Temporal Pose Analysis for False Positive Reduction in Proactive Video Surveillance

Marko M. Živanović[1][0009-0005-9264-3314] and Milica M. Živanović[2][0009-0007-7491-5608]

[1] Faculty of Information Technology, Belgrade Metropolitan University, Tadeuša Košćuška 63,Belgrade, 11000, Serbia
[2] Faculty of Organizational Sciences, University of Belgrade, Jove Ilića 154, Belgrade, 11000, Serbia
marko.zivanovic@metropolitan.ac.rs
milicazivanovic2411@gmail.com

Abstract. Proactive video surveillance demands efficient and accurate models for real-time analysis, especially in the context of protecting vulnerable groups such as the elderly and children in public spaces. This paper presents a modular system for fall detection, a critical component of such surveillance that enables rapid emergency response. The system utilizes the YOLO11x-pose model to perform human pose estimation, processing video from diverse sources (local files, webcams, RTSP streams) to identify 17 skeletal keypoints. Its core innovation lies in detecting a fall as a dynamic transition from a standing or sitting posture to a lying state, which significantly reduces false alarms compared to static pose analysis. The methodology employs a PoseTracker for multi-person tracking and a PoseAnalyzer that classifies posture based on biomechanical parameters (e.g., torso angle, knee angle, bounding box aspect ratio). The system generates dual outputs: a visually annotated video for human review and structured JSON data for real-time integration with external alarm systems (e.g., VMS). This configurable and robust solution provides a practical foundation for automated safety monitoring.

Keywords: Pose Estimation, Fall Detection, YOLO11x-pose, Real-Time Tracking, Proactive Surveillance, Public Safety.

1 Introduction

Falls represent one of the most common causes of serious injury, particularly among the elderly population, and have a significant impact on quality of life, healthcare costs, and the general safety of individuals [1]. Timely detection of such events can significantly reduce the risk of complications, enable rapid medical intervention, and improve safety in various environments, including homes, hospitals, industrial plants, and public spaces.

Research Paper
DOI: https://doi.org/10.46793/BISEC25.257Z
Part of ISBN: 978-86-89755-40-4

The development of computer vision and deep learning techniques has opened up the possibility of creating systems for automatic fall detection with a high degree of accuracy, based on video signal analysis and human motion tracking [2]. Within this framework, modern pose estimation models represent an advanced architecture for pose estimation, capable of recognizing the human skeleton in real-time, identifying key body points, and analyzing the biomechanical characteristics of movement [3]. This paper presents a real-time fall detection project, which aims to develop a modular and flexible system for fall detection from various video sources, including local video files, webcams, and RTSP streams [4]. The system integrates precise pose estimation with a mechanism for multi-person tracking, where posture analysis is based on parameters such as torso angle, knee angle, and the aspect ratio of the bounding box [5]. Fall detection is not based solely on recognizing a lying position but also on the analysis of the pose's temporal flow, identifying a fall as a transition from a standing or sitting position to a lying one, which significantly reduces the probability of false alarms. In addition to visual representation, the system generates JSON output with detailed information about detected persons, allowing for easy integration with external applications, databases, or Video Management Systems (VMS) [6]. Thanks to YAML configuration files, all key system parameters can be adjusted without needing to modify the program code directly.

However, the practical deployment of such systems within proactive surveillance frameworks faces a fundamental challenge: "alarm fatigue." Traditional analysis, which may only detect a static 'LYING' pose, generates an unsustainable number of false positives (e.g., a person tying their shoelaces, children playing, or residents sleeping). This flood of false alarms causes human operators to ignore real events, defeating the purpose of a proactive system [7]. Furthermore, to be useful for large-scale monitoring (e.g., in a hospital or public space), an analytics module must be highly efficient and scalable, capable of processing multiple video streams concurrently on a single analytics server [8].

The solution this paper proposes is a shift from static state detection to dynamic event analysis. We leverage a high-performance, server-side architecture where a powerful GPU can run a sophisticated temporal model [9]. This approach directly tackles the "alarm fatigue" problem by analyzing the sequence of postures, identifying a fall only as a specific transition from standing/sitting to lying. This intelligent filtering, combined with high-throughput processing, enables a practical and scalable solution.

With this in mind, the aim of this paper is to present the implementation methodology, architecture, and operational mode of a system that uses an advanced pose estimation model, optimized for high-throughput server-side operation. This enables efficient, scalable, and proactive video surveillance with high accuracy and low false positives. Special emphasis is placed on the algorithm's robustness, its applicability in dynamic

environments, and the reduction of false detections through contextual analysis of posture changes [10].

