第二十四屆旺宏金矽獎

作品企劃書

觸發式危安事件監測系統

Trigger-based Security Event Monitoring System

參賽組別：應用組

參賽編號：A24-194

隊　　長：陳柏宏

隊　　員：黃亮維、賴宏達

1. Project Proposal:

1.1. Introduction

The proliferation of surveillance technology in public spaces has become a cornerstone for enhancing safety and security. Among these spaces, educational institutions represent a critical area where the well-being of students and staff can significantly benefit from advanced monitoring systems. Our project presents an innovative violence detection system, specifically designed to address safety concerns within the campus environment. Using 4K cameras and SOM (System-on-Module) (**Figure 1**), our system is capable of analyzing both visual and auditory data to detect instances of altercations, physical confrontations, and unusual gathering behaviors that may indicate potential threats.

The system's core functionality revolves around the real-time processing of high-definition video and audio streams, enabling a comprehensive analysis of the monitored environment. By incorporating advanced deep learning algorithms, the system discerns between normal activities and potential violence, effectively minimizing false alarms while ensuring prompt detection of genuine incidents. The integration of SOM enhances the system's ability to learn and adapt to new patterns of behavior, further refining its accuracy over time.

Incorporating SOM into our violence detection system provides a strategic advantage by optimizing the overall processing load. The integration allows for the preliminary analysis of data directly at the source, which effectively reduces the burden on central computing systems. This decentralized processing approach ensures that the central resources are utilized only when necessary, enhancing the system's efficiency by prioritizing the analysis of data segments identified as potentially significant. This methodological choice underscores our system's design philosophy, focusing on resource efficiency and the intelligent allocation of computational tasks, essential for maintaining a responsive and effective surveillance environment within educational campuses.

Figure 1. Surveillance camera in a classroom monitoring students from above.

While detecting violent behavior or abnormal crowd gathering, the system promptly notifies the monitoring personnel via website notification service. In addition, it automatically saves a 10-second video corresponding to the incident, ensuring that crucial evidence is preserved for further investigation or review. This feature not only facilitates immediate response but also aids in post-event analysis, contributing to the development of more effective safety protocols. Targeted primarily for application in educational campuses, our violence detection system offers a proactive approach to maintaining a secure environment. By providing real-time monitoring capabilities, the system ensures that security personnel can swiftly address and mitigate potential threats, thereby fostering a safer atmosphere for learning and academic activities.

It is important to acknowledge that while our system represents a significant advancement in surveillance technology, it is not infallible. The effectiveness of any security system depends on various factors, including the quality of implementation, the training of personnel, and the cooperation of the campus community. As we continue to refine our approach and incorporate feedback from real-world applications, we remain committed to enhancing the safety of our society.

1.2. Specification

1.2.1. SVC-4KF2E (4K Ultra HD Video Conference Camera)

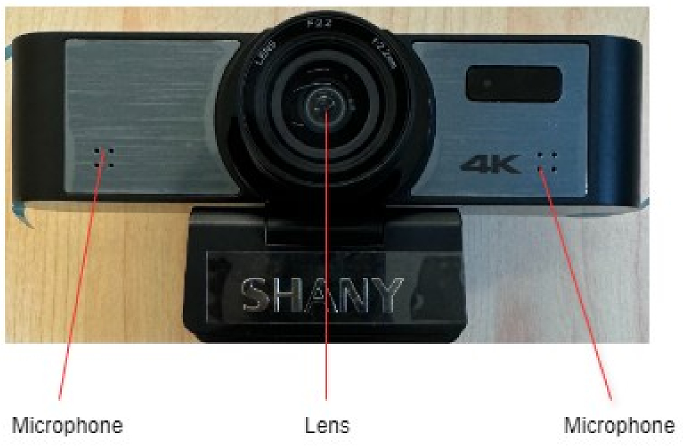

Figure 2. SVC-4KF2E (4K Ultra HD Video Conference Camera)

Product Specification		
Lens	Image Sensor	1/2.7" CMOS Sensor
	Lens	f2.2mm; Horizontal view: 110°
	Minimum Illumination	0.5 Lux @ (F1.8, AGC ON)
	Electronic Shutter	1/30s ~ 1/10000s
	White Balance	Auto, Indoors, Outdoors, One-push,

		Manual
	Digital Noise Reduction	Support 2D&3D noise reduction
	Back Light Compensation	Support
USB Functions	Back Light Compensation	Supports Windows 7 (1080p and below), Windows 8.1, Windows 10 operating systems; macOS TM 10.10 and above; Google™ Chromebook™ Version29.0.1547.70 and above; Linux (supports UVC)
	Hardware Requirements	2.4 GHz Intel® Core 2 Duo CPU or equivalent; 2 GB RAM or more; USB 2.0 port (supports USB 3.0)
	Encoding Format /Compression Format	H.264 / MJPEG / YUY2 / NV12
	Supported Resolutions and Frame Rates	4K@30/25fps; 1080p@30/25fps; 720p@30/25fps; 960x540p@30/25fps; 640x360p@30/25fps
	USB Video Class (UVC)	Support UVC 1.1
	UVC PTZ Control	Support (EPTZ)
	USB Interface	USB3.0*1 (Type-C)
General	Power Supply	DC5V (USB power supply)
	Current Consumption	600mA
	Operating Temperature	0°C ~ 40°C; Storage temperature: -40°C ~ 60°C
	Accessories	Remote control, Damping Rotation Mount
	Dimensions (WxDxH)	118mm x 37.2mm x 30.8mm
	Weight	800g

1.2.2. NVIDIA Jetson Nano

The NVIDIA Jetson Nano (**Figure 3**) is integral to our system's capability to manage and process massive computational loads in real-time violence detection. This powerful platform combines a quad-core ARM Cortex-A57 processor with a 128-core Maxwell GPU, enabling advanced machine learning applications that require real-time processing of audio and visual data streams. Using the MMPose, our system evaluates 26 key points per person per frame to detect aggressive movements, when the audio subsystem simultaneously analyzes speech patterns for potential verbal conflicts.

This dual-modality approach, underpinned by the Jetson Nano's substantial parallel processing power, facilitates a comprehensive surveillance solution. The synergy between the hardware's computational efficiency and our Python-based algorithms ensures that the system can evaluate nuanced patterns of behavior and respond instantaneously to potential security threats, providing not just surveillance but a proactive stance on incident prevention. The Jetson Nano's capacity for real-time processing ensures our system operates effectively under the critical latency thresholds that enable prompt detection and immediate notifications. This feature is paramount in scenarios where continuous monitoring and swift actions are essential, thus demonstrating the Jetson Nano's adaptability to settings demanding persistent

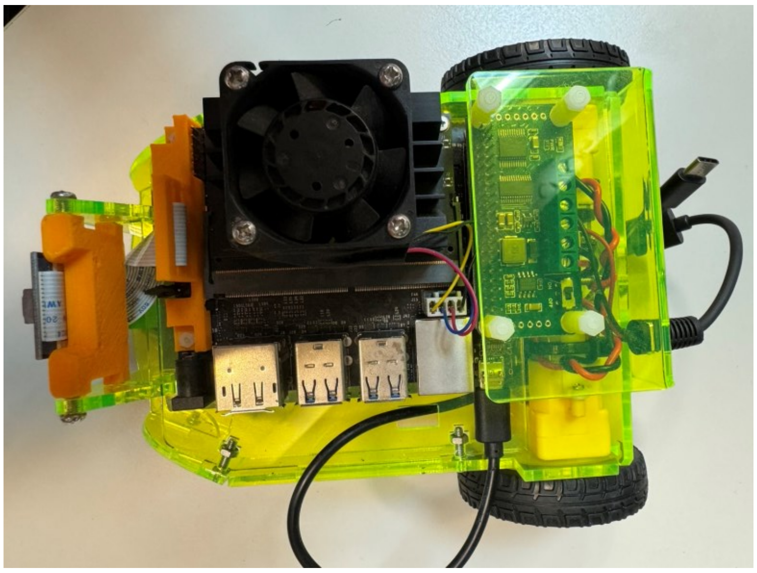
Figure 3. NVIDIA Jetson Nano

surveillance and expeditious responses.

