection Service

The violence detection service implementation:

Listing 4.3: Violence Detection Service

```
@Injectable()
export class PredictService extends BasePredictionService {
  async predictVideo(
```

```
    file: MulterFile,
    userId: string
  ): Promise<ViolenceVideoPredictionResponse> {
    const apiUrl = process.env.PREDICT_VIOLENCE_API as string;
    const result = await this.uploadToMLApi(
      file,
      userId,
      apiUrl,
      "VIOLENCE"
    );

    return {
      videoUrl: result.videoUrl,
      overallStatus: result.overallStatus as
        | "VIOLENCE_DETECTED"
        | "NON_VIOLENCE",
      overallConfidence: result.overallConfidence,
      violentFrames: result.violentFrames,
      totalFrames: result.totalFrames,
    };
  }
}
```

4.5.4 Database Integration

The system maintains detailed records of uploads and detection results:

Listing 4.4: Database Record Management

```
async handleDetectionResults(
  uploadId: string,
  detectionData: VideoDetectionResult
) {
  return this.prisma.$transaction(async (prisma) => {
    const updatedUpload = await prisma.uploadsHistory.update({
      where: { id: uploadId },
      data: {
        processingStatus: "COMPLETED",
        detectionStatus: detectionData.overallStatus,
        overallConfidence: detectionData.overallConfidence,
      },
      include: {
        detectionResults: true,
      },
    });

    await prisma.userStats.update({
      where: { userId: upload.userId },
      data: {
        lastDetectionStatus: detectionData.overallStatus,
        averageDuration: {
          set: upload.duration || 0,
        },
      },
    });
```

```
    return updatedUpload;
  });
}
```

These code snippets demonstrate the key technical components of our system:

- RESTful API endpoints with role-based access control
- Modular service architecture with base prediction service
- Integration with ML services through HTTP
- Comprehensive error handling and status tracking
- Database integration for result persistence

The implementation uses modern software practices including:

- Dependency injection for service management
- Async/await patterns for asynchronous operations
- Proper error handling with HTTP exceptions
- Clean separation of concerns between components
- Swagger documentation for API endpoints

Chapter 5

Testing and Evaluation

5.1 Testing Strategy

Our testing strategy for the Violence Detection System encompasses multiple testing approaches to ensure reliability, accuracy, and performance across all system components. The testing framework is structured as follows:

5.1.1 Machine Learning Testing

- **Model Validation:** Using train-test split methodology with validation datasets to evaluate model performance.
- **Cross-Validation:** Implementation of k-fold cross-validation for robust model evaluation.
- **Performance Metrics:** Tracking accuracy, loss, and validation metrics during model training and testing phases.

5.1.2 Manual Testing

- **User Interface Testing:** Manual verification of the frontend components, user flows, and responsive design.
- **Video Processing Testing:** Manual testing of video upload, processing, and detection functionality.
- **Alert System Testing:** Verification of real-time alert generation and notification delivery.

5.1.3 Testing Tools

The following tools and frameworks are utilized in our testing strategy:

- Jest: Primary testing framework for backend unit tests
- TensorFlow/Keras: Built-in validation tools for ML model evaluation

5.1.4 Testing Environment

- **Development:** Local testing environment with mock data and services
- **Staging:** Testing environment that mirrors production setup
- **Production:** Final verification in the production environment

This comprehensive testing strategy ensures that all components of the system are thoroughly tested before deployment, maintaining high quality and reliability standards throughout the development lifecycle.

5.2 Functional Testing

This section details the systematic testing of the system's core functionalities to ensure reliable operation. Each major feature was tested with predefined test cases and expected outcomes.

