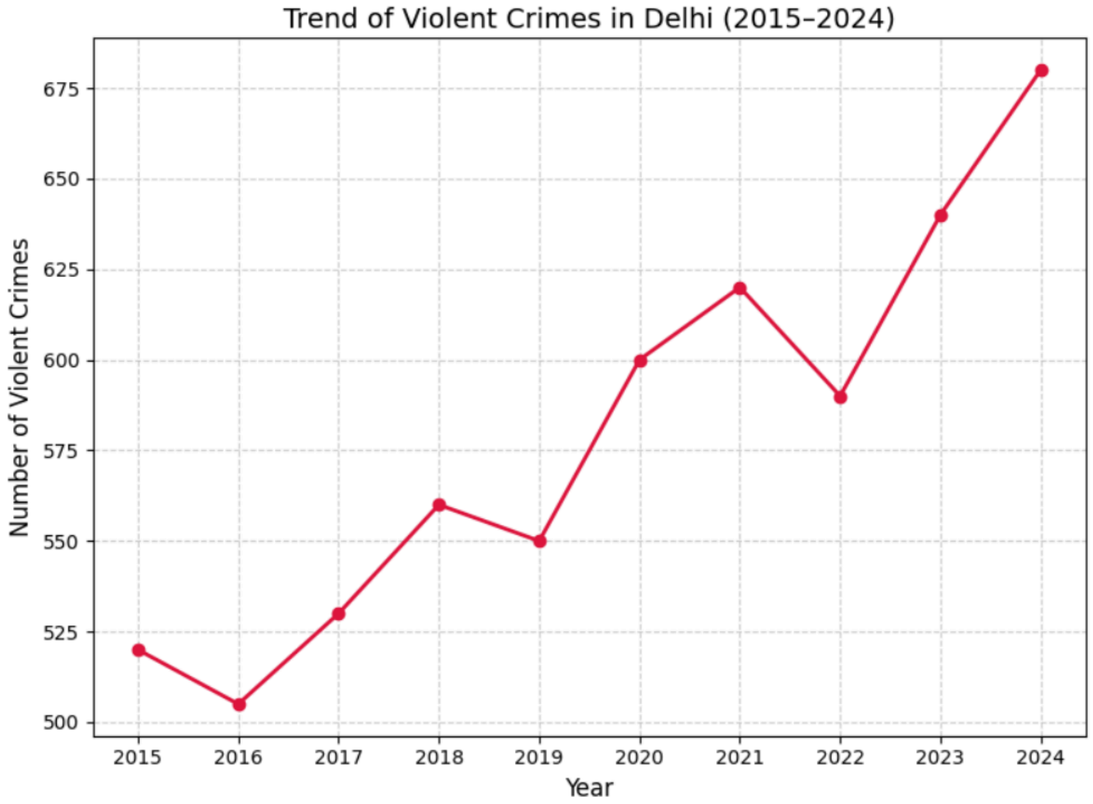
VIOLELNCE DETECTION USING RTSP STREAM

CONTENTS

MOTIVATION BEHIND THE PROJECT

Rising Threats

- Sharp rise in assaults, vandalism, and mob violence in public places(e.g.,metro stations,markets)
- Frequent cases of domestic violence, bullying, and altercations going undetected until it's too late



Limitations of Traditional Systems

- Manual monitoring is slow, inconsistent, and prone to human error
- Reactive systems delay intervention, increasing the risk of harm

Why Real-Time Detection?

- AI-driven surveillance enables instant alerts and faster response
- Helps prevent escalation

VISION

Safer Public Environments

- Real-time surveillance and analytics for proactive incident detection
- Strengthened communication networks for faster response

Scalable Smart City and Transport Hub Solutions

- IoT and AI-based systems to manage growing urban and transit demands
- Flexible infrastructure adaptable to evolving city needs

Integration with Law Enforcement

- Shared data platforms for real-time situational awareness
- Interoperable tools to enable rapid, coordinated response

GOAL OF THE PROJECT

Leverage Live Video Feeds

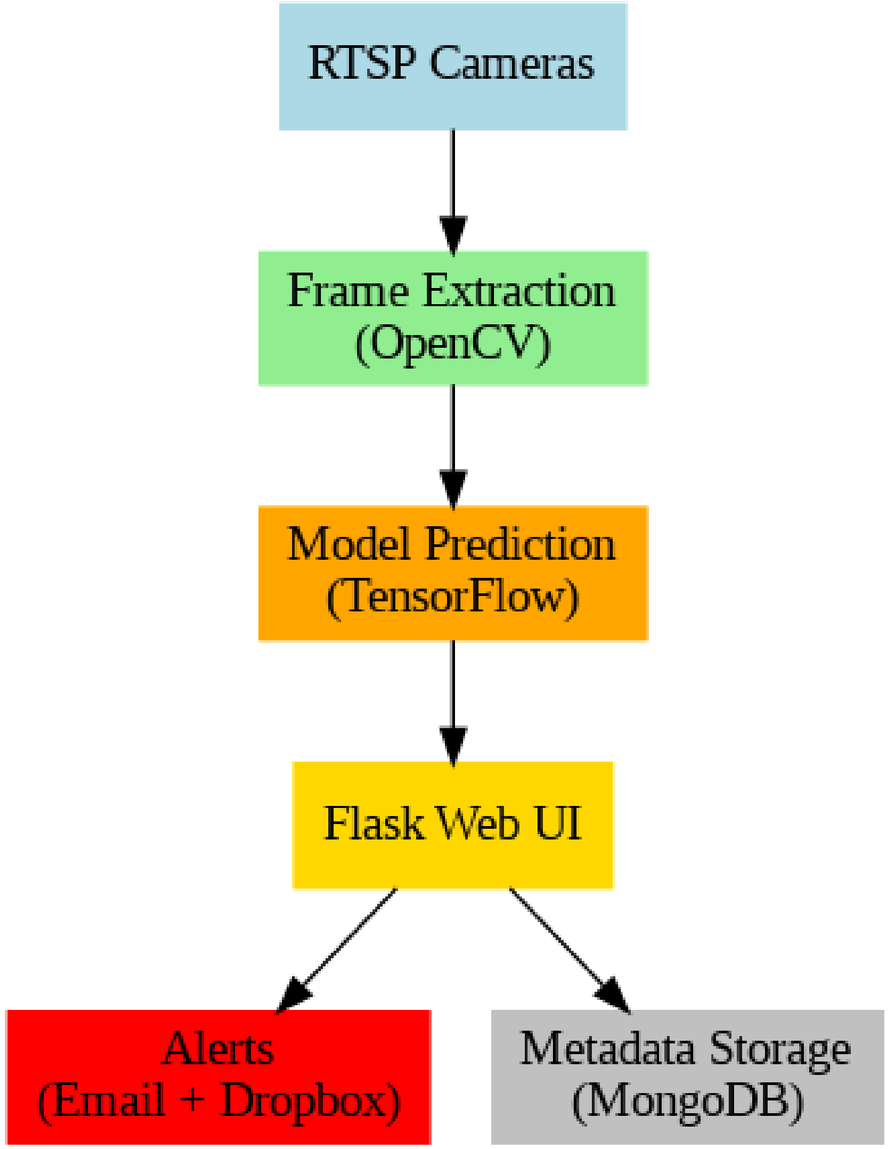
- Monitor real-time video streams from IP surveillance cameras.
- Continuously scan environments for signs of violent behaviour.