Category	Description
GPU	NVIDIA Maxwell™ architecture with 128 CUDA cores
CPU	Quad-core ARM® Cortex®-A57 MPCore processor
Memory	4GB 64-bit LPDDR4
Storage	microSD (for system boot)
Video Output	1x HDMI 4K30 | 1x 1080p60 | 1x 1080p30 | 1x 720p30 (H.264/H.265)
Camera Input	1x 4K60 | 2x 4K30 | 4x 1080p60 | 8x 1080p30 | 18x 720p30 (H.264/H.265)
Connectivity	Gigabit Ethernet, M.2 Key E
Camera Interface	Two 15-position lanes MIPI CSI-2

Display Output	One HDMI 2.0 and one DP 1.2
USB	Four USB 3.0 Type-A ports, one USB 2.0 Micro-B
I/O	40-pin header (UART, SPI, I2S, I2C, PWM, GPIO), 12-pin Power header, 4-pin Fan header, 4-pin POE header
Peripheral Interfaces	Allows for expansion and modular application
Dimensions	100mm x 79mm x 30.21mm (supports wall mounting and brace installation)

1.3 Block diagram

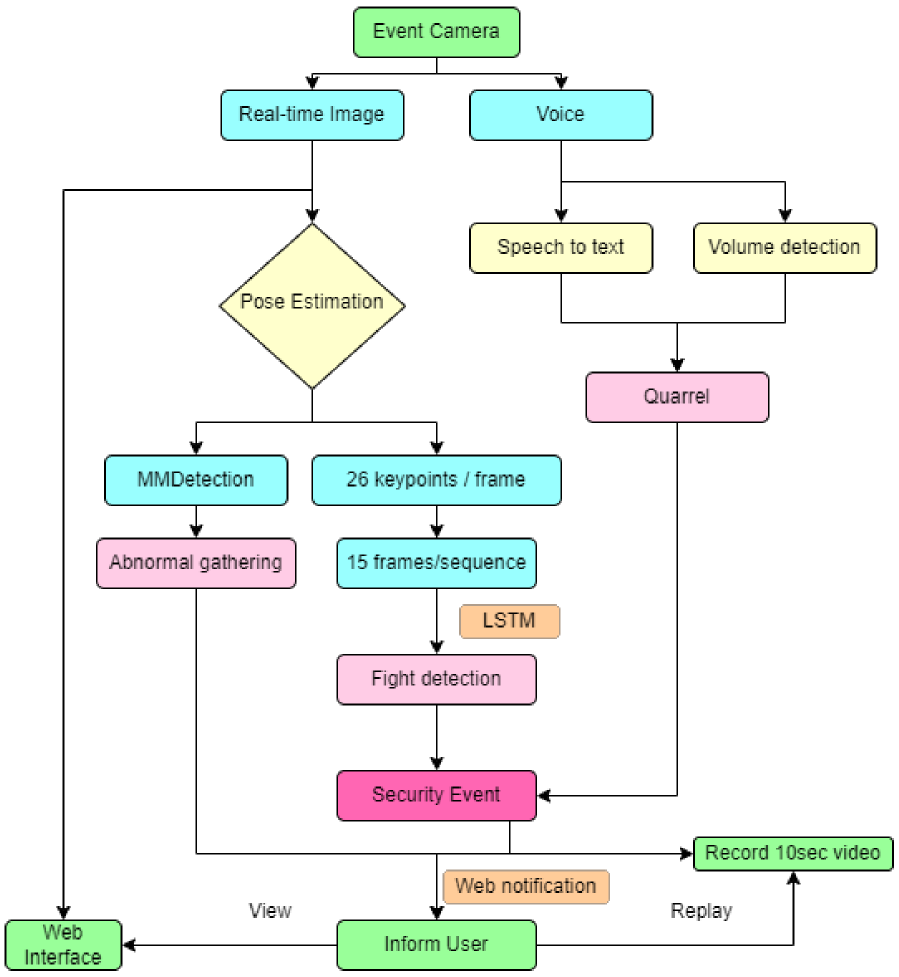

Figure 4. System Workflow

1.4 Work theory

1.4.1 Event camera

The Event Camera (SVC-4KF2E) is interfaced with the SOM (Jetson Nano) via a USB 3.0 connection, facilitating the transfer of audio (specifications: [insert specific audio format and details here]) and video at a frame rate of 30 fps (frames per second).

1.4.2 Voice processing

1.4.2.1 Speech-to-Text Conversion:

We use the Speech Recognition-Google model, which is an open-source, multi-speaker Chinese speech recognition model that is both free and allows for unlimited usage. We have determined that processing audio in five-second intervals is optimal for our use case, as it balances the need for timely foul words detection with system performance constraints. By monitoring the frequency of foul words, which serve as an indicator of potential disputes, within the dialogue, the system enables the implementation of preemptive measures to defuse escalating situations.

In our speech-to-text conversion module, Although this model may not achieve the highest accuracy in comparison with certain commercial alternatives, it is the only option that offers open-source multi-speaker recognition for Chinese, making it a uniquely viable choice for our system. As we look to the future, we plan to evaluate more precise models as they become accessible, with the intention of enhancing the accuracy of our speech recognition capabilities while retaining the advantages of our current setup.

We have compiled a database of common foul words and conduct comparisons after each speech recognition instance, including homophones with different characters. This detail can be integrated into the section discussing the implementation of the speech-to-text conversion module, highlighting the comprehensive approach taken to ensure the accuracy and relevancy of detected speech against potential disputes indicators.

1.4.2.2 Volume monitoring:

In the volume monitoring component, the intensity of sound signals is quantified by calculating the amplitude. This amplitude is then converted into decibel (dB) levels, a process pivotal for assessing environmental noise and potential disturbances. The formula utilized for this conversion is as follows:

$$dB = 20\log_{10}\left(\frac{V_{rms}}{V_{ref}}\right) \tag{1}$$

In our system, volume levels are comparative; sounds exceeding 70 decibels are categorized as loud (**Figure 6** and **Figure 7**). However, decibel detection is highly sensitive to all sounds, assigning lower weight in the rating system. The source of sound does not affect its assessment; intensity alone dictates its classification, serving as a supplementary factor in our analysis. This approach underscores the system's nuanced handling of audio data, prioritizing clear indicators of potential disturbances while acknowledging the limitations of decibel sensitivity in isolating specific sound sources.