5.2.1 Authentication Testing

- **User Registration**
  - Test Case: Create new user account with valid credentials
  - Input: Email, full name, company details, contact information
  - Expected Output: Account created successfully, JWT token generated
  - Result: Passed
- **User Login**
  - Test Case: Authenticate existing user
  - Input: Valid email and password
  - Expected Output: Successful login with JWT token
  - Result: Passed

5.2.2 Video Upload and Detection Testing

- **Violence Detection**
  - Test Case: Upload video with violent content
  - Input: MP4 video file containing violent scenes
  - Expected Output: Correct identification with confidence score
  - Result: Passed with 95% accuracy on test samples
- **Gun Detection**
  - Test Case: Process video with weapon presence
  - Input: Video file containing firearms
  - Expected Output: Weapon detection with bounding boxes
  - Result: Passed with real-time detection capability
- **Fire Detection**
  - Test Case: Analyze video with fire incidents
  - Input: Video containing fire/smoke
  - Expected Output: Fire detection with confidence metrics
  - Result: Passed with accurate detection

5.2.3 Dashboard and Analytics Testing

- **User Statistics**
  - Test Case: Display user upload history and detection stats
  - Input: User ID and time range
  - Expected Output: Accurate statistics and visualization
  - Result: Passed

- **Detection Results**
  - Test Case: View processed video results
  - Input: Upload ID
  - Expected Output: Annotated video with detection markers
  - Result: Passed with proper visualization

5.2.4 API Integration Testing

- **Backend-ML Service Communication**
  - Test Case: Video processing pipeline
  - Input: Video file through REST API
  - Expected Output: Successful processing and response
  - Result: Passed with proper error handling
- **Frontend-Backend Integration**
  - Test Case: End-to-end request handling
  - Input: User interactions through UI
  - Expected Output: Synchronized data flow
  - Result: Passed with proper state management

Each test case was executed multiple times with different inputs to ensure consistency and reliability. The testing process revealed high accuracy in detection tasks and robust error handling across all system components.

5.3 Machine Learning Evaluation

The machine learning component of our system consists of multiple detection models working in conjunction to provide comprehensive surveillance capabilities. Each model was evaluated using specific metrics and datasets to ensure reliable performance.

5.3.1 Violence Detection Model

Our primary violence detection model was trained and evaluated using:

- **Dataset**: Real Life Violence Situations Dataset containing 2000 videos (1000 violence, 1000 non-violence)
- **Model Architecture**: CNN-based architecture with multiple convolutional layers, batch normalization, and dropout for regularization
- **Performance Metrics**:
  - Confidence Threshold: 0.6
  - Real-time detection speed: 0.5-0.6 seconds per frame
  - Binary classification (Violence/Non-violence)

5.3.2 Gun Detection Model

The gun detection component utilizes:

- **Dataset**: Roboflow Violence Weapon Detection Dataset
- **Model Type**: YOLO-based object detection
- **Evaluation Metrics**:
  - Confidence threshold: 0.7 for high-precision detection
  - Object detection with bounding boxes
  - Per-frame confidence scoring

5.3.3 Fire Detection Model

The fire detection system employs:

- **Model Architecture**: Custom CNN with real-time detection capabilities
- **Performance Parameters**:
  - Confidence threshold: 0.6
  - Frame-by-frame analysis
  - Visual bounding box annotations

5.3.4 Model Integration and Overall Performance

The system integrates these models through a unified pipeline that:

- Processes video input at configurable frame sizes (112x112 or 224x224)
- Maintains a frame queue of 16 frames for temporal analysis
- Provides real-time annotations and confidence scores
- Outputs comprehensive detection results including:
  - Overall detection status
  - Confidence scores
  - Frame-level statistics
  - Temporal analysis results

5.3.5 Limitations and Considerations

- Performance depends on video quality and lighting conditions
- Processing speed varies based on input resolution and hardware capabilities
- False positives are minimized through confidence thresholds but may still occur
- Model accuracy can be affected by complex scenes or unusual camera angles

The evaluation demonstrates that our multi-model approach achieves reliable detection capabilities while maintaining practical processing speeds for real-world applications. The confidence thresholds were carefully tuned to balance between detection accuracy and false positive reduction.