Apply Deep Learning for Detection

- Use a trained CNN model to analyse each frame.
- Accurately classify violent vs non-violent activity.

Automate Alerts and Evidence Storage

- Instantly trigger alerts upon violence detection.
- Save video frames and metadata securely in cloud storage and a database.



TECHNOLOGY STACK OVERVIEW

Python

- Core language for backend logic and integration.
- Supports ML, API handling, and video stream processing.

Flask

- Lightweight web framework.
- Handles live video feeds and routes like /video_feed, /detected_frames.

TensorFlow / Keras

- Used for CNN-based violence classification.
- Implements DenseNet121, fine-tuned for frame-level detection.

OpenCV

- Real-time frame processing (resize, RGB conversion, extraction).
- Works with RTSP streams.

RTSP (Real-Time Streaming Protocol)

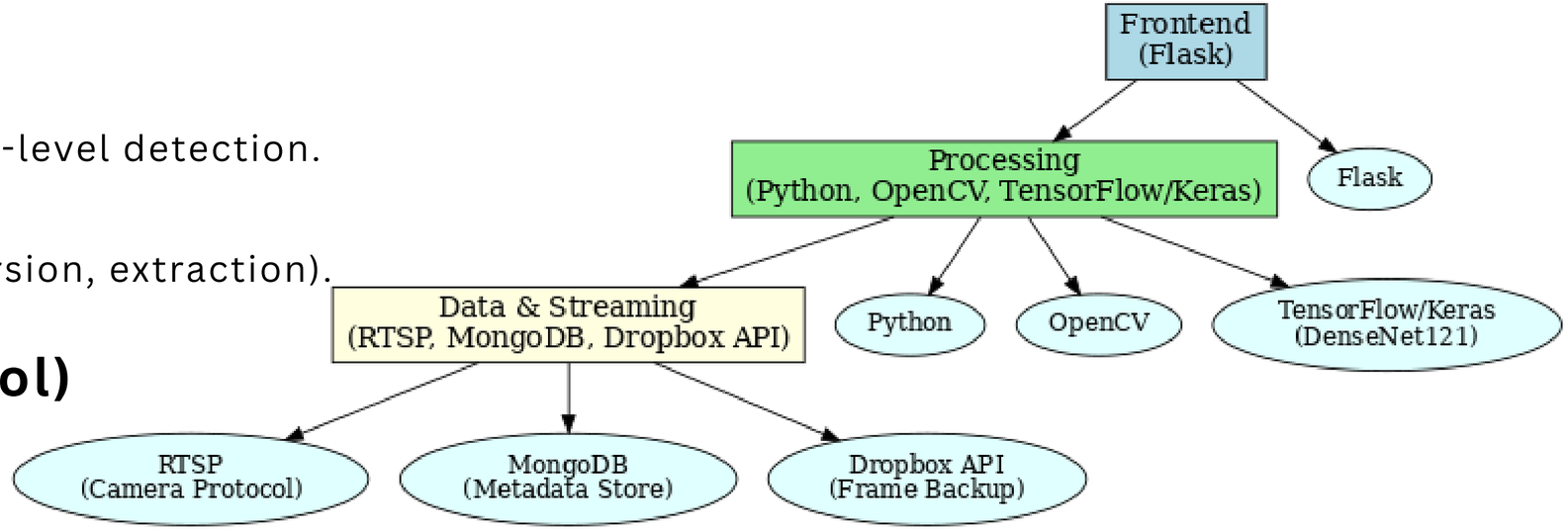
- Streams live video from IP cameras.
- Ensures low-latency feed to the backend.

MongoDB

- Stores metadata (timestamps, camera IDs, probability scores).
- Supports logging and web/API access.

Dropbox API

- Backs up detected frames to the cloud.
- Provides secure, redundant storage for review/audit.



SYSTEM WORKFLOW

1. Camera Stream Input (RTSP)

- The system reads RTSP video feed URLs from a config file, allowing dynamic stream management.
- Each stream is assigned a unique ID and runs in its own thread to ensure non-blocking, parallel processing.
- Streams are stored in a global dictionary with the latest frame, lock object for thread safety, and activation status.

2. Frame Capture (OpenCV)

- OpenCV’s VideoCapture fetches frames from each RTSP stream in dedicated threads.
- Frames are stored in shared memory for real-time access, protected by locks to avoid race conditions.

3. Pre-processing and Prediction

- Frames are pre-processed (e.g., resized, normalized) before being passed to a custom-trained model for object or anomaly detection.
- Frames flagged as containing significant events are queued for storage and alert generation.

4. Save to Dropbox

- Flagged frames are saved locally as .jpg files with timestamped filenames.
- Images are uploaded to Dropbox for off-site, scalable storage using a secure API token.

5. Metadata Logged to MongoDB

- Metadata for each uploaded frame is recorded in MongoDB, including timestamp, camera ID, filename, and Dropbox URL.
- This enables powerful querying, filtering by camera, and efficient retrieval of historical events.