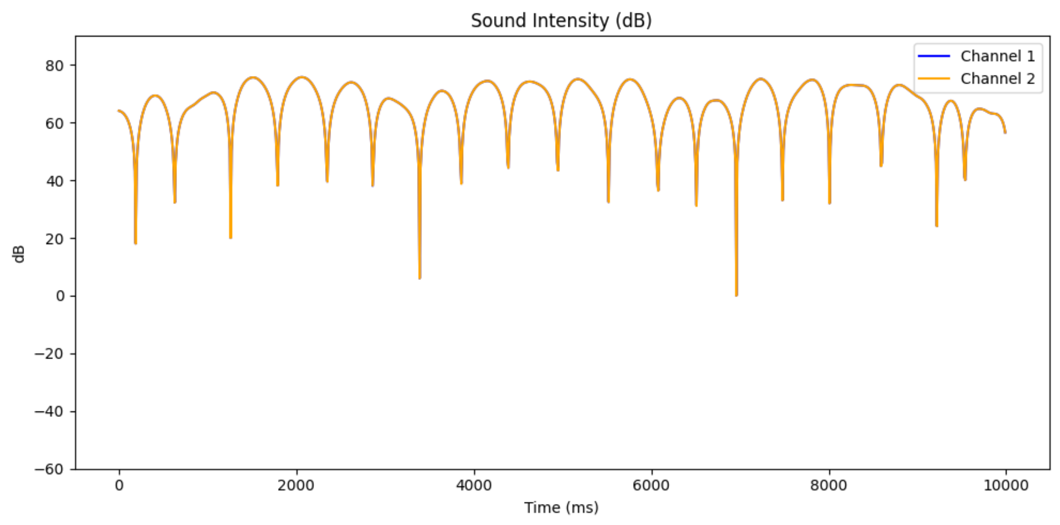

Figure 5. Graph of high intensity (Two channels)

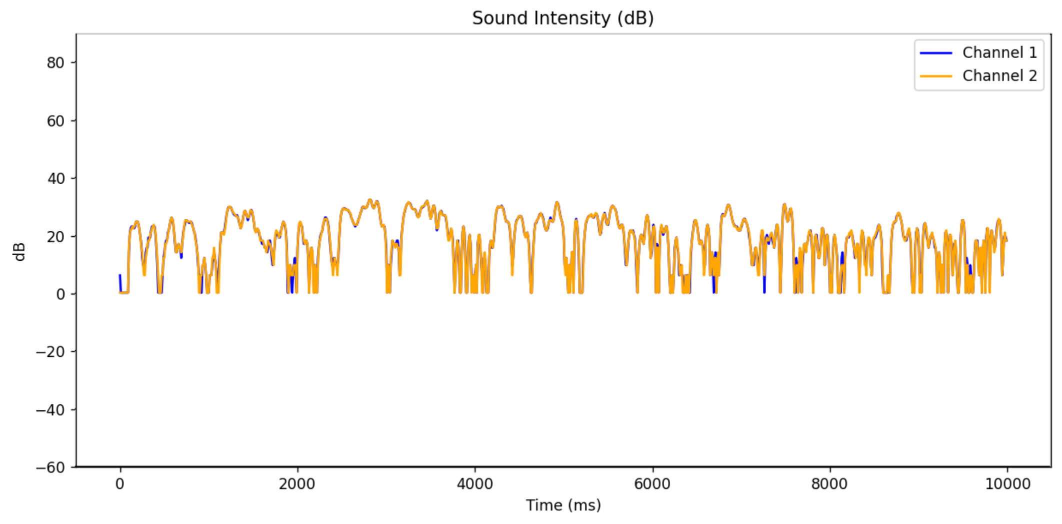

Figure 6. Graph of low intensity (Two channels)

1.4.3 OpenMMLab

OpenMMlab is a leading open-source system for computer vision algorithms in the era of deep learning. The architecture we use is shown in the following. MMEngine serves as a training framework that unifies the execution foundation in computer vision. It consists of a training engine, evaluation engine, and module management, providing a unified executor with a unified interface and customizable training processes. MMCV, as a data transformation module, preprocesses data including augmentation, scaling, etc., and converts data into dictionary format, facilitating data passing among various libraries with customizable options. With the support of these two components, MMPose and MMDetection can be utilized effectively. The flow chart of OpenMMLab is presented in **Figure 7**.

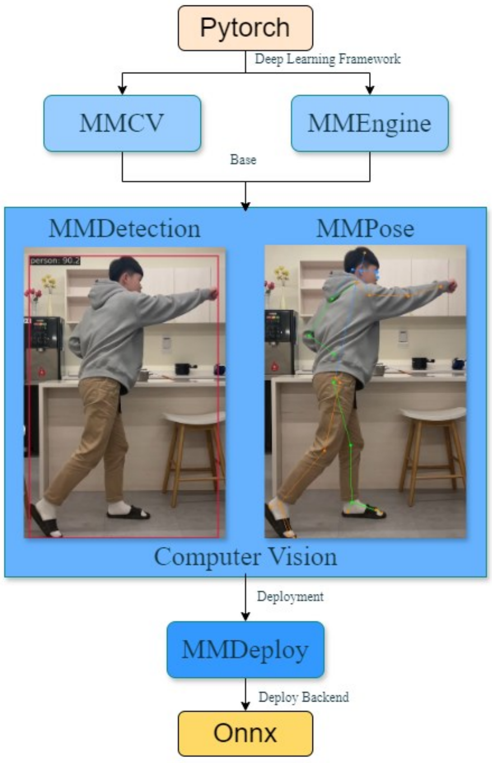

Figure 7. User exercising with AR glasses and wearing exoskeleton alongside a virtual coach.

1.4.3.1 MMDetection

1.4.3.1.1 Detection for Pose Estimation

MMDetection [1] stands as a versatile toolbox within the OpenMMLab, encompassing a comprehensive suite of object detection and instance segmentation algorithms. In our project, we employ the RTMdet-m [2] model, an efficient real-time object detector (**Figure 8**) with extensions to instance segmentation and rotated object detection, as the cornerstone of our detection framework. Its macro architecture is broken down into backbones, neck, and head. The

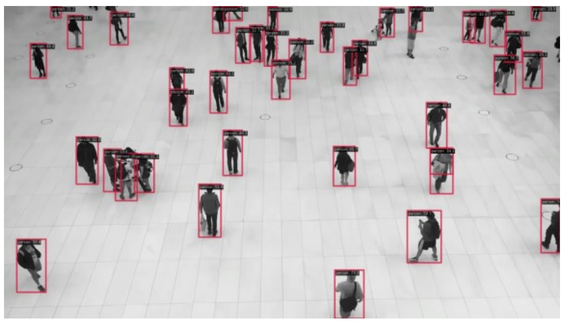

Figure 8. Real-time Multi-person Detection (RTM-Det) showcasing the detection of multiple individuals in a scene. (Image from Kaggle - Human Tracking & Object Detection Dataset)

backbone adopts CSPDarkNet, consisting of several stages with basic building blocks, and the neck utilizes a feature pyramid with bottom-up and top-down propagation to enhance the feature map. This general architecture is applicable to both standard and rotated object detection and can be extended for instance segmentation with additional heads. Also, its detection is built upon a typical one-stage object detector. It enhances model efficiency by integrating large-kernel convolutions in the backbone and neck, which are then balanced in depth, width, and resolution to optimize model efficiency. Soft labels are incorporated into dynamic label assignment strategies to boost accuracy, coupled with improved data augmentation and optimization strategies.