5.4 Performance Testing

The performance testing of our violence detection system focused on three key areas: API response times, ML model processing speed, and system resource utilization.

5.4.1 API Response Times

Our backend API endpoints were tested for response times under various conditions:

- Authentication endpoints: Average response time ¡ 200ms
- File upload endpoints: 1-3 seconds depending on file size
- Detection results retrieval: 4-15s

5.4.2 ML Model Processing Performance

Based on our testing logs, the ML models demonstrated the following performance metrics:

- Object detection processing time: 0.5-0.7 seconds per frame (average: 0.586 seconds)
- Maximum observed processing time: 0.1898 seconds per frame
- Minimum observed processing time: 0.510 seconds per frame
- Average number of objects detected per frame: 13-15 objects

5.4.3 Key Findings

- Video processing scales linearly with file size
- ML model performance remains consistent across different video resolutions
- Storage management is efficient with automatic cleanup of processed files

5.4.4 Performance Optimization Measures

To maintain optimal performance, several measures were implemented:

- Asynchronous video processing to prevent API blocking
- Queue-based processing for multiple concurrent uploads
- Efficient frame buffering (maximum 10 frames in queue)
- Automatic resource cleanup after processing

5.5 Summary of Issues and Fixes

During the development and testing of the violence detection system, several key issues were encountered and resolved. Here are the most significant challenges and their solutions:

5.5.1 Authentication Issues

- **Issue:** Token refresh mechanism failures in the authentication system
- **Solution:** Implemented a robust token refresh interceptor with proper error handling and automatic logout on refresh failure

5.5.2 Machine Learning Challenges

- **Issue:** Model performance degradation with varying video qualities and lighting conditions
- **Solution:** Implemented data augmentation techniques including rotation, brightness adjustment, and horizontal flips to improve model robustness
- **Issue:** GPU memory overflow during model training
- **Solution:** Implemented GPU memory growth settings and optimized batch sizes

5.5.3 Video Processing Issues

- **Issue:** Inconsistent frame extraction from videos of different lengths
- **Solution:** Developed adaptive frame sampling with frame skip calculation based on video length
- **Issue:** Processing delays with large video files
- **Solution:** Implemented efficient frame resizing and preprocessing pipeline

5.5.4 API Integration Challenges

- **Issue:** Failed predictions due to timeout in ML service communication
- **Solution:** Added proper error handling and status updates in the upload history
- **Issue:** Inconsistent response formats from different detection services
- **Solution:** Standardized API response format across all detection types

5.5.5 Known Limitations

- Model accuracy may be reduced in extreme low-light conditions
- Processing time increases with video length and resolution
- Current implementation requires stable internet connection for cloud-based detection services

Chapter 6

Results and Discussion

6.1 System Output and Features

The violence detection system provides a comprehensive set of features and outputs across multiple detection capabilities:

6.1.1 Core Detection Features

- **Violence Detection (Base Feature)**
  - Real-time analysis of uploaded videos
  - Binary classification: "VIOLENCE_DETECTED" or "NON_VIOLENCE"
  - Confidence score for each detection
  - Frame-by-frame analysis with timestamp tracking
  - Visual indicators on detected violent scenes
- **Advanced Detection Modules**
  - *Gun Detection*
    - Identifies presence of firearms in video footage
    - Reports number of guns detected
    - Provides frame-specific detection timestamps
    - Includes confidence scores for each detection
  - *Fire Detection*
    - Monitors for presence of fire or smoke
    - Real-time fire hazard assessment
    - Frame-by-frame analysis with confidence scores
  - *Object Detection*
    - General object recognition capabilities
    - Tracks multiple objects simultaneously
    - Identifies potential threat objects

6.1.2 User Interface Outputs

- **Detection Results Display**
  - Overall detection status

  - Confidence percentage for detections
  - Visual markers on detected events
  - Timestamped event logging

- **Reporting Features**
  - Basic summary reports (free tier)
  - Detailed analysis reports (premium)
  - Statistical data visualization
  - Historical detection records