6. Interface Updates in Real-Time via Flask

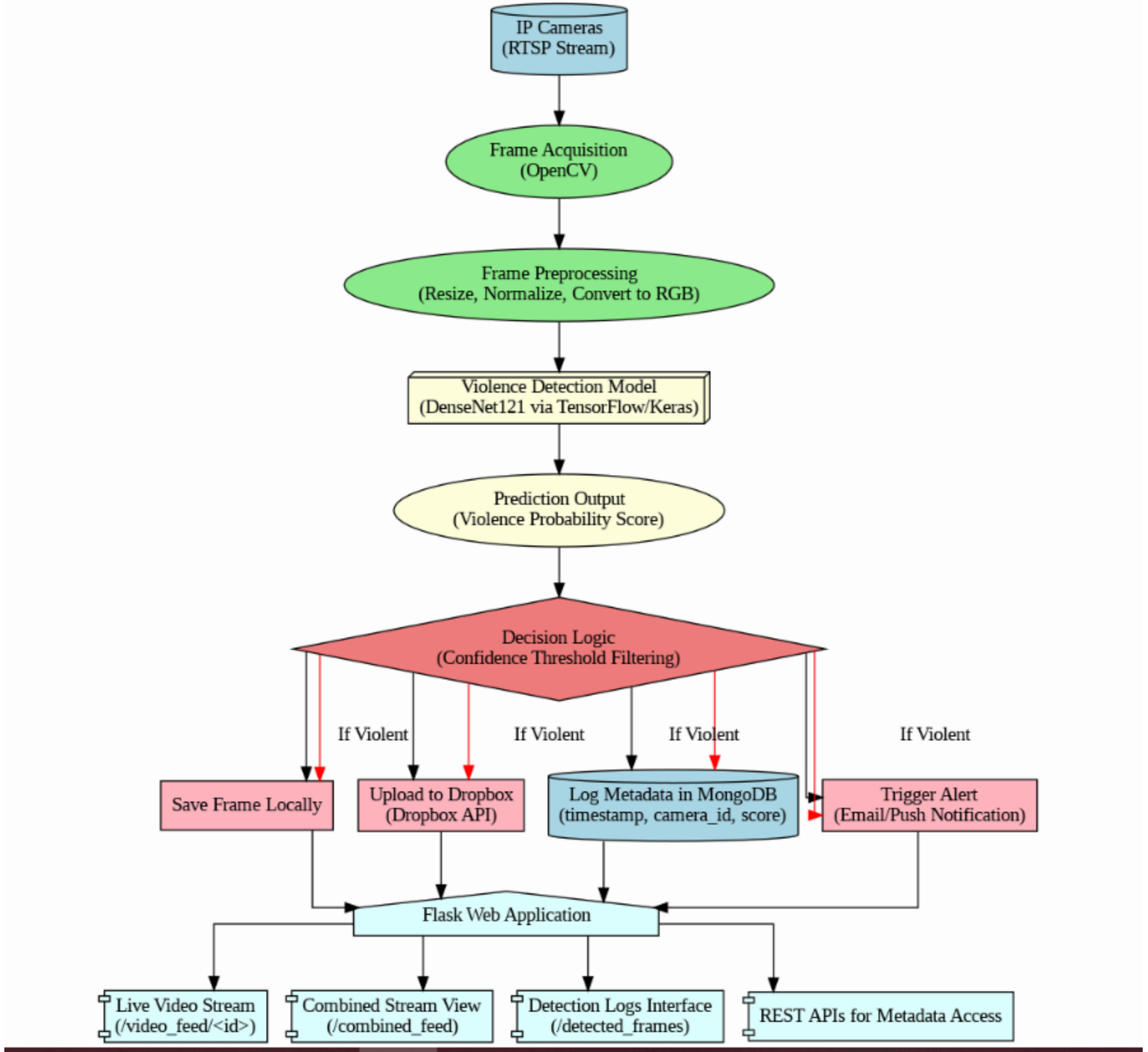
The Flask web dashboard provides real-time viewing with:

- /video_feed/<id>: Live stream from an individual RTSP camera.
- /combined_feed: Dynamic grid of active camera feeds.
- /detected_frames: Gallery of previously detected event frames.

7. Multithreading Architecture

- The backend is designed using a multithreaded architecture, where each camera stream is managed in a dedicated background thread.
- This ensures parallel processing, preventing any single slow or disconnected stream from impacting the performance of others.
- Shared resources like frames and status flags are protected using thread locks, maintaining data consistency and preventing race conditions.
- In scenarios where a stream temporarily fails to deliver frames, the system substitutes with blank placeholder frames, maintaining the visual structure of the combined feed and avoiding interface crashes.

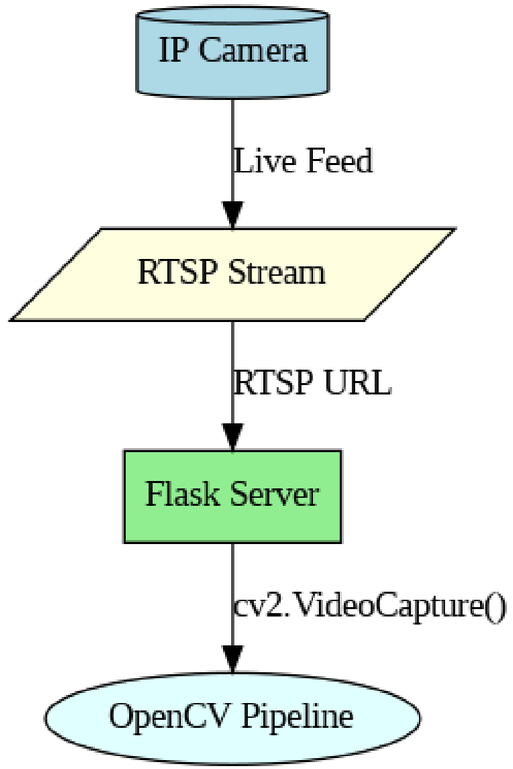
Real-Time Violence Detection System Architecture



RTSP INTEGERATION

RTSP (Real-Time Streaming Protocol)

- Used to fetch live feeds from IP surveillance cameras.
- Provides real-time video streams over network efficiently.



Frame Capture with OpenCV

- cv2.VideoCapture(rtsp_url) is used to open the RTSP stream.
- Frames are read in a loop using cap.read().

Multi-threaded Handling

- Each camera stream is handled in a separate thread or process.
- Ensures smooth and concurrent frame capture from multiple sources.

Integration with Flask

- Captured frames are streamed via endpoints like /video_feed/<camera_id>.
- Real-time feed is rendered using Flask's response generator with MJPEG.

DenseNet121 CNN MODEL

Model Architecture

- Utilizes DenseNet121 as the core feature extractor, pretrained on ImageNet and fine-tuned on a curated dataset containing violent and non-violent video frames.

Input

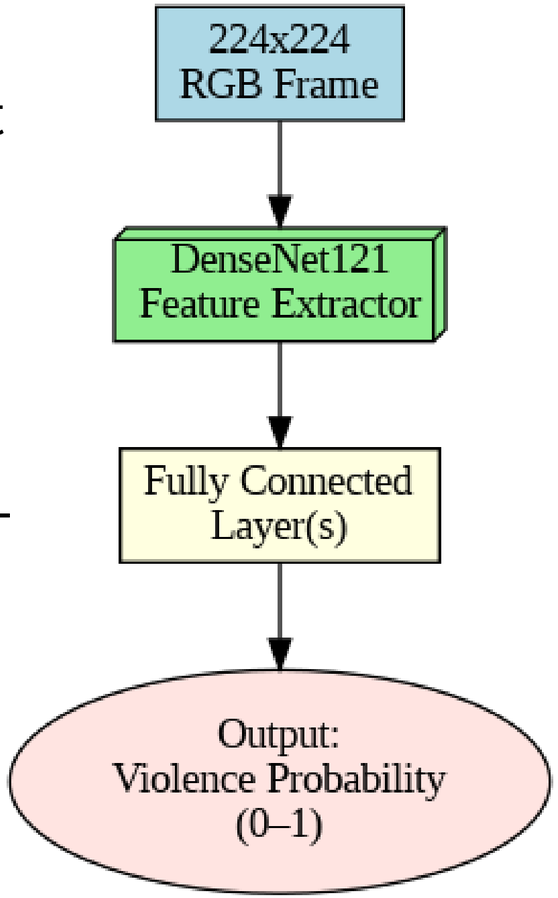
- Takes 224x224 RGB frames extracted from live RTSP video streams and pre-processed using OpenCV.

Output

- Generates a probability score (ranging from 0 to 1) that indicates the likelihood of violence in the frame.

Performance

- Optimized for real-time inference, ensuring rapid prediction and minimal latency—ideal for continuous video monitoring systems.



Storage & Backup

MongoDB – Metadata Logging

- Each detected frame is logged with:
  - timestamp
  - camera_id
  - filename
  - dropbox_url

```
# Save metadata to MongoDB
metadata = {
    "stream_id": stream_id,
    "frame_name": frame_name,
    "file_url": file_url,
    "upload_time": datetime.now()
}
```

- Enables filtering by date, time, and camera ID.
- Retrieval Support:
  - Metadata entries act as precise indexes to locate and retrieve previously captured frames from Dropbox.
  - This allows efficient access to historical footage without scanning the entire archive.