1.4.3.1.2 Detection for Abnormal Crowd Gathering

In addition to being a prerequisite for key points detection, detection also provides vital information on the number of individuals present, allowing crowd density to be monitored. Any unusual surge in density requires heightened attention to ensure absolute safety. The phenomenon of unusual crowd gatherings, while seemingly benign in certain circumstances, harbors inherent risks that cannot be overlooked. When large numbers of individuals assemble without a clear purpose or in informal gatherings, the behavior of the group can swiftly change due to various triggers, even if the initial intentions were peaceful. This capability is crucial for preemptively identifying and mitigating potential risks associated with dense gatherings.

1.4.3.2 MMPose

The MMPose [3] employs a top-down approach, utilizing a pre-existing detector to obtain bounding boxes, followed by independent pose estimation for each detected individual. In our system, we utilize RTMpose-m, one of the MMpose models focusing on real-time tasks. The RTMPose framework advances the field of real-time human pose estimation (**Figure 9**) by skillfully combining mmWave radar data with the innovative CSPNeXt architecture. This integration is meticulously designed to harness the strengths of both modalities, where the efficient

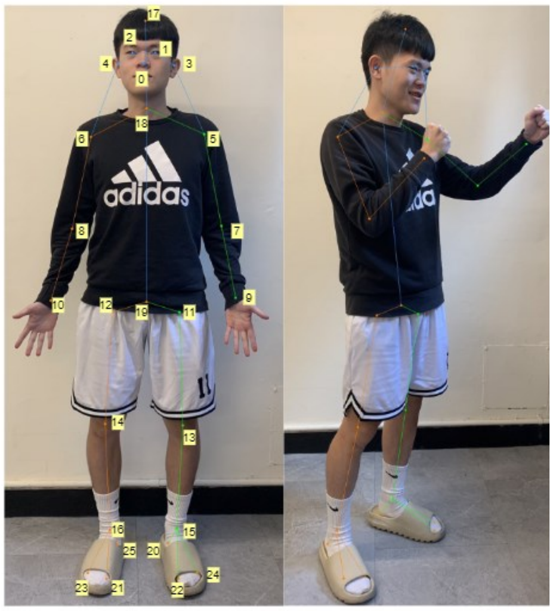
Figure 9. Real-time Pose Estimation

processing of radar signals is complemented by the depth and clarity provided by visual data. The training of RTMPose is predicated on a comprehensive dataset, meticulously annotated to capture a wide spectrum of human poses and movements, ensuring the model's adaptability to diverse real-world scenarios. By leveraging CSPNeXt, known for its exceptional balance of speed and accuracy, RTMPose undergoes a rigorous optimization process. This process is carefully calibrated to enhance the model's ability to perform instantaneously while maintaining high levels of precision, making it particularly suited for applications where real-time feedback is crucial. The seamless melding of these technologies within RTMPose not only signifies a leap in pose estimation capabilities but also sets a benchmark for future innovations in the domain, promising unparalleled accuracy and efficiency in real-time applications.

1.4.4 Real-Time imaging: Pose Estimation & Web Interface

The camera transmits imagery at 30 frames per second (fps) to the SOM, and to reduce overall computational demand, the frame rate is down sampled to 15 fps for pose estimation. Detected skeletal information by MMpose model is overlaid on each frame before being uploaded online. The Jetson Nano board acts as a web server, displaying videos with key points on a webpage (**Figure 10**). This allows users to access the camera augmented with skeletal data by connecting to a specific IP address via a computer.

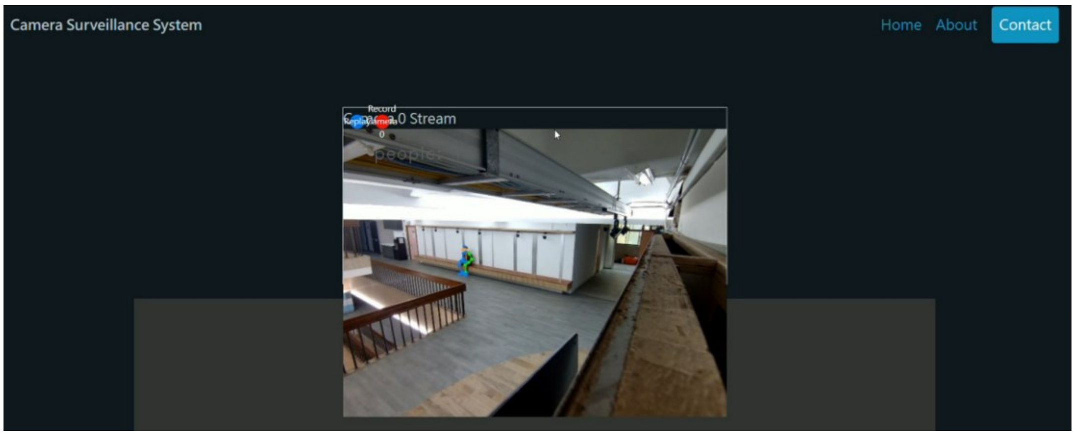

Figure 10. Web interface

1.4.5 LSTM-Fight detection

Each frame captures data on 26 skeletal key points for each individual, highlighting both spatial and temporal relationships. These key points act as markers of human gestures and movements, facilitating the identification of unusual behavior patterns in the video feed. To manage computational load, especially considering that the camera operates at 30 frames per second (fps), the system down-samples the data to 15 fps. This reduction in frame rate significantly decreases the volume of data processed without compromising the ability to detect critical behavioral anomalies. Utilizing a sliding window technique, where each segment covers 15 frames and advances by 5 frames for each analysis, the system effectively maintains continuous monitoring over time. Through the application of a sophisticated Long Short-Term Memory (LSTM) model, the sequence of skeletal key points is analyzed to determine if the crowd's behavior strays from normal patterns. This LSTM model is adept at identifying signals that may suggest escalating violence or conflict, thereby categorizing observations into binary outcomes: potential threats or non-threats. Such binary classification is instrumental in supporting the decision-making process for security measures, guaranteeing prompt action against potential dangers.

1.4.5.1 LSTM Algorithm

Long Short-Term Memory (LSTM) [4] is a type of recurrent neural network (RNN) architecture used in the field of deep learning. LSTMs are designed to address the issue of long-term dependencies and vanishing gradient problem which conventional RNNs struggle with. They are particularly well-suited for processing sequences of data for applications such as time series analysis.

LSTM networks are composed of memory cells (**Figure 11**) and various gating mechanisms that control the flow of information within the network. Each LSTM cell contains a cell state, which serves as the "memory" of the cell and can store information over long sequences. The key components of an LSTM cell are:

- X_t : Input data vector
- h_t : Hidden state vector
- C_t : Cell state vector
- h_{t-1} : Hidden state from previous output
- C_{t-1} : Cell state from previous output
- W and U : Weight matrices for input and hidden state vectors (subscripted with I , f, o, c to represent input gate, forget gate, output gate, and cell candidate vector respectively)
- b: Bias vectors (subscripted similarly to the weight matrices)
- σ: Sigmoid activation function
- tanh : Hyperbolic tangent activation function

Forget Gate:

$$f_t = \sigma\,(W_f * x_t + U_f * h_{t-1} + b_f) \qquad (2)$$

This gate decides which information to discard from the cell state. By taking the previous hidden state h_{t-1} and the current input x_t, it applies a sigmoid function (σ) to generate a forget gate vector (f_t). This vector has values between 0 and 1, indicating how much of each component of the cell state should be forgotten (where 0 means "completely forget" and 1 means "completely retain").