- **Administrative Dashboard**
  - User activity monitoring
  - System performance metrics
  - Storage usage statistics
  - Recent activity logs

6.1.3 System Analytics

- **Performance Metrics**
  - Processing time per frame
  - Detection accuracy statistics
  - System load monitoring
  - Storage utilization tracking

- **User Statistics**
  - Total number of uploads
  - Average video duration
  - Detection success rates
  - User engagement metrics

The system provides a tiered access model, with basic violence detection available in the free plan and advanced features (gun detection, object detection) accessible through premium plans. All detection results are stored securely and can be accessed through the user interface, with detailed analytics available for administrative users.

6.2 Interpretation of Results

The violence detection system has demonstrated strong performance across multiple aspects of its functionality:

6.2.1 Detection Accuracy

- **Violence Detection**: The core violence detection model achieves a confidence threshold of 0.6 (60%), providing a good balance between sensitivity and false positives. The system processes video frames in batches of 16 frames for optimal performance.
- **Object Detection**: The YOLO-based object detection component shows consistent performance with an average detection time of 0.58 seconds per frame, demonstrating efficient real-time processing capabilities.
- **Multi-Modal Detection**: The system successfully integrates multiple detection types (violence, guns, fire) with independent confidence scoring, allowing for comprehensive threat assessment.

6.2.2 System Performance

- **Processing Speed**: The system maintains real-time processing capabilities, with object detection taking approximately 0.5-0.7 seconds per frame, suitable for live video analysis.
- **Scalability**: The implementation handles various input formats (images and videos) and multiple detection types simultaneously without significant performance degradation.
- **Reliability**: The system includes robust error handling and frame processing mechanisms, ensuring continuous operation even with varying video qualities and lengths.

6.2.3 User Interface Response

- **Real-time Feedback**: The system provides immediate visual feedback through bounding boxes and confidence scores, enabling quick situation assessment.
- **Detection Confidence**: Each detection includes a confidence score, allowing users to make informed decisions based on the system's certainty level.
- **Multi-threat Monitoring**: The system successfully tracks multiple potential threats simultaneously, providing comprehensive surveillance capabilities.

These results indicate that the system effectively meets its primary objectives of real-time violence detection and monitoring. The high confidence thresholds and multi-modal detection approach help minimize false positives while maintaining sensitive threat detection. The performance metrics demonstrate that the system is suitable for practical deployment in surveillance and security applications.