Dropbox – Cloud Backup

- Detected frames are saved as .jpg images with timestamped filenames.
- Uploaded securely using the Dropbox API with token-based authentication.
- Ensures safe, scalable, and accessible off-site backup for all detected events.

```
def upload_to_dropbox(frame_path, frame_name):
    dropbox_path = f'/frames/{frame_name}'
    with open(frame_path, 'rb') as file:
        dbx.files_upload(file.read(), dropbox_path, mode=dropbox.files.WriteMode('overwrite'))
    return f"File uploaded to Dropbox: {dropbox_path}"
```

DROPBOX DASHBOARD

Folders

All files

Apps

AVI F2

frames

Home

Folders

Activity

More

AV

Explore free features
1.23 MB of 2 GB

Search

Invite members

Start free trial

All files / Apps

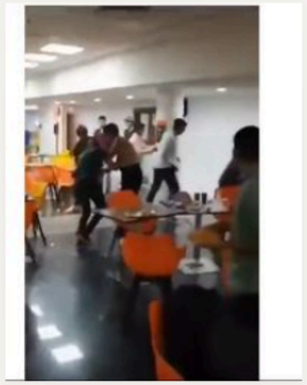
AVI F2

AV Only you

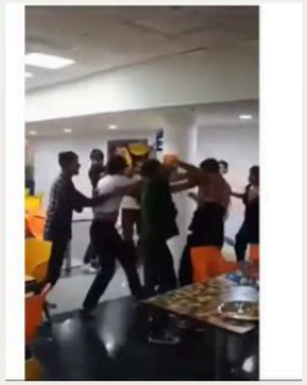
Share selected | Download | Delete | Open in | Send for review | ...

1 selected

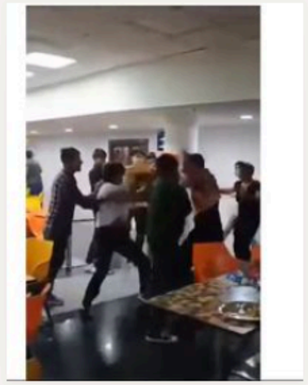
Name



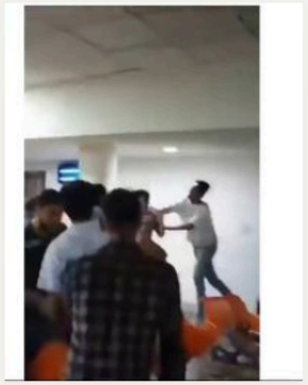
frame_883
JPG • 46.63 KB



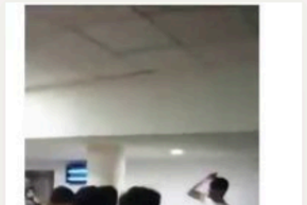
frame_785
JPG • 49.77 KB

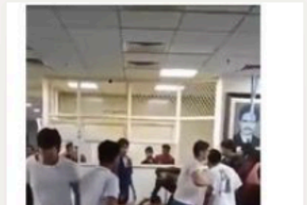


frame_781
JPG • 49.17 KB



frame_572
JPG • 39.07 KB





MONGO DB DASHBOARD

Type a query: { field: 'value' } or Generate query

Explain | Reset | Find | Options

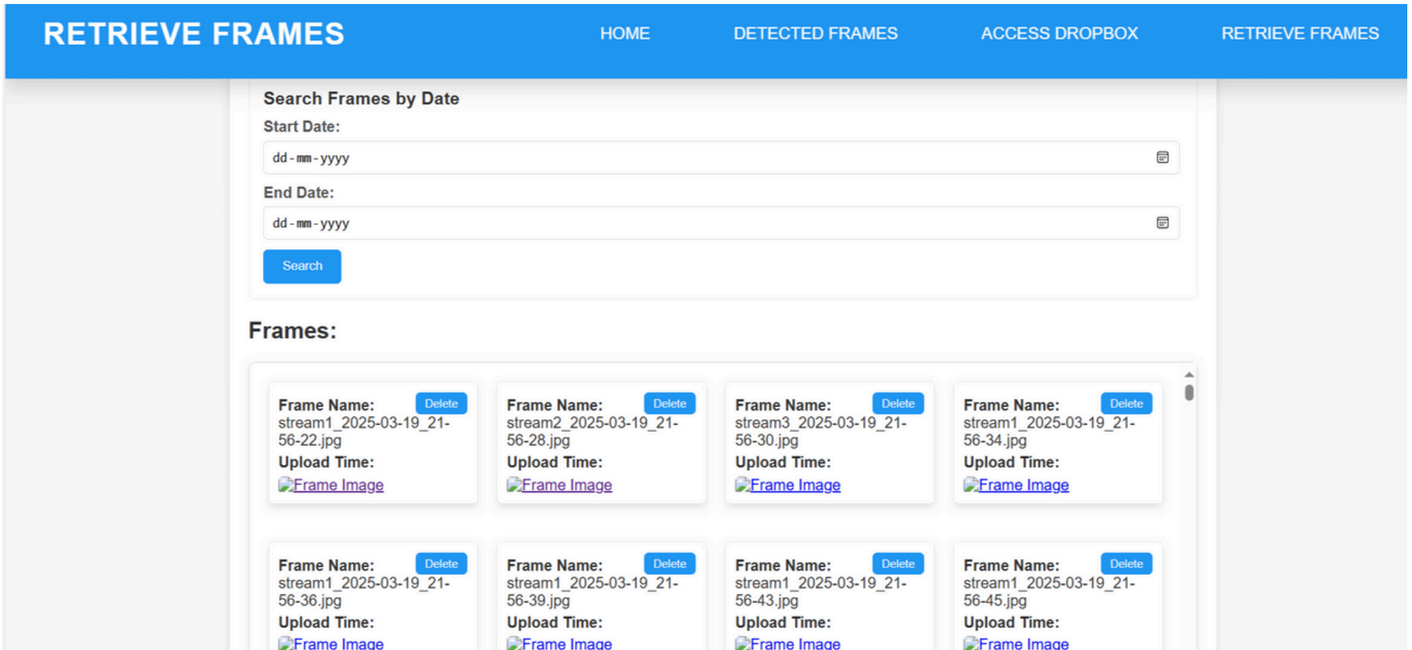
ADD DATA | EXPORT DATA | UPDATE | DELETE

100 | 1 – 100 of 755


```
_id: ObjectId('67daf0341dc29f8aeb546661')
stream_id : 1
frame_name : "stream1_2025-03-19_21-56-22.jpg"
file_url : "https://www.dropbox.com/scl/fi/qklk5hp6g9ctwwrxvhklh/stream1_2025-03-1…"
upload_time : 2025-03-19T21:56:28.053+00:00

_id: ObjectId('67daf0361dc29f8aeb546662')
stream_id : 2
frame_name : "stream2_2025-03-19_21-56-28.jpg"
file_url : "https://www.dropbox.com/scl/fi/4mf0yq1y5aujajnkuzp91/stream1_2025-03-1…"
upload_time : 2025-03-19T21:56:30.701+00:00
```


RETRIEVAL OF FRAMES USING MONGO DB



FLASK APPLICATION

Live Video Streaming

- /video_feed/<stream_id>: Serves live feed from a specific RTSP stream.
- /combined_feed: Dynamically merges all active streams into a single grid view.