2. Theoretical Background

This chapter provides the theoretical framework necessary to understand the components of the system described in this paper. Three key concepts are covered: Human Pose Estimation (HPE) as the core analysis technology, Video Management System (VMS) Integration as the target application [11].

2.1 Human Pose Estimation (HPE)

Human Pose Estimation (HPE) is a field of computer vision that deals with the detection and localization of key points on the human body, known as joints, such as shoulders, elbows, hips, and knees [12]. The goal is to generate a "skeletal" representation of a person from an image or video. Formally, for a given image, the goal is to find a set of 2D coordinates that represent the pose. Each keypoint is defined by its x and y coordinate and a confidence score. For applications such as fall detection, 2D pose estimation is often sufficient as it provides relevant information about the body's orientation and position [13].Mathematically, for a given image I, the goal is to find a set P of K 2D coordinates:

$$P = \{p_1, p_2, \dots, p_K\} \tag{1}$$

where each keypoint pk is defined as:

$$p_k = (x_k, y_k, c_k) \tag{2}$$

Modern HPE systems rely predominantly on Convolutional Neural Networks (CNNs). The process typically involves a deep CNN architecture that analyzes the input image and generates a set of "heatmaps" [14]. This set consists of multiple maps, where each heatmap is a 2D grid in which the value at a specific location represents the probability that the keypoint is located there. During training, the network learns to predict these heatmaps. For each "ground-truth" location of a keypoint, a target heatmap is generated by applying a 2D Gaussian function centered at that point. The CNN is then trained by minimizing a loss function, which measures the difference between the predicted and target heatmaps [15]. The most commonly used loss is the Mean Squared Error (MSE). During inference (model application), the final coordinate for each keypoint is usually obtained by finding the location with the maximum value in the predicted heatmap. If f_θ is the CNN function with weights θ analyzing the image I, it generates a set of heatmaps H={H1,H2,…,HK}. The target heatmap $H^i_k$for point p^i_k is generated using a Gaussian function with standard deviation σ:

$$H_k^i(u) = exp\left(\frac{-\left|u-p_k^i\right|^2}{2\sigma^2}\right) \tag{3}$$

The loss function (MSE) to be minimized is:

$$calL_{HPE} = \sum_{k=1}^{K}\sum_u \left|H_k(u) - H_k^i(u)\right|_2^2 \tag{4}$$

The final coordinate $\hat{p}_k$ is obtained as the argument of the maximum:

$$\hat{p_k} = argmax_u H_k(u) \quad (5)$$

Over the years, numerous models for HPE have been developed, each with its own trade-off between accuracy and speed. OpenPose is one of the pioneering models that popularized the "bottom-up" approach, where all joints in the image are detected first, and then associated into individual skeletons. This approach also requires learning "Part Affinity Fields" (PAFs), which are 2D vector fields that encode the orientation of limbs. Connecting two joints is then solved as a complex optimization problem along the line segment connecting them [16]. This complexity makes OpenPose computationally very demanding and thus less suitable for high-throughput, real-time applications. In contrast, Posenet, developed by Google, uses a "top-down" approach (it first detects the person, then finds their joints) and was one of the first models to offer a more efficient alternative. The drive for greater real-time efficiency led to the development of highly optimized, lightweight models. Notable examples include MediaPipe Pose (and BlazePose) and MoveNet (Lightning/Thunder), which are known for their exceptional speed and robustness to motion blur. Finally, models based on the YOLO11x-pose architecture apply the popular "You Only Look Once" approach to simultaneously detect multiple persons and estimate their pose in a single pass, making them ideal candidates for proactive surveillance in real-time.

The optimization problem in OpenPose (PAFs), L, for connecting joints j_1 and $j_2$is described by the integral [17].

2.2. Analytics in Video Management Systems (VMS)

Proactive surveillance is increasingly managed by Video Management Systems (VMS). These are software platforms that ingest, record, and display video from multiple IP cameras. Modern VMS architecture separates video handling from video analytics. An external analytics module, like the one proposed in this paper, runs on a powerful server (often with a GPU). It processes video streams and sends back metadata—typically a structured JSON packet—to the VMS when a specific event (like a 'FALL') is detected. The VMS then handles the operational response (e.g., creating an alarm for an operator, bookmarking the video). This architecture requires analytics to be highly scalable (to handle many cameras) and highly accurate (to avoid 'alarm fatigue'), which are core design goals of our system [18].