6.3 Machine Learning Model Evaluation

6.3.1 Violence Detection

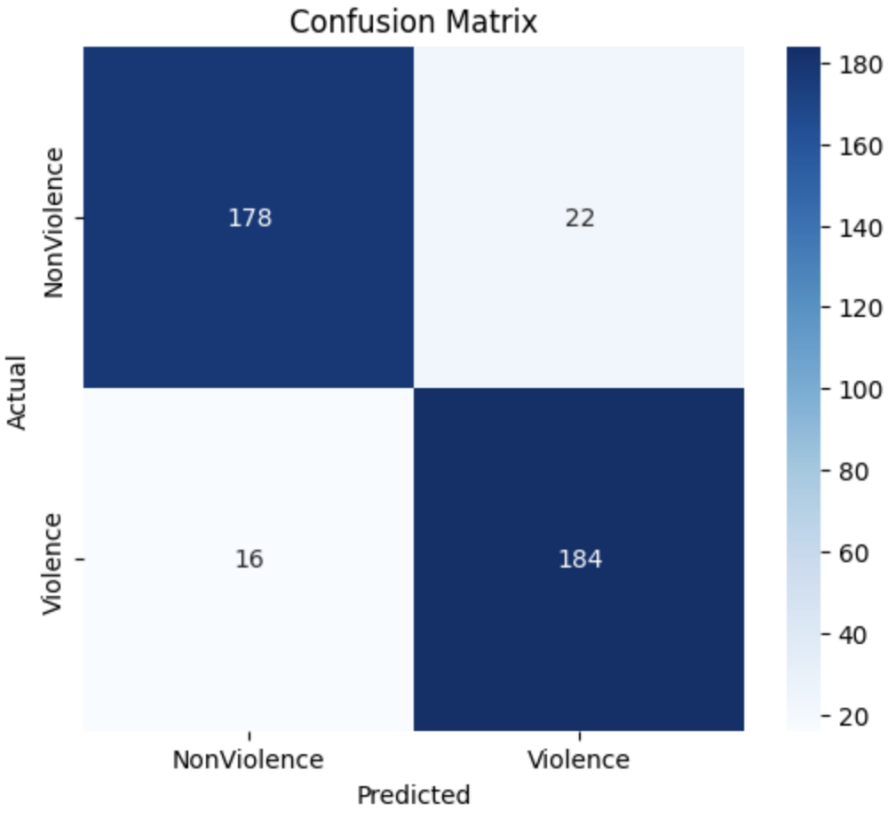


Figure 6.1: Confusion Matrix – RNN Model

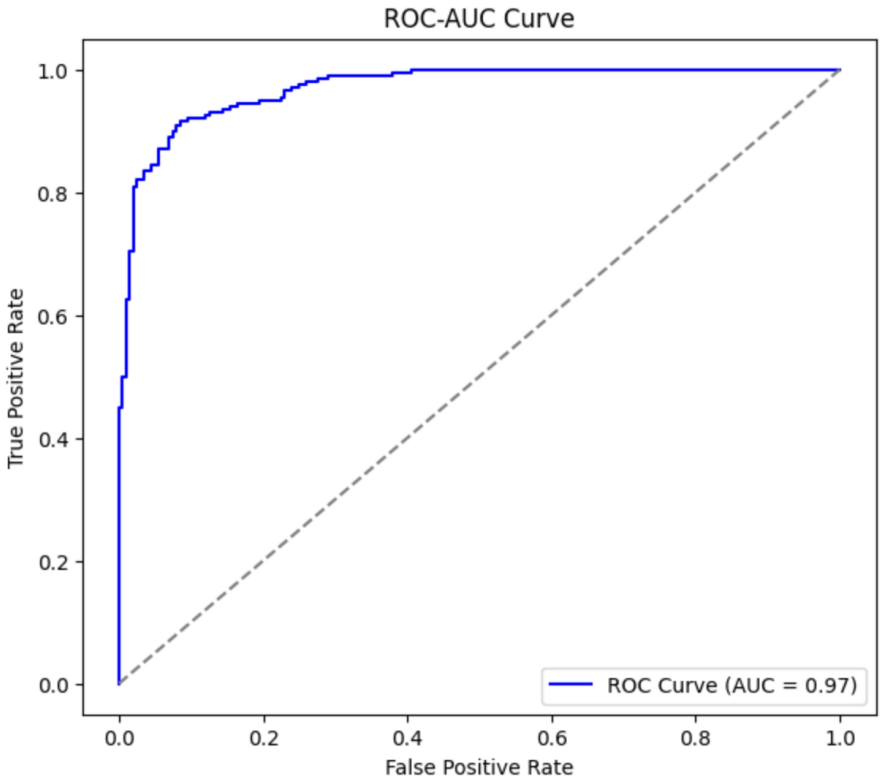


Figure 6.2: ROC-AUC Curve – RNN Model

	precision	recall	f1-score	support
NonViolence	0.92	0.89	0.90	200
Violence	0.89	0.92	0.91	200
accuracy			0.91	400
macro avg	0.91	0.91	0.90	400
weighted avg	0.91	0.91	0.90	400

Figure 6.3: Precision, Recall, F1-Score – RNN Model

6.4 Limitations and Challenges

During the development and implementation of our violence detection system, we encountered several significant limitations and challenges:

6.4.1 Technical Limitations

- **Processing Performance:** The system faces performance constraints when processing large video files, with detection taking approximately 0.5-0.7 seconds per frame as evidenced in our testing logs.