Input Gate:

$$i_t = \sigma\,(W_i * x_t + U_i * h_{t-1} + b_i) \tag{3}$$

$$\check{C}_t = \tanh\,(W_c * x_t + U_c * h_{t-1} + b_c) \tag{4}$$

The input gate decides what new information will be stored in the cell state. It consists of two parts: a sigmoid layer that determines which values in the cell state to update, and a tanh layer that creates a vector of candidate values ($\check{C}_t$) to be added to the state. The input gate (i_t) scales the candidate values, effectively selecting the new information to be included.

Cell State Update:

$$C_t = f_t * C_{t-1} + i_t * \check{C}_t \tag{5}$$

This equation updates the cell state by combining the old state (C_{t-1}) scaled by the forget gate (f_t)—effectively forgetting things as decided—and the new candidate values ($\check{C}_t$) scaled by the input gate (i_t)—adding new information as decided. The updated cell state (C_t) carries forward through time, encapsulating both new and old information deemed relevant.

Output Gate:

$$o_t = \sigma\,(W_o * x_t + U_o * h_{t-1} + b_o) \tag{6}$$

$$h_t = o_t * \tanh\,(C_t) \tag{7}$$

The output gate determines the next hidden state (h_t), which will be passed to the next time step of the LSTM and can also be used as part of the output. The gate decides which parts of the cell state (C_t) are output based on the current cell state and the past hidden state, applying a sigmoid function to generate the output gate vector (o_t). The final hidden state (h_t) is obtained by scaling the tanh of the cell state (C_t) with the output gate, controlling the information flow to the next LSTM cell and potentially to the output of the network.

Green part : the operator for generating forget gate, input gate, and output gate.
White part : the calculation for generating forget gate, input gate, and output gate.

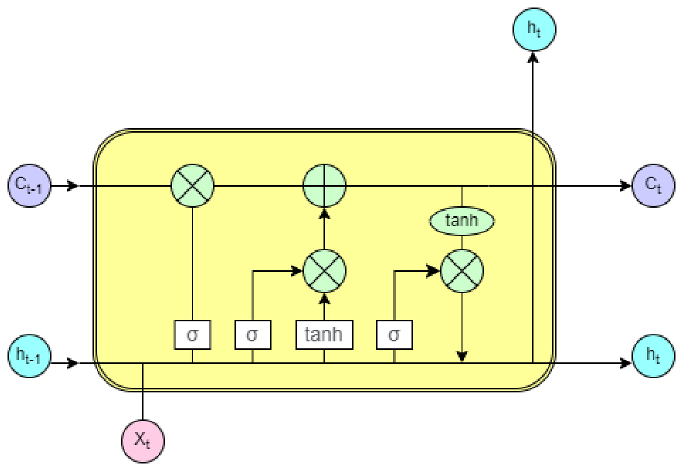

Figure11. LSTM cell structure

In conclusion, LSTMs improve upon the limitations of RNNs in processing sequence data, especially in handling long sequences and capturing long-term dependencies, making them a more preferred model for numerous sequence processing tasks. This inherent strength of LSTMs in managing long-term dependencies and their ability to efficiently process sequential information make them particularly suited for our task [5]. The capability to remember and utilize past information over extended sequences is crucial for understanding the temporal dynamics and nuances in our data, ensuring more accurate and insightful outcomes.

1.4.6 Notification

Notifications are delivered via web alerts for three scenarios: suspected quarreling, potential fighting behavior, and unusual crowd gathering. Upon receiving an alert, the monitor can access the webpage to view the live feed. Simultaneously, the system starts saving a 10-second video clip for further review. This integrated approach ensures prompt and efficient incident management, enhancing the response to potential safety threats.

Quarrel : If four consecutive sentences(5sec) detected match words from our database of foul words, and there are fifteen instances(10ms) within these twenty seconds where the decibel level exceeds 70, a quarreling notification is triggered.

Abnormal Crowd Gathering : In our selected scenarios(vary when environment changes), if the scene displays more than ten individuals, an alert for unusual crowd gathering is sent.

Fight Detection : Each dataset comprises 15 frames (down sampled), with each iteration shifting forward by five frames. If, within five consecutive datasets, any set is identified by LSTM model as fighting behavior, this condition triggers the sending of a notification.

The goal of our system is to provide early warnings of violent incidents as they occur or before they escalate, aiming to promptly intervene and prevent further harm. Core focuses are on detecting fighting behavior and unusual crowd gatherings, with quarrel detection serving as an auxiliary element, considering that verbal disputes often precede physical conflicts.

Upon receiving a notification, user can not only view the live scene immediately on the webpage but also have access to a 10-second instant video replay provided by the system for further review.

2. Detailed Design Data Files

2.1. Source code

2.1.1 Class Fight_Detect()

2.1.1.1 Pose Estimation

- **visualize():**

Pose Connections and Color Configuration: Determining pairs of skeletal points to connect, the color palette, the colors of the links, and the colors of the points based on the **skeleton_type** configuration(**'halpe26'**).

Extraction of Keypoints and Scores: The method processes the result**s** parameter to extract keypoints, bounding boxes, and confidence scores for each detected pose in the frame.

Drawing Skeletal Connections: Iterating through detected keypoints, connections between keypoints are drawn on the video frame if their scores exceed 0.5(threshold). Lines and circles represent skeletal connections and keypoints, respectively, with colors specified by the configuration.

Recording Keypoint Data: For further analysis, keypoints and their scores are appended to a list (kp), and it serves as the input to the LSTM model for prediction of fight.

Overlaying Information: The total count of detected people and the frame rate (FPS) are optionally added as text overlays on the frame to provide additional context to the viewer.

- **generate_frames(camera_id):**

Camera Initialization: Checks if the specified camera_id is already recorded in self.camera_record**s**. If not, it initializes the camera for video capture and adds it to the record with its recording status set to False.

Pose Tracking Setup: Initializes a PoseTracker instance with pre-defined detection and pose models, setting up for real-time pose estimation on the video stream.

Frame Processing Loop: Continuously reads frames from the camera. For each frame:

a. Checks for successful frame capture; if not successful, exits the loop.

b. Applies the pose tracker to the frame, obtaining results that include detected keypoints (pose information).

c. Calls the visualize method to overlay visual representations of detected poses onto the frame, along with any other specified visual enhancements (like counting people in the frame).

d. Keeps track of the number of processed frames and calculates the frames per second (FPS) rate, adjusting for real-time performance monitoring.

e. Optionally, if recording is enabled for the camera, writes the processed frame to a video file through an FFmpeg process (video_writer).

Streaming the Processed Frames: Encodes the processed frame into JPEG format and yields a byte stream formatted for HTTP multipart response. This allows the frames to be continuously sent to and displayed on a web interface, providing a real-time view of the camera feed with pose detection overlays.

- **send_notification(message):**

When condition is reached, a notification message using Socket.IO in a Flask application. is emitted to all connected clients via WebSocket, carrying a payload that contains the message.