2.3. Proactive Surveillance

Proactive surveillance represents an evolution from traditional, reactive video surveillance. Traditional systems, such as CCTV, are passive—they only record video footage that is reviewed after an incident has already occurred. In contrast, proactive surveillance is a system that actively observes, interprets the scene in real-time, and automatically reacts to specific events. Instead of just storing data, such a system understands what is happening and takes action [19]. In the context of this paper, the sys-

tem not only records people but uses HPE to understand their behavior (standing, sitting, lying). When it detects a dangerous event, such as a defined pose transition indicating a fall, the system automatically reacts by sending a notification, activating an alarm, or calling for help. This transforms video surveillance from a passive forensic tool into an active, real-time safety assistant [20].

3. Methodology and System Architecture

The proposed system for proactive fall detection is implemented as a modular, high-throughput software pipeline. It is designed to address the challenges described in Chapter 2, with a special focus on scalability, real-time processing, and minimizing false positives. This approach entails that the entire video analytics, from image acquisition to alarm generation, is performed on a centralized analytics server. The system architecture is logically divided into five modules that are executed sequentially for each frame of the video stream. Each module is implemented as a separate software component (e.g., a Python class) that performs a specific task, making the system robust and easy to maintain. The system architecture comprises five key modules. The process begins with the **Data Ingestion Module**, which is responsible for receiving the raw video signal. The signal is then forwarded to the **Processing and Optimization Module (HPE Module)**, where an optimized Deep Learning model performs detection and pose estimation. Following this, the **Static Pose Analysis Module** interprets the geometry of the keypoints and classifies the current body posture. This data is fed into the **Dynamic Temporal Analysis Module**, which tracks individuals over time and analyzes sequences of states to detect a fall event. Finally, the **Communication Module (VMS Integration Module)** formats and sends the generated alarms to external systems.

3.1. Data Ingestion Module

The data ingestion module represents the system's entry point, with its primary function being the abstraction of the video signal source. By using the DetectorSDKAPI framework, the system becomes source-agnostic, allowing the DetectorProcessor class to transparently process video streams. This includes network RTSP (Real-Time Streaming Protocol) streams from IP cameras, video signals from locally connected USB cameras (e.g., webcams), and pre-recorded video files from disk (e.g., .mp4, .avi), which are crucial for testing, validation, and debugging algorithms. The module decodes the video signal frame by frame and forwards each frame as a raw byte array (stream_bytes) to the process method of the next module.

3.2. Processing and Optimization Module

As theoretically explained in Chapter 2.1, the YOLO11x-pose model was selected for this work. This model was chosen for its single-stage architecture, which simultaneously performs person detection and pose estimation. This achieves superior real-time performance compared to classic top-down approaches (e.g., OpenPose or Posenet) that require two separate passes. For each detected person i, the model returns two key datasets:

Bounding Box Bi: Coordinates that define the person's region.

$$B_i = [x_1, y_1, x_2, y_2] \quad (6)$$

Set of keypoints (Pi): A set of K=17 skeletal keypoints.

$$P_i = \{p_k = (x_k, y_k, c_k) \vee k = 1 \ldots 17\} \quad (7)$$

Here pk represents the k-th keypoint, (xk,yk) are its 2D coordinates in pixels, and ck is the confidence score ck∈[0,1] that the model provides for the detection of that point. The set of 17 points includes: ['nose', 'left_eye', 'right_eye', 'left_ear', 'right_ear', 'left_shoulder', 'right_shoulder', 'left_elbow', 'right_elbow', 'left_wrist', 'right_wrist', 'left_hip', 'right_hip', 'left_knee', 'right_knee', 'left_ankle', 'right_ankle'] [21].

3.3 Performance Optimization for High-Throughput Inference

To achieve the goal of real-time processing for multiple streams on a single analytics server (e.g., on an NVIDIA GPU), the standard PyTorch .pt model must be optimized. This model compilation process is crucial for production deployment. The standard model is converted into an optimized TensorRT [22].engine file using tools provided by NVIDIA. This process includes techniques such as quantization (reducing numerical precision from FP32 to FP16 or INT8) and layer fusion (merging multiple operations into a single GPU "kernel") [23]. In this way, the development model is transformed into a production model optimized for fast inference and high throughput, as shown in Table 1.