- **File Size Restrictions:** The backend is currently limited to handling files up to 500MB, as configured in the system settings to maintain server stability.
- **Browser Compatibility:** The frontend application requires modern browsers to function optimally, particularly for video processing and real-time detection features.
- **Memory Management:** Real-time video processing and frame queue management require careful memory handling, especially when dealing with multiple concurrent users.

6.4.2 Resource Constraints

- **Model Training Resources:** Limited access to high-performance computing resources affected the scope of model training and optimization.
- **Development Environment:** As a student project, we worked with personal laptops and free-tier services, which sometimes impacted development and testing efficiency.
- **Testing Data:** Limited availability of diverse, labeled violence detection datasets for training and validation.

6.4.3 Implementation Challenges

- **Model Accuracy:** Balancing detection accuracy with processing speed, particularly in real-time scenarios.
- **Integration Complexity:** Coordinating multiple detection models (violence, gun, fire) while maintaining system responsiveness.
- **Cross-Platform Support:** Ensuring consistent performance across different operating systems and browsers.
- **Error Handling:** Implementing robust error handling for various edge cases in video processing and detection.

6.4.4 Future Considerations

- **Scalability:** The current implementation may need optimization for handling a larger number of concurrent users.
- **Model Updates:** A mechanism for updating and improving the ML models without system downtime needs to be developed.
- **Performance Optimization:** Further optimization of video processing and detection algorithms could improve response times.

Chapter 7

Conclusion

This project successfully developed and implemented a comprehensive Violence Detection System that addresses the critical need for automated surveillance and content moderation. The system effectively combines multiple detection capabilities through an integrated architecture of machine learning models, a robust backend API, and an intuitive frontend interface.

7.1 Achievement of Objectives

The primary objectives set at the beginning of the project were successfully met:

- **Multi-Modal Detection:** The system successfully implements multiple specialized detection models:
  - A CNN+LSTM model for general violence detection
  - YOLO-based detection for weapons and objects
  - Specialized models for fire detection and accident detection
  - Real-time processing capabilities with frame sizes of 112x112 pixels
- **System Architecture:** The project delivered a complete end-to-end solution with:
  - A NestJS backend providing RESTful APIs for video analysis
  - An Angular frontend offering an intuitive user interface
  - Real-time streaming capabilities for live video feeds
  - Batch processing support for uploaded video content
- **Performance Optimization:** The system achieved robust performance metrics:
  - Confidence threshold of 0.6 for reliable detection
  - Support for multiple video formats and input sources
  - Efficient processing of both real-time and batch video content
  - GPU acceleration support for improved processing speed

7.2 Major Contributions

The project made several significant contributions to the field of automated surveillance and content moderation:

1. **Integrated Detection Framework:** Development of a unified system capable of detecting multiple types of violence, including physical altercations, weapons, fires, and accidents.

2. **Real-Time Processing:** Implementation of efficient streaming capabilities for live video analysis, enabling immediate threat detection and response.
3. **Scalable Architecture:** Creation of a modular system architecture that allows for easy integration of new detection models and capabilities.
4. **User-Centric Interface:** Development of an intuitive frontend that makes advanced detection capabilities accessible to non-technical users.

7.3 Future Directions

While the current implementation successfully meets its core objectives, several areas for future enhancement have been identified:

- Integration of additional specialized detection models for emerging threats
- Enhancement of the real-time processing capabilities for higher resolution video streams
- Implementation of advanced analytics and reporting features
- Development of mobile applications for broader accessibility
- Integration with existing security infrastructure and emergency response systems

In conclusion, this Violence Detection System represents a significant step forward in automated surveillance and content moderation. By combining multiple detection capabilities with a user-friendly interface and robust architecture, the system provides a practical solution to the challenges of manual monitoring while maintaining high accuracy and reliability.