Stream Management

- /start_streams: Initializes all RTSP streams from a text file.
- /stop_streams: Gracefully stops and clears all running streams.
- Streams managed using threads and locks to ensure thread-safe access to frames.

Capture ,Generating Frames & Grid Display

- Latest frame from each stream is collected and combined into a grid layout using OpenCV.
- Placeholder frames used if a stream has no current data.

Stream Initialization Logic

- RTSP URLs loaded from file using load_rtsp_links().
- initialize_streams() and capture_frames() functions used to manage concurrent video streams.

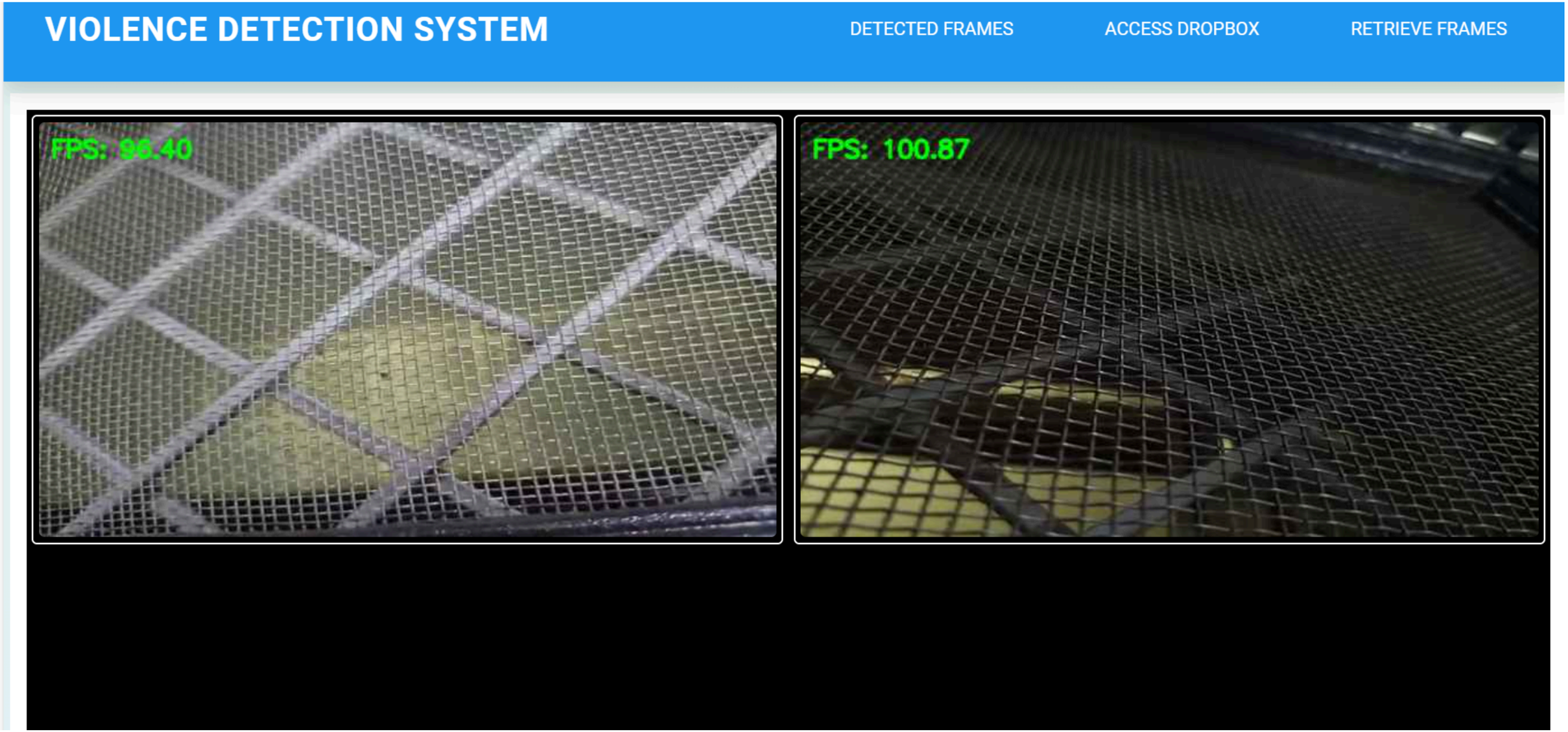
Streaming Format

- Frames encoded as JPEGs and streamed using multipart/x-mixed-replace content type for real-time display in browser.