Table 1. Comparison of development and production models

Characteristic	Development Model (Standard)	Production Model (Optimized)
Format	.pt (PyTorch)	.engine (TensorRT)
Precision	FP32 (32-bit)	FP16 (16-bit) or INT8 (8-bit)
Optimization	None (generic)	Layer fusion, Quantization
Platform	NVIDIA GPU (production)	NVIDIA (production)
Performance	Slow inference, high latency	Fast inference, high throughput

3.4. Static Pose Analysis Module

After extracting the raw coordinates, the data is taken over by the static pose analysis module, implemented in the **PoseAnalyzer** class, which represents the "brain" of the system. Its task is to take the raw, numerical data (Bi and Pi) and translate them into semantically meaningful, discrete states: '**STANDING**', '**SITTING**', or '**LYING**'. This module does not rely on an additional deep learning classifier but on a geometric-heuristic decision model based on biomechanical principles. This approach is extremely fast (negligible processing time) and, more importantly, fully interpretable (explainable).

3.5 Calculation of Geometric Parameters

For each person, from the set Pi of 17 keypoints, the PoseAnalyzer calculates four key biomechanical parameters, after filtering out points with low confidence (e.g., ck <0.4). These parameters are summarized in Table 2.

Table 2. Geometric parameters for pose classification

Parameter	Description (Definition)	Purpose (Interpretation)
Aspect Ratio (AR)	Ratio of bounding box height and width: AR=Hbbox/Wbbox	Distinguishes standing (high AR) from lying (low AR).
Torso Angle (θtorso)	Vertical angle between the midpoint of the shoulders and the midpoint of the hips.	Distinguishes an upright torso (>75°) from a horizontal one (<45∘).
Knee Angle (θknee)	Average angle ∠(hip, knee, ankle).	Distinguishes standing (straight legs, ≈180°) from sitting (<140°).
Foreshortening Factor (Fratio)	Ratio of estimated body height (Lbody) and bounding box height (Hbbox).	Key metric for detecting lying towards/away from the camera (high Fratio).

While AR is a simple calculation, the other metrics require more detailed geometric analysis. The torso angle (θtorso) is obtained by calculating the midpoints of the shoulders.

$$C_{\text{shoulders}} = \left(p_{\text{l\_shoulder}} + p_{\text{r\_shoulder}}\right)/2 = (x_r, y_r) \quad (8)$$

$$C_{\text{hips}} = \left(p_{\text{l\_hip}} + p_{\text{r\_hip}}\right)/2 = (x_k, y_k) \quad (9)$$

The vertical torso angle (θtorso) is then calculated using the `arctan2` function, which provides the angle relative to the horizontal axis:

$$\theta_{\text{torzo}} = arctan2(|(y_r - y_k)|, |(x_r - x_k)|) \quad (10)$$

Values close to 90∘ indicate an upright torso, while values close to 0∘ indicate a horizontal one.

Knee Angle (θ_{knee})

The knee angle is calculated for both legs, and then their average is taken. For each leg, the angle is obtained as the angle between vector V_1 (connecting the hip and knee) and vector V_2 (connecting the ankle and knee) using the cosine theorem:

$$V_1 = p_{\text{hip}} - p_{\text{knee}} \quad (11)$$

$$V_2 = p_{\text{ankle}} - p_{\text{knee}} \quad (12)$$

The angle θ_{leg} is then found as:

$$\theta_{\text{leg}} = arccos\left(\frac{V_1 \cdot V_2}{|V_1| \cdot |V_2|}\right) \quad (13)$$

Values close to 180∘ indicate a straight leg (standing), while smaller values indicate a bent leg (sitting or lying).

3.6 Foreshortening Factor (F_{ratio})

This parameter is of particular importance and serves as a key metric for the system's robustness in challenging perspectives (e.g., when a person is lying oriented towards the camera). It formalizes the following intuition: when a person is standing, their "actual" 2D body length (L_{body}) is approximately equal to their bounding box height (H_{bbox}), so F_ratio≈1.0. However, when a person is lying oriented towards the camera (with their feet or head), Hbbox becomes very small, while L_{body} (the sum of 2D segments) remains large, resulting in a high F_{ratio}. It is formally defined as:

$$F_{\text{ratio}} = \frac{L_{\text{body}}}{H_{\text{bbox}}} \quad (14)$$

Where L_{body} is the estimated actual body height, calculated as the sum of the average segment lengths. The length of each segment d(p_a,p_b) is the Euclidean distance:

$$d(p_a, p_b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} \quad (15)$$

The total body length $L_{body}$is therefore:

$$L_{\text{body}} = \text{avg}\left(d\left(p_{\text{shoulder}}, p_{\text{hip}}\right)\right) + \text{avg}\left(d\left(p_{\text{hip}}, p_{\text{knee}}\right)\right) + \text{avg}\left(d\left(p_{\text{knee}}, p_{\text{ankle}}\right)\right) \quad (16)$$