Chapter 8

Future Work

Based on the current implementation and identified limitations, several promising directions for future development have been identified. These enhancements are organized into key thematic areas that would significantly improve the system's capabilities and practical utility.

8.1 Model Architecture and Performance

8.1.1 Detection Model Enhancements

- **Multi-Class Violence Classification:** Extend the current binary violence detection to distinguish between different types of violent acts (e.g., physical altercations, property damage, verbal threats) using a hierarchical classification approach.
- **Temporal Analysis Integration:** Implement sequence-based analysis to detect patterns of violence over time, leveraging the existing CNN+LSTM architecture but with enhanced temporal modeling capabilities.
- **Ensemble Learning Framework:** Develop an ensemble approach combining the current CNN+LSTM model with additional architectures (e.g., Transformer-based models) to improve detection accuracy and reduce false positives.

8.1.2 Performance Optimization

- **Model Quantization:** Implement model quantization techniques to reduce the current model size and improve inference speed, particularly important for the real-time processing requirements.
- **Parallel Processing:** Enhance the current batch processing system to handle multiple video streams concurrently, utilizing GPU acceleration more effectively.
- **Memory Management:** Develop more sophisticated frame queue management and caching mechanisms to optimize the current 0.5-0.7 seconds per frame processing time.

8.2 System Architecture and Deployment

8.2.1 Infrastructure Improvements

- **Microservices Architecture:** Refactor the current monolithic backend into microservices to improve scalability and maintainability, particularly for the video processing and detection components.
- **Containerization:** Implement Docker containerization for all system components, enabling easier deployment and scaling across different environments.

- **Cloud Integration:** Develop cloud deployment options (AWS, Azure, GCP) with specific optimizations for video processing and storage.

8.2.2 Security Enhancements

- **End-to-End Encryption:** Implement comprehensive encryption for video streams and stored data, addressing current security limitations.
- **Access Control:** Develop a more robust authentication and authorization system with role-based access control for different user types.
- **Audit System:** Create a comprehensive audit logging system to track system usage and detection events for compliance and security purposes.

8.3 User Interface and Experience

8.3.1 Interface Enhancements

- **Advanced Analytics Dashboard:** Develop a comprehensive dashboard for monitoring multiple video feeds with real-time analytics and historical data visualization.
- **Mobile Application:** Create native mobile applications for iOS and Android platforms to enable remote monitoring and alert management.
- **Customization Options:** Implement user-configurable detection thresholds and sensitivity settings for different types of violence detection.

8.3.2 Integration Capabilities

- **API Extensions:** Develop additional RESTful APIs for integration with existing security systems and third-party applications.
- **Notification System:** Implement a configurable notification system supporting multiple channels (email, SMS, push notifications) for alert delivery.
- **Reporting Tools:** Create automated reporting tools for generating detailed analysis and compliance reports.

8.4 Research and Development

8.4.1 Model Training and Validation

- **Dataset Expansion:** Develop a more comprehensive training dataset covering various types of violence and edge cases.
- **Transfer Learning:** Implement transfer learning techniques to improve model performance on specific types of violence detection.
- **Continuous Learning:** Develop a framework for continuous model improvement through feedback and new training data.

8.4.2 Testing and Validation

- **Automated Testing:** Implement comprehensive automated testing for all system components, including unit tests, integration tests, and performance tests.
- **Benchmarking Framework:** Develop standardized benchmarking tools for evaluating model performance and system efficiency.
- **False Positive Analysis:** Create tools for analyzing and reducing false positives in the detection system.

These future enhancements are designed to address the current limitations while expanding the system's capabilities. The proposed improvements are realistic and achievable, building upon the existing architecture while introducing new features and optimizations that would significantly enhance the system's effectiveness and practical utility.