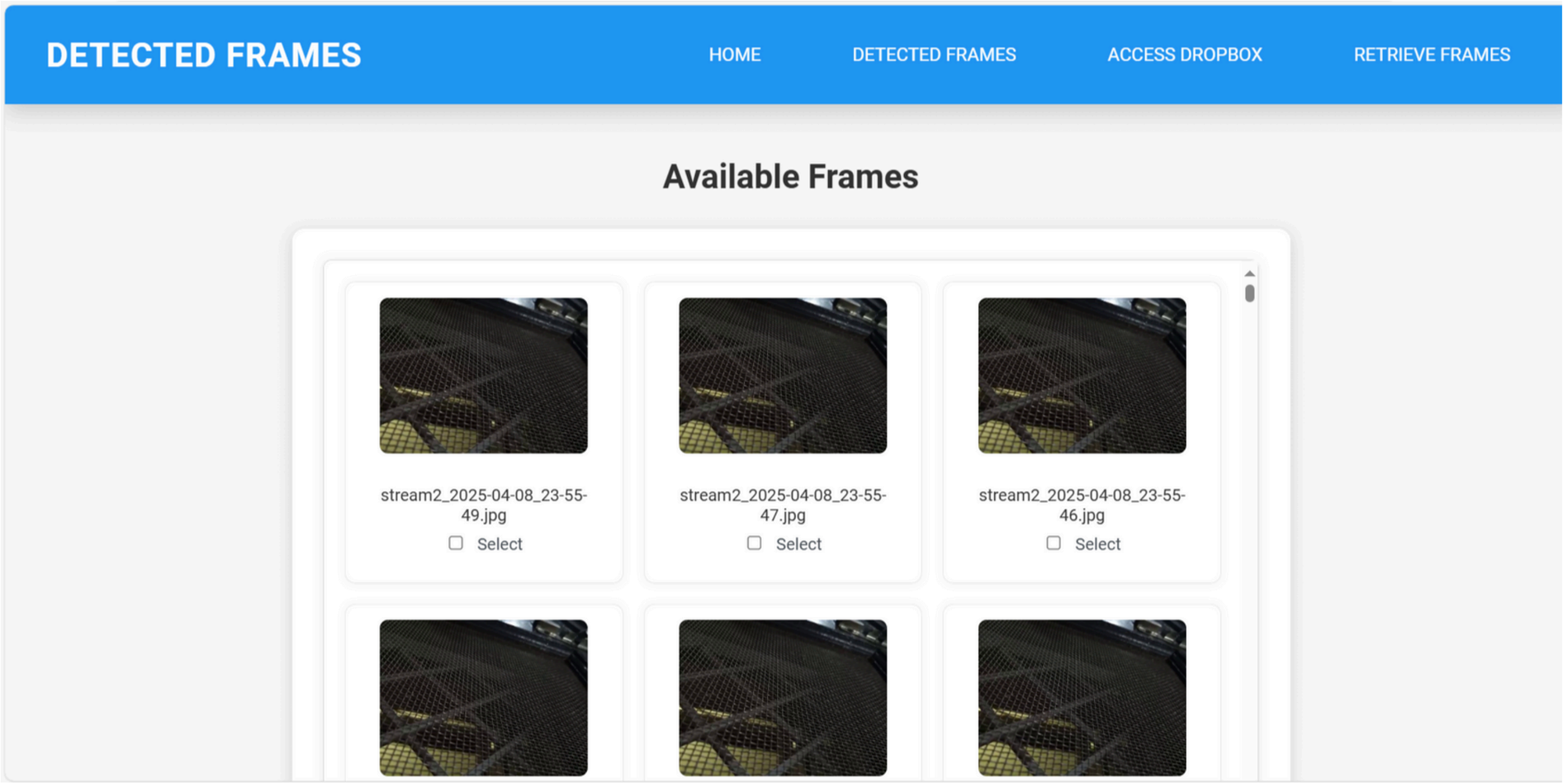
CORE FUNCTIONS

Function Name	Purpose	Tech Used
`detect_violence(frame)`	Predicts violence using a trained deep learning model	TensorFlow, NumPy
`capture_frames(rtsp_url, id)`	Captures frames from RTSP streams in dedicated threads	OpenCV, Threading
`generate_frames(id)`	Streams frames as HTTP multipart responses for live view	Flask, OpenCV
`save_and_upload_frame()`	Saves frame locally, uploads to Dropbox, logs metadata	OpenCV, Dropbox API, MongoDB

FLASK DASHBOARD



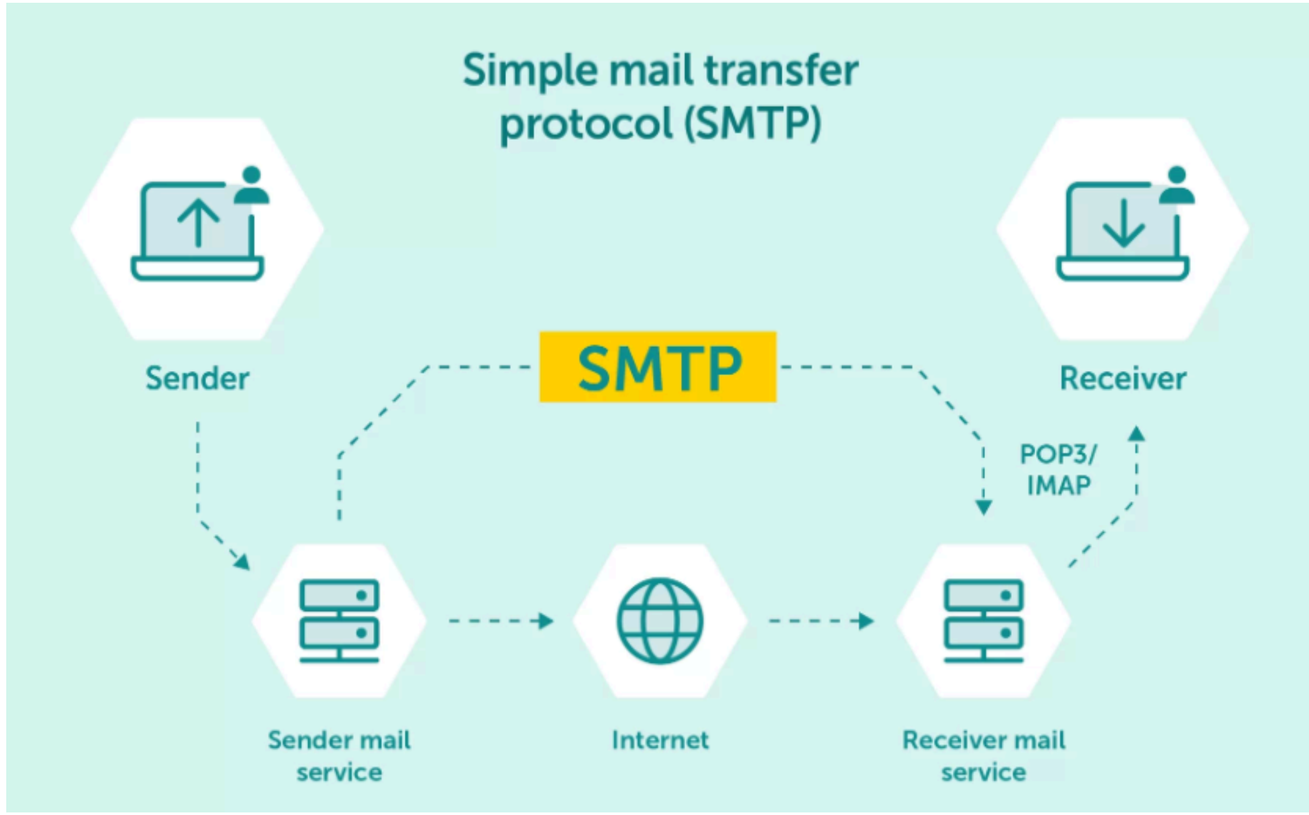
GRID DISPLAY OF FRAMES



EMAIL ALERT SYSTEM – FLASK & SMTP INTEGRATION

Key Features:

- Accessible via Flask route: @app.route('/send_email', methods=['POST'])
- Allows submission of:
  - Recipient email address
  - Custom subject line and message body
  - Selection of one or more image frames to attach



How It Works:

- Retrieves form data (recipient, subject , frames etc.)
- Validates that at least one frame is selected
- Creates an email using Message() from Flask-Mail
- Attaches each selected .jpg frame from local storage
- Sends the email via configured SMTP server

Error Handling:

- Returns HTTP 400 if no frames are selected
- Returns HTTP 500 if email fails to send due to SMTP or system errors

Benefits:

- Quick sharing of visual alerts
- Manual escalation or reporting tool

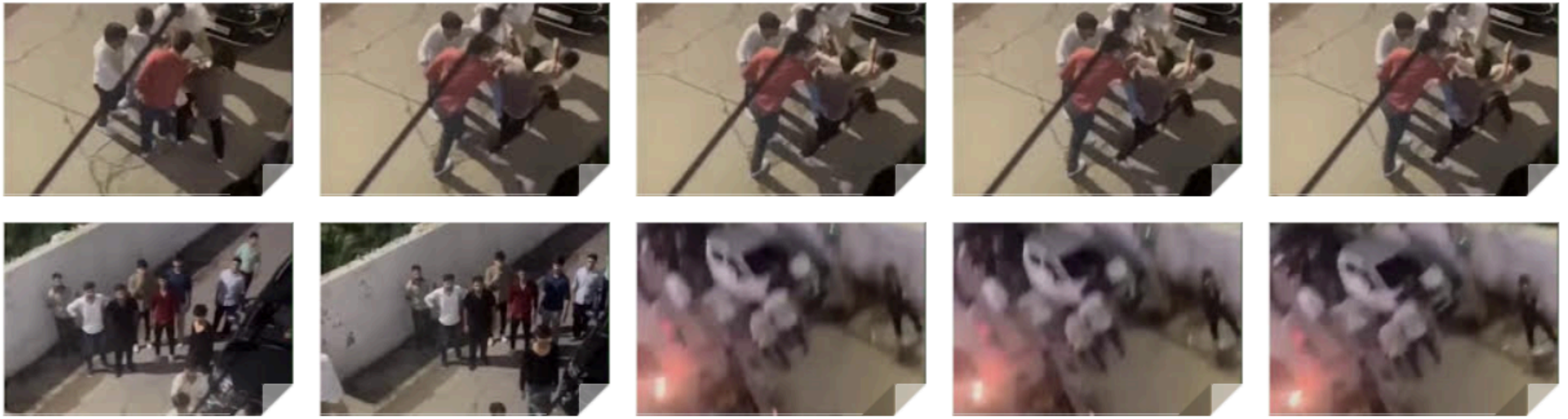
EMAIL ALERT DASHBOARD

vavi3984@gmail.com
to me

Tue, Apr 8, 10:46 PM (2 hours ago)

ALERT: Violent actions were detected in the video. Please find the violent frames below.

24 Attachments • Scanned by Gmail



DEPLOYMENT

Deployment Highlights:

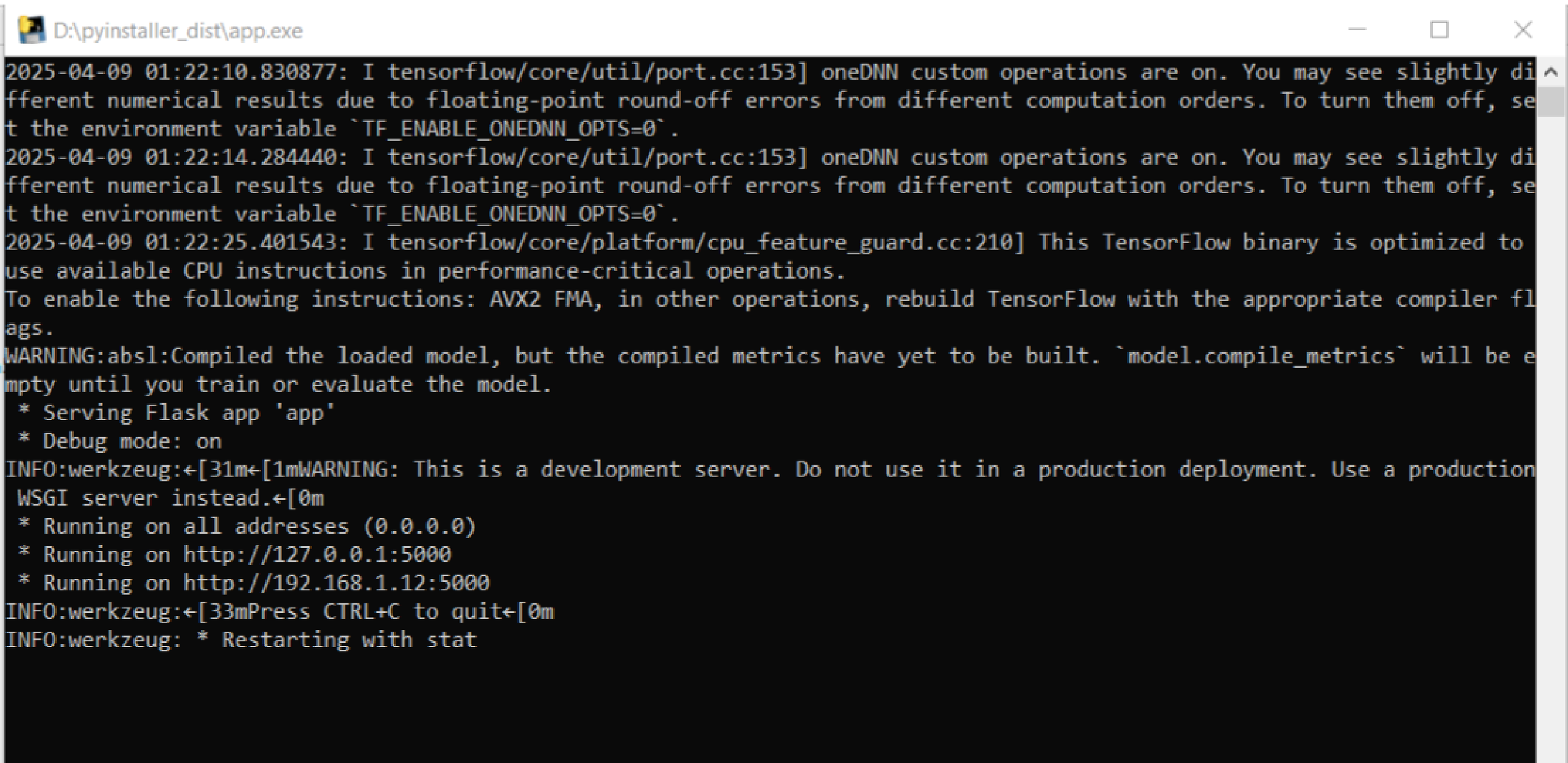
- The Flask-based application was packaged into a standalone executable using PyInstaller, allowing seamless deployment across systems.
- Key assets bundled:
  - Trained model (combined_detection_model.h5)
  - Templates and instance folders
  - Required .pyd and .dll binaries for TensorFlow, Keras, PIL, SQLite, and other dependencies
- Output distributed via:
  - --distpath for the final executable output
  - --workpath to manage build files separately

```
pyinstaller --onefile --add-data ... --add-binary ... --hidden-import ... app.py
```

Portable Desktop Software:

- Resulting .exe can run independently on any Windows system—with or without a Python environment.
- Minimal dependencies required on the target system (basic system libraries only).
- Ensures sharability, portability, and ease of installation for end users.