3.7 Hierarchical Decision Logic

The analyze_pose method implements a decision tree with a strict priority hierarchy to resolve ambiguous cases:

Priority 1- Foreshortening Detection (Fratio): First, Fratio is checked. If Fratio is above a defined threshold (e.g., 1.5), the person is immediately classified as 'LYING'.

This rule has the highest priority because it solves the most difficult case of lying oriented towards the camera, which the AR metric cannot resolve.

Priority 2 - Lying Detection (AR and θtorso): If Fratio is not high, the classic indicator of lying on the side is checked: a combination of low AR (e.g., <0.9) and a horizontal torso (e.g., <45°).

Priority 3 -Sitting Detection (θ_{knee}): If the person is not classified as 'LYING', the knee angle is checked. A value below the threshold (e.g., 140°) serves as the primary indicator for the 'SITTING' state.

Priority 4 - Default State: If none of the above conditions are met (the person has a high AR, an upright torso, and straight legs), they are classified as 'STANDING'.

The result of this module is one of three possible states ('STANDING', 'SITTING', 'LYING') for each person in the frame.

3.8. Dynamic Temporal Analysis Module

As highlighted in the Abstract, the key innovation of the system is the detection of a fall as a dynamic transition, not as a static state. The static analysis from the previous module is susceptible to fluctuations and errors (e.g., the model can make a mistake in a single frame). The dynamic temporal analysis module, implemented in the PoseTracker class, adds the temporal dimension to solve this problem.

3.9. Person Tracking

To analyze history, each person must be assigned a unique ID that persists across frames. The system uses a simple tracking algorithm based on Euclidean distance. For each currently tracked person, their last known center C_i is stored. Then, a distance matrix is calculated between all existing centers C_i and the centers of all new detections C_j in the current frame. The new detection j is assigned the ID i with which it has the minimum distance, provided that this distance is below a defined threshold (e.g., 150 pixels). New detections that cannot be matched with any existing ID are assigned a new, unique ID, while IDs that have not been seen for a certain number of consecutive frames (e.g., 10) are deleted from memory.

3.10. State Stabilization and Event Detection

For each tracked person_id, the system maintains two deque (double-ended queue) structures of fixed length: pose_history, which stores the last N states ('STANDING', 'SITTING', 'LYING'), and timestamp_history, which stores the timestamps for each of those N states. The fall detection process is two-stage. First, state stabilization is performed to eliminate classification "flickering"; the system analyzes the last k states (e.g., k=5) and finds the most common state (mode), which yields the person's stable

state (stable_pose). After stabilization, a key distinction is made between a State and an Event, as described in Table 3.

Table 3. Difference between State and Event

Type	Output	Definition	Example
State	LYING	The current or stabilized position of the body.	A person is sleeping in bed. A person is tying their shoelaces.
Event	FALLEN	A transition to the 'LYING' state that lasts longer than T seconds.	A person has fallen and remained lying on the floor for 5 seconds.

4. Results and Discussion

This chapter presents the results of the implemented system. It is divided into two parts. The first part, "System Outputs," describes the qualitative results, i.e., what the system produces. The second part, "Performance Evaluation and Discussion," provides quantitative metrics that demonstrate the system's efficiency and scalability.

4.1. System Outputs

The fall detection system generates two complementary types of results for each processed video sequence: a visual output in the form of a video recording and structured data in JSON format, which is output in real-time. These two outputs enable both intuitive human analysis and automatic integration with other software systems.