2.1.1.2 Web Interface developed with Flask

Although libraries like Django, which contain rich backend resources, are available, we opted for a Flask application for rapid deployment, enabling early testing of the system prototype. Leveraging OpenCV, we built a network camera streaming service, with web pages dynamically adjusting to accommodate the number of connected cameras. FFmpeg ensures proper encoding of stored videos for replay using HTML5's new features. Most backend functionalities are initiated by users through the web interface, including starting and stopping video streams and managing video files. While running on a local host, this web application serves on port 5000 of localhost. To access this service from anywhere on the internet, we initially used ngrok to securely expose the local service for third-party testing. After deploying the model to a Linux-based edge system and converting it into a server, we briefly utilized LocalTunnel to swiftly verify the system's feasibility. Due to its low memory requirements and the autonomous nature of detecting violent incidents, utilizing an edge system as a server proves feasible.

2.1.1.2.1 Backend: Flask

- **start_video_writer(camera_id):**

Initializes and starts an FFmpeg process to write the video stream from the camera to a disk file.

- **detect_cameras(limit=10):**

  Detects and returns a list of available camera IDs in the system.

- **start_recording(camera_id):**

  Initiates recording of the video stream from a specific camera.

- **stop_recording(camera_id):**

  Stops recording the video stream from a specific camera and closes the associated FFmpeg process.

- **list_videos(camera_id):**

  Lists all recorded video files for a specific camera.

- **video(camera_id):**

  Returns the real-time video stream from the specified camera.

- **index():**

  Renders the main page, displaying all detected cameras and providing interfaces for starting and stopping recording.

2.1.1.2.2 Frontend: HTML, CSS, JavaScript

HTML

Navigation Bar (**<nav>**):

- Displays the system title "Camera Surveillance System".
- Includes hyperlinks to the homepage, about page, and contact page.

Main Content Area (**<main>**):

Camera Container (**<div class="camera-container">**):

- Displays the video streams of all cameras using flexbox.
- Applies styles to the video stream containers, such as borders and hover effects.

For each camera stream:

- Includes buttons for replay (**<button class="replay-button">**) and recording (**<button class="recording-button">**).
- Displays the real-time video stream using **<img>** tag.

Video Replay Modal (**<div class="modal">**):

- Displays a modal when the replay button is clicked, allowing users to select and watch recordings from the camera.
- Each modal includes a close button (**<span class="close">**) and a **<ul>** to display the recording list.

Video Player (**<video id="videoPlayer">**):

- Controls the playback of selected recordings.

CSS Styling

- Defines the width, height, and borders of the camera video stream containers, as well as hover animation effects.
- Replay and recording buttons have defined positions, sizes, colors, and circular styles.
- Modals have corresponding styles for background color, borders, and positions, as well as a fade-in and fade-out animation effect.

JavaScript to activate the backend

showReplayModal(modalId, cameraId):

- This function is used to display a modal (popup) when the user clicks the replay button below each camera video.
- The function requests the recording list of the camera from the server via the fetch API, and dynamically generates a selectable list of recording files in the modal.

playVideo(cameraId, videoFileName):

- When the user selects a recording file from the modal, this function is called.
- The function updates the src attribute of the <video> tag to point to the selected recording file, and controls the video player to play the selected recording.

closeModal(modalId):

- This function is used to close the opened modal.

blinkReplayButton(button):

- This function is used to create a blinking effect on the specified replay button, simulating a flashing animation.

startRandomBlinkingForButton(button):

- This function sets a random interval for each replay button to start the blinking effect, creating a visually random blinking effect.

toggleRecording(cameraId, isRecording, button):

- This function is called when the user clicks the recording button.
- It starts or stops recording based on the current recording status of the camera, and updates the display text of the button accordingly.

DOMContentLoaded event listener:

- This listener checks whether the browser supports desktop notifications after the document is fully loaded.
- If supported, it requests permission from the user to display notifications and displays a notification after permission is granted.

2.1.2 Class AudioProcessor()

2.1.2.1 Speech-to-text

- **sperec():**

Continuously capturing audio data from the microphone and attempting to recognize speech using Google's speech recognition service. It listens for audio within a set timeout period, then tries to recognize any spoken words and prints the recognized text. If the speech cannot be recognized or if there's an error with the speech recognition service, it will print an appropriate message.

2.1.2.2 Voice intensity

- **intensitygraph():**

  For extracting and processing audio data to analyze its intensity, potentially for applications that monitor environmental noise levels, analyze speech loudness, or any other scenario where audio signal intensity is of interest. The calculations make the sound intensity information more accessible and understandable by converting it to decibels, a more commonly used and standardized measure of sound intensity.

2.1.3 Class PersonKeypoints()

- **self.count:**

  A counter used to process every other frame (if self.count % 2 == 0:). This reduces the computational load by down sampling the data.

- **Frame Skipping Logic:**

  For every alternate frame, it updates self.count and proceeds to process the keypoints. This approach helps in reducing the processing for every frame and hence saves computational resources.

- **Change in Number of People Detected:**

  It checks if the number of people detected (len(keypoint_person)) has changed compared to the previous frame (self.prenumper). If there's a change, it calls self.refresh() to reset the tracking data, preparing for new data collection.

  If the number of people detected has exceeded the threshold(10 in our setting), then will call the function send_notification().

- **Recording Keypoints:**

Initial State (No Keypoints Recorded Yet): If this is the first set of keypoints being recorded (len(self.keypoints_15_list) == 0), it validates and stores keypoints for each detected person.

Subsequent States: For subsequent frames, it continues to append new keypoints data for each person, tracking changes over time.

- **Validity Check of Detected Persons:**

  The _is_valid_person function checks whether a detected person has all the required keypoints with valid (x, y) coordinates. This ensures that only complete and valid data sets are processed.

- **Data Collection for Pose Tracking:**

For each valid detected person, keypoints are appended to self.keypoints_15_list, which

stores the sequence of poses for analysis.
A self.person_count dictionary tracks the number of data points collected for each person.

- **<u>Refresh on Invalid Data</u>:**

  If a person's keypoints are incomplete or missing, the method calls self.refresh() to clear the current data and start fresh, ensuring the reliability of the tracking and analysis.

- **<u>Incident Detection</u>:**

The method calls self.detect() to check if enough data (window_size=5) has been collected for reliable analysis.
It then calls self.fight_detect() to analyze the collected keypoints data and determine if fight actions have been detected by LSTM model based on the pose information and its changes over time.

2.2 Self-documented coding style

Given that the entire project was accomplished through the collaborative effort of our team, with various members responsible for different segments of the work, we prioritized the readability and maintainability of the code by adopting a self-documenting coding style. This approach involves the use of readable variables, clear logic, and a well-structured hierarchical coding plan. We also strive to minimize the time and space complexity of the code to enhance computational efficiency, while incorporating comments to facilitate ease of reading and understanding.

Finally, each functionality was encapsulated within its own class, allowing for organized and modular code. Consequently, all functionalities could be efficiently invoked from the main.py file. This coding methodology not only facilitated smoother teamwork by making the code more accessible and understandable to all team members but also enhanced the overall development efficiency. It allowed for easier debugging, quicker onboarding for new team members, and streamlined the process of future code modifications or additions.