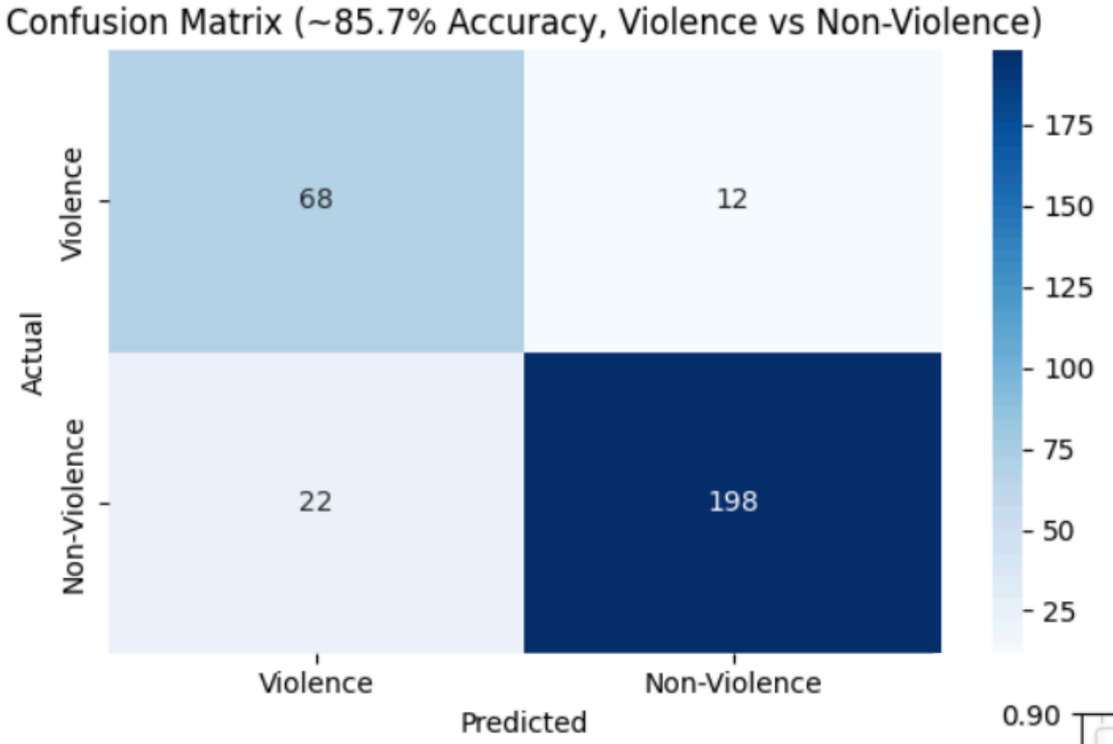
APP.EXE WORKING WINDOW



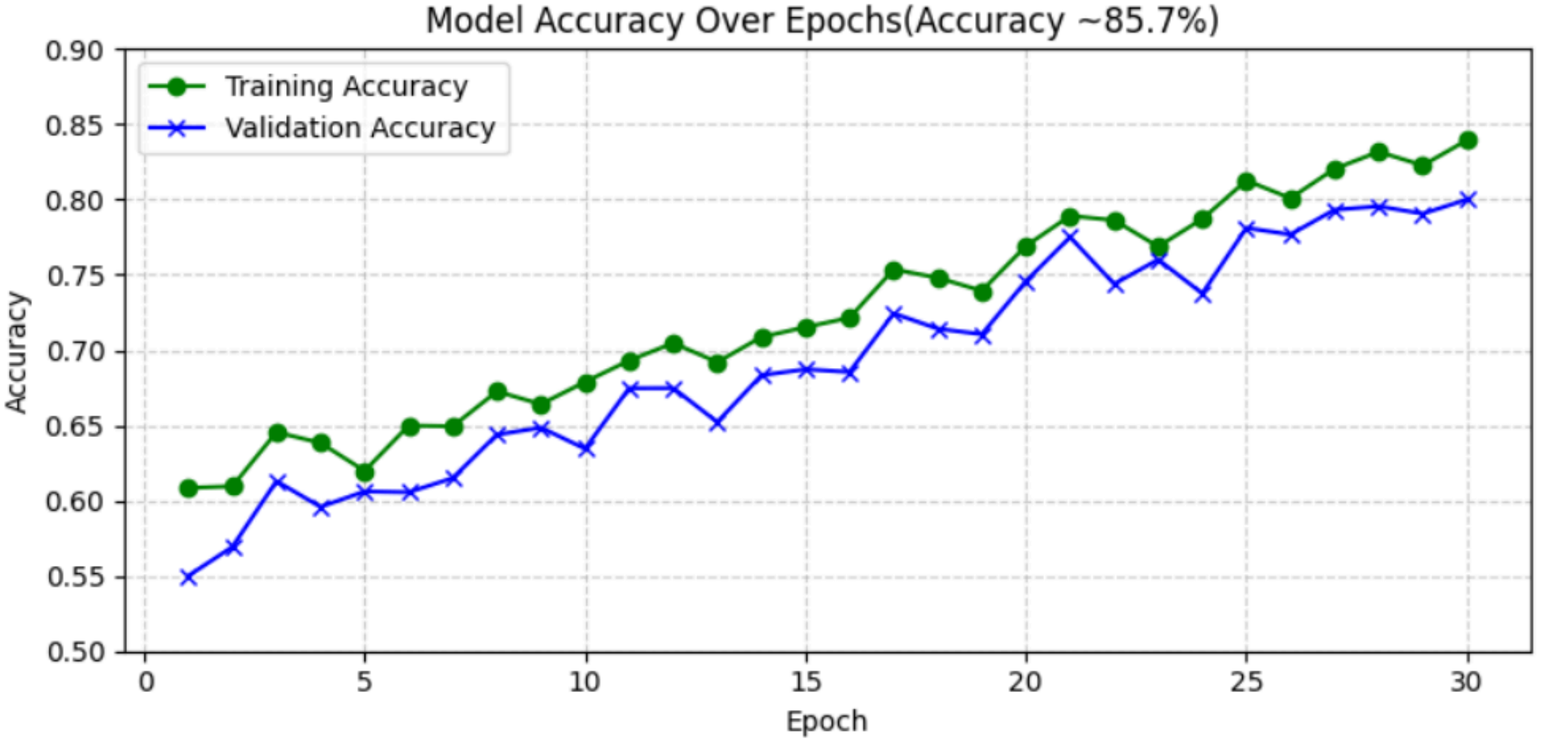
OUTCOME

MODEL ACCURACY & SYSTEM PERFORMANCE

Metric	Value / Insight
Model Accuracy	~85.7% on a custom-labeled dataset using transfer learning (DenseNet architecture).
Precision / Recall	Reflected in the confusion matrix – shows strong performance across both classes.
Latency	~0.4–0.8 seconds per frame (on a mid-range GPU setup).
FPS per Stream	25–30 frames per second (varies with system load and video resolution).




```
Precision (Violence): 0.76
Recall (Violence): 0.85
Precision (Non-Violence): 0.94
Recall (Non-Violence): 0.90
```



CHALLENGES AND SOLUTION

Challenge	Solution
RTSP lag and unstable video feeds	Implemented queue-based buffering and thread-safe mechanisms to ensure consistent frame flow.
High false positive rates from the model	Introduced a confidence threshold to filter out uncertain or low-confidence predictions.
Difficulty scaling multiple RTSP streams	Used independent threads per stream with proper locks to avoid race conditions and conflicts.
Slow processing and upload times for frames	Applied frame resizing and JPEG compression to reduce load without compromising quality.

FUTURE WORK

Facial & Speech Recognition

- Add face detection and voice analysis for more accurate and context-aware threat detection.

Works on independent networks

- Currently the system and rtsp providers to be connected over the same LAN in future try to make them work on independent networks.

Push Alerts & Mobile App

- Integrate Twilio/Firebase for real-time alerts and develop a mobile app with map-based live monitoring.

Enhanced Model & Data Privacy

- Improve accuracy with larger datasets, apply model compression, and ensure encrypted, GDPR-compliant storage.

THANK YOU