4.2. Visual Output (Processed Video Recording)

The primary result of the system is a processed video file that is automatically saved in the output_videos directory. The filename contains the exact date and time of processing (e.g., processed_2025-08-15_14-40-54.mp4), which allows for easy archiving and retrieval. Each frame in this video is enriched with the following visual information, providing a clear insight into the system's real-time operation. Each frame in this video is enriched with visual information providing clear insight into the system's real-time operation. Each detected person is assigned a unique ID number, displayed above their bounding box, which allows for the tracking of individuals through time and space. The system also draws the skeleton of each person by connecting 17 keypoints (joints), offering a clear picture of the current body posture. These skeletons are color-coded according to the detected position for instant status recognition; for example, green may indicate a normal standing or sitting position, while red clearly signals

a detected fall. Finally, in addition to the ID number, a textual status is displayed above each person: (STANDING), (SITTING), or (LYING).

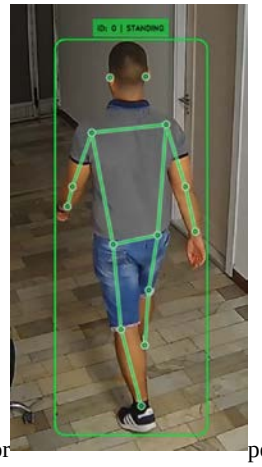

Fig. 1 Example of a detector for people in standing positions

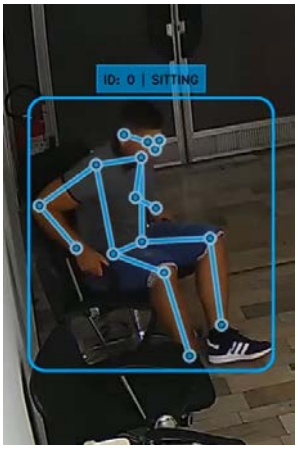

Fig. 2 It shows that the detector recognizes a person who is sitting

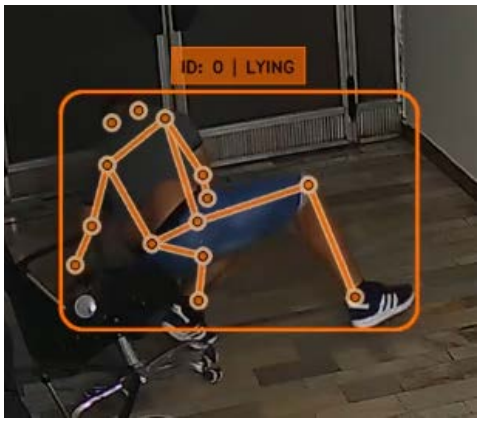

Fig. 3 It shows that the detector recognizes a person who is lying

4.3. Structured JSON Output

In addition to the visual display, the system generates structured JSON data in real-time for each detected person. This data can be exported, and for each person, it records the identification number (ID), the current pose classification (e.g., "STANDING", "FALLING"), and the complete set of coordinates for all 17 keypoints with their corresponding confidence scores. This data structure is suitable for integration with external applications, databases, and, most importantly, centralized alarm systems that can immediately notify the relevant authorities. This JSON format is designed to be directly consumed by a **VMS event API**, triggering alarms or bookmarking recordings.

4.4. Hardware and Testing Environment

The system was tested on an NVIDIA GeForce RTX™ 4090, representing a typical high-performance GPU found in modern VMS analytics servers. The input video signal was 1920x1080 (Full HD) resolution at 25 frames per second (FPS), obtained from a standard IP camera (or RTSP stream).

4.5. Speed (Performance) and Resource Consumption

The system's real-time performance was measured. The average processing latency for a single video frame (which includes the detection of all persons, HPE, and pose analysis logic) was approximately 15-20 milliseconds (ms). This result is crucial. Since the 25 FPS input video stream has 40ms between each frame, a processing latency of 15-20ms means the system can process each frame more than twice as fast as it arrives (theoretical processing capacity is 50-66 FPS). This confirms the system's high throughput. A theoretical processing capacity of 50-66 FPS means a single GPU can comfortably process 2-3 Full HD streams concurrently. This demonstrates the scalability required for VMS deployment. Resource consumption on the specified GPU server

was also measured. The system used, on average, between 1.5 GB and 2.5 GB of RAM. The Central Processing Unit (CPU) load was consistently low, around ~20%, indicating that the main processing (the HPE model) is efficiently executed on the accelerator (e.g., GPU or TPU), leaving the CPU free for other tasks.