3. Verification Results and Materials

3.1 Demonstration

3.1.1 Fight detection

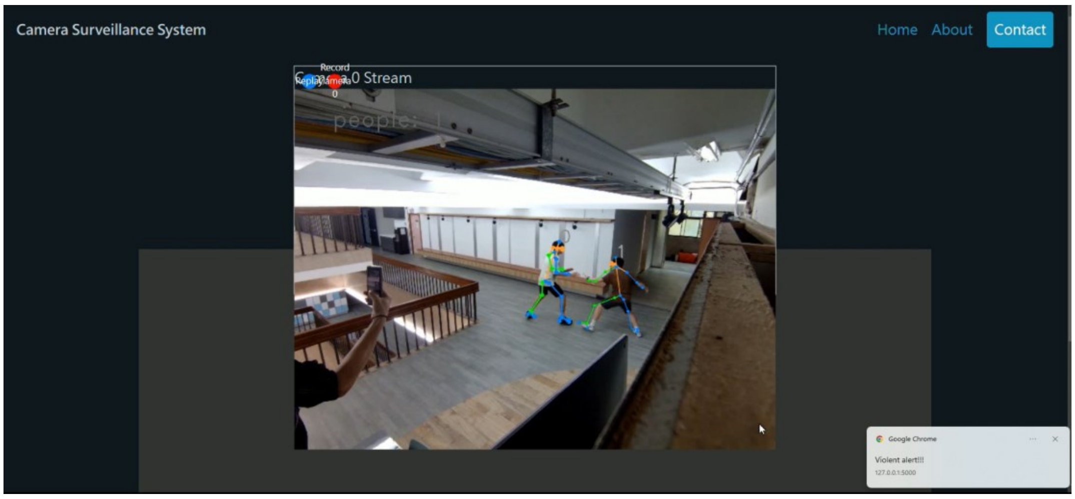

Figure12. Fight detected by the system

3.1.2 Abnormal crowd gathering

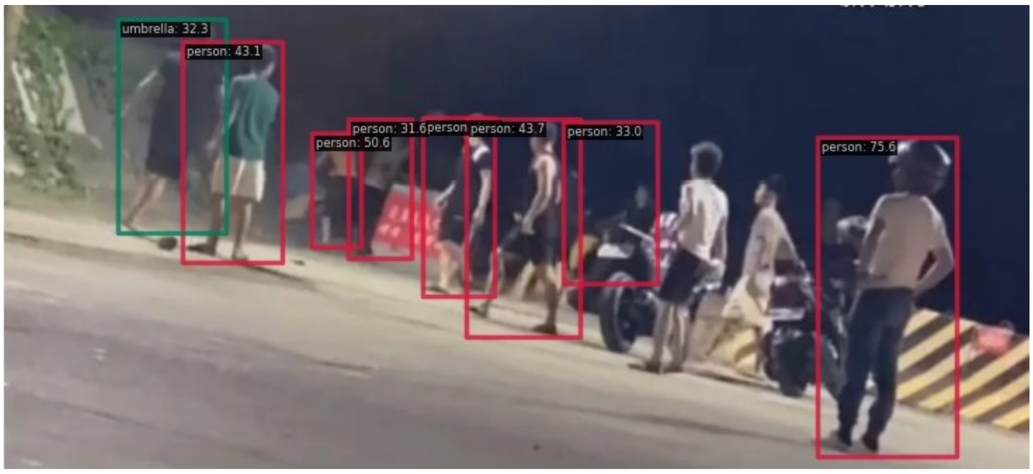

Figure13. Abnormal crowd gathering by the system

3.2 LSTM model performance

Data is the cornerstone of all learning algorithms. However, producing large-scale annotated data is a very laborious and costly task. YouTube is an excellent source for collecting video data. However, according to their policy, videos that are bloody, disturbing, and violent should not be uploaded, and the remaining relevant information is also obscured by mosaics. Additionally, open-source datasets available online or those used in similar studies suffer from blurry image quality, resulting in motion capture models being unable to recognize violent actions. These factors have greatly limited the ability to download and collect footage of weapon-related violence scenes. These considerations prompted us to decide to record our own dataset.

The scenes we recorded are divided into two types. The first type (**Figure 13.**) involves filming actor from the front with smartphones and tablets, with the actor positioned centrally in the frame. They perform both violent actions, such as punching, kicking, pushing, elbowing, and grasping, while non-violent actions with slower speeds, such as walking, sitting, running, and stretching. The actor also rotated at different angles within the filming range to increase the dataset size and enable the model to learn from various perspectives. The second type (**Figure 14.**) of scene takes place in a rectangular area surrounded by four cameras positioned at each corner, angled slightly downward from above. The actor performed actions in the center, including both violent and non-violent actions.

With four cameras capturing simultaneously from different angles in each frame, and the actor changing direction and rotating in the middle, the dataset provides comprehensive angle information for model training.

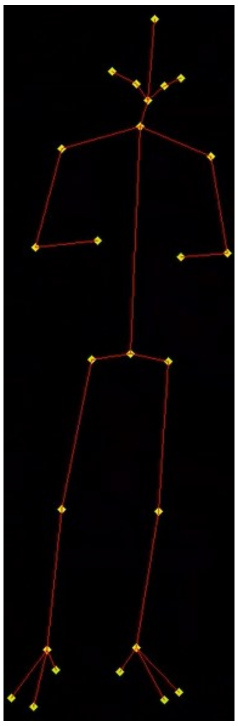
Figure 14. Facing the camera

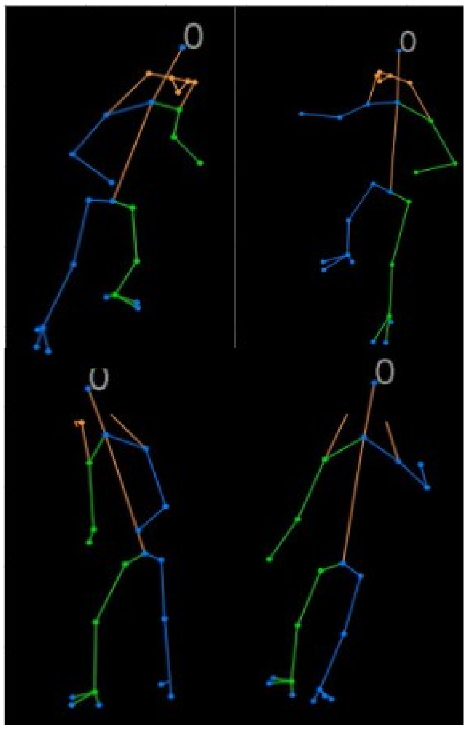

Figure 15. Four angles

The data is organized into sequences, each consisting of 30 frames, where each frame contains 26 keypoints represented by (x, y) coordinates. The starting frame of each subsequent sequence is shifted by 10 frames from the previous one, creating a sliding window of 10 frames.