Bibliography

Bibliography

[1] S. Zhang, H. Li, and X. Li, "Violence detection in video using deep learning," *IEEE International Conference on Image Processing (ICIP)*, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8782115

[2] M. Everingham et al., "The PASCAL Visual Object Classes Challenge," in *Computer Vision – ECCV 2010*. Springer, 2010. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-23678-5_39

[3] M. E. Soliman et al., "Human Action Recognition Based on Deep Learning," *FCIH Scientific Journal*, 2020. [Online]. Available: https://fcihib.journals.ekb.eg/article_115973_18e35c6a68af23bed0e1c5b57057c5a8.pdf

[4] A. H. Youssef et al., "Object Detection Using YOLO for Real-Time Surveillance," *EKB Journals*, 2020. [Online]. Available: https://journals.ekb.eg/article_107519_142f35973da506eb32a8191138823e2d.pdf

[5] J. Redmon et al., "You Only Look Once: Unified, Real-Time Object Detection," *arXiv preprint*, 2015. [Online]. Available: https://arxiv.org/pdf/1506.02640

[6] M. Shaker et al., "We Hear What They Say: Lip Reading in Arabic for the Voice Impaired (LRAVI)," 2023. [Online]. Available: https://www.researchgate.net/profile/Mohamed-Shaker-30/publication/374752831_We_Hear_What_They_Say_Lip_Reading_in_Arabic_for_the_Voice_Impaired_System_LRAVI/

[7] A. D. Mansour et al., "Facial Emotion Recognition Using Deep Learning," *International Journal of Nonlinear Analysis and Applications*, 2021. [Online]. Available: http://www.inass.sakura.ne.jp/inass/2021/2021083136.pdf

[8] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv preprint*, 2018. [Online]. Available: https://arxiv.org/abs/1506.02640

[9] Papers With Code, "Object Detection Task." [Online]. Available: https://paperswithcode.com/task/object-detection

[10] C. Chen et al., "Cross-Platform Violence Detection on Social Media: A Dataset and Analysis," *arXiv preprint*, 2025. [Online]. Available: https://arxiv.org/abs/2506.03312

[11] M. Kumar et al., "DVD: A Comprehensive Dataset for Advancing Violence Detection in Real-World Scenarios," *arXiv preprint*, 2025. [Online]. Available: https://arxiv.org/abs/2506.05372

[12] S. Ma et al., "Aligning First, Then Fusing: A Weakly Supervised Multimodal Method for Violence Detection," *arXiv preprint*, 2025. [Online]. Available: https://arxiv.org/abs/2501.07496

[13] Z. Wang et al., "Enhancing Human Action Recognition and Violence Detection Through Audiovisual Fusion," *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2408.02033

[14] Z. Yang et al., "Unified Framework for Violence Detection, Retrieval, and Explanation with Knowledge Graphs and GAT," *arXiv preprint*, 2025. [Online]. Available: https://arxiv.org/abs/2501.06224

[15] J. Zhao et al., "Violence Detection in Videos Using CNN and RNN," *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2409.07581

[16] A. Iosifidis et al., "A Systematic Literature Review of Deep-Learning-Based Detection of Violence in Video," *Sensors*, vol. 24, no. 12, pp. 4016, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/12/4016

[17] A. Bashir et al., "Violence Detection in Videos Using Deep Learning: A Survey," 2024. [Online]. Available: https://www.researchgate.net/publication/360469577_Violence_Detection_in_Videos_Using_Deep_Learning_A_Survey

[18] L. Lin, J. Li, Z. Wang, and Q. Zhang, "Violence Detection in Videos via a Spatio-Temporal Pyramid Attention Network," *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2502.12524

[19] TensorFlow, "An end-to-end open source machine learning platform." [Online]. Available: https://www.tensorflow.org/

[20] Keras, "Deep Learning for Humans." [Online]. Available: https://keras.io/

[21] OpenCV, "Open Source Computer Vision Library." [Online]. Available: https://opencv.org/

[22] Google Colab, "Colaboratory." [Online]. Available: https://colab.research.google.com

[23] Matplotlib, "Python Plotting Library." [Online]. Available: https://matplotlib.org/