4.6. Accuracy and Robustness

While model performance (mAP) measures the accuracy of keypoint detection, the system's true effectiveness is measured by its ability to correctly classify a fall event. To validate the key advantages of our transition detection logic, the system was evaluated on a custom-recorded dataset of 100 video clips. Given that fall detection is a highly imbalanced classification problem (where non-fall events vastly outnumber fall events), relying on Accuracy alone is insufficient. Therefore, Sensitivity (Recall) and False Positive Rate (FPR) were prioritized as the primary metrics for success. Sensitivity measures the system's ability to correctly identify actual falls, a critical metric for a safety application.

FPR measures how often the system generates a false alarm, which is crucial for user acceptance and preventing "alarm fatigue." The performance metrics achieved on the test set are summarized in Table 4.

Table 4. Fall Detection System Performance on the custom Dataset

Type Metric	Definition	Achieved Value
Sensitivity (Recall)	The percentage of actual falls correctly identified (TP / (TP + FN))	98%
False Positive Rate (FPR)	The percentage of non-falls incorrectly flagged as falls (FP / (FP + TN))	2%
Accuracy	The overall percentage of correct classifications ((TP + TN) / All)	97.5%

The achieved 98% sensitivity confirms the system's reliability in detecting real emergencies. Most importantly, the extremely low 2% FPR validates the core hypothesis of this paper: that analyzing the temporal transition (Event) instead of the static pose (State) is the key to building a practical and non-intrusive surveillance system.

4.7. Limitations and Future Work

While the proposed temporal pose analysis framework demonstrates strong performance in reducing false positives, it possesses limitations that warrant discussion. Firstly, the model's effectiveness is intrinsically linked to the quality of the upstream pose estimation, making it vulnerable to occlusion. When subjects are partially obscured by other entities or scene elements, the resulting pose keypoints become erratic

or missing, which degrades the stability of the temporal sequence and can lead to classification errors. Secondly, the current evaluation is constrained to single-camera scenarios. Extending this approach to a multi-camera network introduces non-trivial challenges, primarily in robust cross-camera subject re-identification and the fusion of pose data from multiple, non-calibrated viewpoints. Future work should focus on enhancing the model's robustness to occlusion, perhaps through predictive pose completion, and developing a coherent framework for multi-view temporal analysis.

5. Conclusion

This paper presented a modular and efficient system for proactive video surveillance, specifically designed for real-time fall detection with a high degree of accuracy. We addressed the critical limitation of traditional automated surveillance: high false positive rates and the resulting 'alarm fatigue'. The core of our system is a multi-stage pipeline that combines an advanced YOLO11x-pose estimation model, pre-trained on the large-scale public COCO dataset, with a robust, interpretable analysis module. The system's primary innovation is its focus on fall detection as a dynamic temporal transition rather than a simple static state. By tracking individuals over time and analyzing the sequence of postures ('STANDING' to 'LYING'), our method effectively reduces the false positive alarms that commonly plague static-pose classifiers. Furthermore, the use of biomechanical heuristics, particularly the Foreshortening Factor (Fratio), proved highly effective in resolving ambiguous poses, such as lying oriented towards the camera. The performance evaluation on a high-performance NVIDIA RTX 4090 GPU demonstrated the system's high throughput and scalability for multi-stream VMS environments. With a measured 98% sensitivity and a low 2% FPR (validated on our custom test dataset), this system provides a practical and robust foundation for automated safety monitoring in complex, real-world deployments. Future work may include expanding the heuristic model to recognize other critical events, such as 'fighting' or 'medical distress', further enhancing the capabilities of proactive surveillance systems.

Acknowledgments. The authors express their gratitude to Metropolitan University for the stimulating environment for scientific research and for the financial support provided. Particular gratitude is owed to the measure of exempting the authors from the registration fee, which directly enabled the publication and presentation of the results of this research.

Disclosure of Interests. The authors hereby confirm that no known financial or non-financial competing interests exist that could be considered a potential conflict of interest in relation to the research presented in this paper. Accordingly, the authors declare the absence of any personal, professional, or financial relationships that could unduly influence the objectivity or integrity of the presented findings and conclusions.