class	Violence	Non Violence	Total
number of sequences	2072	1975	4047

Table 1. Data distribution

LSTM model layers

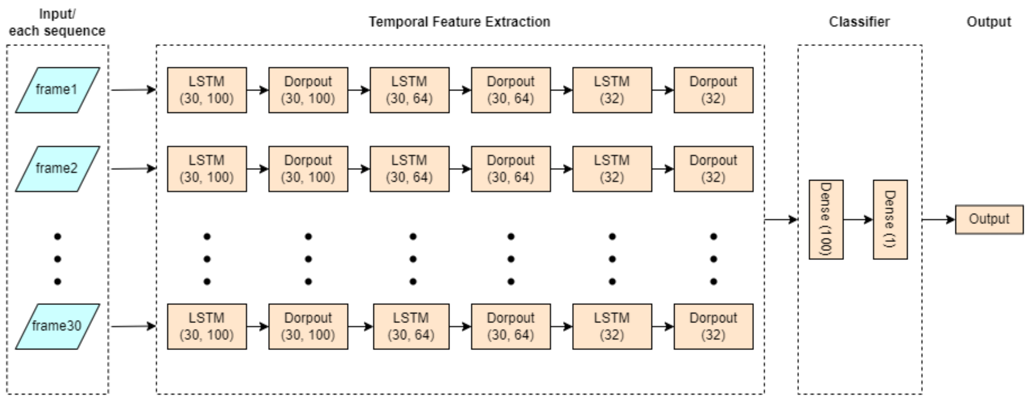

Figure 16. Our model structure

Our model summary

Layer (type)	Output Shape	Param
LSTM_1	(None, 30, 100)	61200
Dropout_1	(None, 30, 100)	0
LSTM_2	(None, 30, 64)	42240
Dropout_2	(None, 30, 64)	0
LSTM_3	(None, 32)	12416
Dropout_3	(None, 32)	0
Dense_1	(None, 100)	3300
Dense_2	(None, 1)	101

Total parameters: 119257

Trainable parameters: 119257

Non-trainable parameters: 0

Table 2. Model summary

Model performance

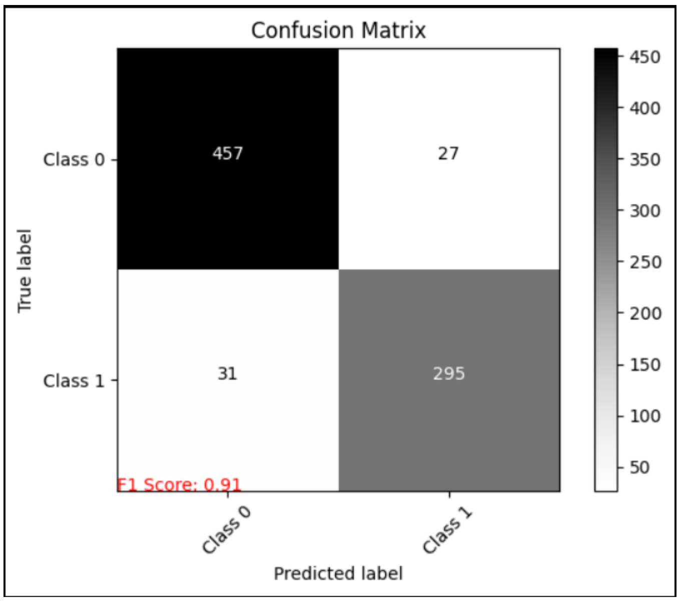

Figure 17. Confusion Matrix of the LSTM Model. True positives and true negatives for class 0 and class 1 show the model's performance in real-time multi-person detection, achieving an F1 Score of 0.91.

3.4 The art work of the project

In our system, we strive to present a user interface that epitomizes simplicity, allowing users to grasp key information at a glance. The notification system employs common web notifications, which users can receive as long as they are connected to the internet, even without actively viewing the webpage. This feature significantly enhances user convenience. Moreover, the post-processed skeletal information displayed on the monitor can be customized according to user needs, transitioning into a detection display that can, for instance, count the number of people. This adaptability not only enriches the user experience but also extends the functionality of traditional surveillance approaches.

4. Summary Report

4.1 Summaries of the project

We have developed an advanced detection system that utilizes computer vision and deep learning to identify precursors to conflict. Distinct from other fight detection project[6-9], our system harnesses the power of 2D skeletal point coordinates to comprehend spatial and temporal dependencies, requiring only a single camera and a System-on-Module (SOM) to operate effectively as a full-fledged system. Also, we can not only work as a surveillance camera but also have the ability to prevent violence spread. In light of recent regrettable incidents within school environments, our primary objective has been to tailor the system to these settings, aiming to ensure the safety and well-being of students and staff.

To enhance our system's sensitivity to the nuances of human interaction, we have integrated audio analysis, which scrutinizes vocal tone and volume variations, aiding in the early identification of potential verbal confrontations. This dual analysis of sight and sound creates a layered approach to conflict prediction, facilitating the de-escalation of situations before they intensify.

Our solution is especially designed to be adaptive to different indoor scenarios and diverse lighting conditions, providing reliable performance in the dynamic school setting. It is capable of discerning normal interactions from potential threats, thereby empowering educational institutions with the tools to respond swiftly and appropriately to maintain a peaceful and secure learning environment.

4.2 Proposal for future plan as a result of the project

In our project, there are critical areas identified for improvement. Firstly, the optimization of the model and algorithm is necessary due to computational constraints. Originally, the LSTM sequence was set for 30 frames per second (fps), but because pose estimation could only process at 15 fps, we had to down sample. Although the performance of the trained model did not significantly deteriorate and the accuracy in practical discrimination was high, we believe that a more comprehensive skeletal information could enhance the model's performance further.

Secondly, the aspect of speech recognition requires advancement. The powerful speech recognition models available today rely on extensive databases. Training a model fully tailored to our task is beyond our current capability, so we had to settle for a readily available open-source model. However, these models are primarily designed for single-speaker scenarios, while our application is oriented towards multi-speaker situations such as arguments. To access more accurate multi-speaker models that require substantial translation volumes, payment is necessary, which was not feasible for us. Consequently, we opted for the Google model within the speech recognition library, which does not offer the highest accuracy. Therefore, we aim to build our database and train our model to optimize speech recognition.

Lastly, our goal is to apply our system across various settings—public spaces, streets, restaurants—replacing traditional surveillance systems that are often relegated to playback functionality, leaving security personnel a step behind and only able to react to situations, rather than prevent them. We also contemplate integrating speakers for remote broadcasting or issuing alarms to deter the onset of conflicts. These enhancements could transform our system into a proactive tool for maintaining public order and safety.

5. References

1. Chen, K., et al. (2019). "MMDetection: Open mmlab detection toolbox and benchmark." arXiv preprint arXiv:1906.07155.
2. Lyu, C., et al. (2022). "Rtmdet: An empirical study of designing real-time object detectors." arXiv preprint arXiv:2212.07784.
3. Sengupta, A., et al. (2020). "mm-Pose: Real-Time Human Skeletal Posture Estimation Using mmWave Radars and CNNs." IEEE Sensors Journal 20(17): 10032-10044.
4. Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory." Neural computation 9(8): 1735-1780.
5. Sharma, A. and R. Singh (2023). "ConvST-LSTM-Net: convolutional spatiotemporal LSTM networks for skeleton-based human action recognition." International Journal of Multimedia Information Retrieval 12(2).
6. Fu, E. Y., et al. Automatic Fight Detection Based on Motion Analysis, IEEE.
7. Akti, S., et al. Vision-based Fight Detection from Surveillance Cameras, IEEE.
8. Vijeikis, R., et al. (2022). "Efficient Violence Detection in Surveillance." Sensors 22(6): 2216.
9. Nadeem, M. S., et al. (2019). WVD: A New Synthetic Dataset for Video-Based Violence Detection, Springer International Publishing: 158-164.