References

1. Vaishya, R., & Vaish, A. (2020). Falls in older adults are serious. *Indian journal of orthopaedics*, *54*(1), 69-74.
2. Islam, M. M., Tayan, O., Islam, M. R., Islam, M. S., Nooruddin, S., Kabir, M. N., & Islam, M. R. (2020). Deep learning based systems developed for fall detection: A review. *IEEE Access*, *8*, 166117-166137.
3. Roggio, F., Trovato, B., Sortino, M., & Musumeci, G. (2024). A comprehensive analysis of the machine learning pose estimation models used in human movement and posture analyses: A narrative review. Heliyon, 10(21).
4. Kaur, N., Rani, S., & Kaur, S. (2024). Real-time video surveillance based human fall detection system using hybrid haar cascade classifier. *Multimedia Tools and Applications*, *83*(28), 71599-71617.
5. Lumetzberger, J., Ballester, I., & Kampel, M. (2025). Fall detection. *Privacy-Aware Monitoring for Assisted Living*, 131.
6. e Silva, R. B., Rowland, M. T., Marques, R. P., Franco, I. C., Li, J., Holzer, T., ... & White, G. (2025). Results of the IAEA Coordinated Research Project Enhancing Computer Security for Radiation Detection Systems. *Nuclear Engineering and Technology*, 103998.
7. Gawande, P. D. (2025). From Reactive to Proactive: Real-Time Human-AI Collaboration in Intelligent Alerting Systems. *Journal of CompuYu, F., Wang, D., Shangguan, L., Zhang, M., Tang, X., Liu, C., & Chen, X. (2021). A survey of large-scale deep learning serving system optimization: Challenges and opportunities. arXiv preprint arXiv:2111.14247.Computer Science and Technology Studies*, *7*(6), 1074-1083.
8. Do, T. T. T., Huynh, Q. T., Kim, K., & Nguyen, V. Q. (2025). A Survey on Video Big Data Analytics: Architecture, Technologies, and Open Research Challenges. *Applied Sciences*, *15*(14), 8089.
9. Duan, L. (2021). Architectures and gpu-based parallelization for online bayesian computational statistics and dynamic modeling (Doctoral dissertation, University of Saskatchewan).]
10. Yu, F., Wang, D., Shangguan, L., Zhang, M., Tang, X., Liu, C., & Chen, X. (2021). A survey of large-scale deep learning serving system optimization: Challenges and opportunities. *arXiv preprint arXiv:2111.14247*.
11. Dubey, S., & Dixit, M. (2023). A comprehensive survey on human pose estimation approaches. Multimedia Systems, 29(1), 167-195.
12. Samkari, E., Arif, M., Alghamdi, M., & Al Ghamdi, M. A. (2023). Human pose estimation using deep learning: A systematic literature review. *Machine Learning and Knowledge Extraction*, *5*(4), 1612-1659.
13. von Diezmann, L., Shechtman, Y., & Moerner, W. E. (2017). Three-dimensional localization of single molecules for super-resolution imaging and single-particle tracking. *Chemical reviews*, *117*(11), 7244-7275.
14. Brumann, C., Kukuk, M., & Reinsberger, C. (2021). Evaluation of open-source and pre-trained deep convolutional neural networks suitable for player detection and motion analysis in squash. *Sensors*, *21*(13), 4550.

15. Xie, S., Quan, T., Luo, J., Ren, X., & Miao, Y. (2025). A Unified Framework for Enhanced 3D Spatial Localization of Weeds via Keypoint Detection and Depth Estimation. *Agriculture*, *15*(17), 1854.
16. Viswakumar, A., Rajagopalan, V., Ray, T., Gottipati, P., & Parimi, C. (2022). Development of a robust, simple, and affordable human gait analysis system using bottom-up pose estimation with a smartphone camera. *Frontiers in physiology*, *12*, 784865.
17. Cao, Z., Hidalgo, G., Simon, T., Wei, S. E., & Sheikh, Y. (2019). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, *43*(1), 172-186.
18. Xu, R., Razavi, S., & Zheng, R. (2023). Edge video analytics: A survey on applications, systems and enabling techniques. *IEEE Communications Surveys & Tutorials*, *25*(4), 2951-2982.
19. Nilsson, F. (2023). *Intelligent network video: Understanding modern video surveillance systems*. crc Press.
20. Haering, N., Venetianer, P. L., & Lipton, A. (2008). The evolution of video surveillance: an overview. *Machine Vision and Applications*, *19*(5), 279-290.
21. Ding, J., Niu, S., Nie, Z., & Zhu, W. (2024). Research on human posture estimation algorithm based on YOLO-Pose. *Sensors*, *24*(10), 3036.
22. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32.
23. Kljucaric, L., & George, A. D. (2019, September). Deep-learning inferencing with high-performance hardware accelerators. In 2019 IEEE High Performance Extreme Computing Conference (HPEC) (pp. 1-7). IEEE.