Person Tracking Using AI and Surveillance Cameras

By

Muhammad Abdullah
Enrollment No. 01-133222-041

Ammad Ahmed Satti
Enrollment No. 01-133222-008

Supervised By

Dr Imran Fareed Nizami

Session 2022-26

A Report is submitted to the Department of Electrical Engineering,
Bahria University, Islamabad.
In partial fulfillment of requirement for the degree of BS(EE).

Certificate

We accept the work contained in this report as a confirmation to the required standard for the partial fulfillment of the degree of BS(EE).

_____.
Head of Department

_____
Supervisor

_____.
Internal Examiner

_____
External Examiner

Dedication

We dedicate this work to our parents, whose love, prayers, and sacrifices made this journey possible. To our supervisor, Dr. Imran Fareed Nizami, for his guidance and support. And to all those who believe in the power of technology to change the world.

Acknowledgments

All praise and gratitude are due to Almighty Allah, the Most Gracious and the Most Merciful, whose blessings guided us through every step of this journey. We would like to express our deepest gratitude to our supervisor, Dr. Imran Fareed Nizami, for his exceptional guidance, continuous encouragement, and invaluable technical insight throughout this project. His expertise in the field of artificial intelligence and computer vision proved instrumental in shaping the direction and quality of this work. We are truly grateful for his patience, availability, and the time he devoted to reviewing our progress. We extend our sincere thanks to the faculty and staff of the Department of Electrical Engineering, Bahria University, Islamabad, for providing us with an enriching academic environment and access to the resources needed to complete this final year project. Our heartfelt appreciation goes to our parents and families, whose unwavering support, patience, and prayers gave us the strength and motivation to persevere. Their belief in us has been a constant source of inspiration. We also acknowledge the open-source communities behind YOLOv10, DeepSORT, ResNet, PyTorch, and PySide6, whose publicly available tools and frameworks formed the technical backbone of this system. Finally, we thank our fellow students and peers at Bahria University for their camaraderie and the many productive discussions that enriched this experience. Muhammad Abdullah Ammad Ahmed Satti

Abstract

The rapid development of urban infrastructure and the growing demand for stricter safety measures have made video surveillance an essential part of modern safety management. But a basic problem with traditional surveillance systems is the need for human operators to watch, assess and follow people in large camera networks. This project “Person Tracking Using AI and Surveillance Cameras” is aimed at solving these major in-efficiencies by building a complete automated system for real-time person tracking in multiple non-overlapping camera views. The main purpose of this work is to keep “Identity Continuity” i.e. to identify that a person entered in the field of view of one camera is the same person who entered in the field of view of another camera moments ago.

The core of the proposed approach is a complex AI pipeline combining deep learning based Re-Identification (Re-ID) with state-of-the-art object identification. For the detection part, we used the YOLOv10 (You Only Look Once version 10) architecture. Unlike previous versions YOLOv5 and YOLOv8 which have computational overhead caused by Non-Maximum Suppression (NMS), YOLOv10 uses NMS-free training. Therefore, our system can achieve high speed real-time person detection with low latency even on resource-limited hardware. This is to ensure that the system can handle multiple HD streams at the same time, without frame drops, at high security environments.

The most innovative feature of this project is the Re-ID Manager which uses a ResNet18 backbone to overcome the “Tracking Gap” in tra-ditional surveillance. When a person is detected, the system extracts a 128-dimensional feature embedding (a digital signature) based on visual characteristics such as clothing patterns, color, height, and body shape. These embeddings are stored and compared with Cosine Similarity measures. This takes away the privacy-intrusive nature of facial recognition and allows for cross-camera tracking. The technology generates a contin-uous record of movement across various physical areas by automatically assigning the previous ID to the person when the similarity score between

a new detection and a saved embedding is above a calibrated threshold.

We developed a full-scale Multi-Camera Dashboard with the PySide6 framework that allows the security staff to use this technology. The interface is multi-threaded, so that the heavy-duty AI processing is done in the background and the user interface stays responsive. The interface displays up to four video feeds simultaneously and tracks individuals with persistent ID boxes. We also added a database management system, SQLite3, to log all tracking events. This data base stores the unique PersonID, CameraID and timestamps to allow a searchable historical record, which may be used for forensic research or post-event inquiry.

A variety of indoor settings, such as hallway transitions and different lighting conditions, were used to test the implementation. Compared to conventional tracking-by-detection techniques, preliminary results show a high level of identity retention stability and a notable decrease in ID-switching mistakes. The project successfully connects the dots between unprocessed video footage and intelligent, useful information. Future research will concentrate on optimizing the system for densely populated areas and incorporating 2D path reconstruction, which entails mapping these digital IDs onto a top-down floor plan. This technology improves the overall accuracy and dependability of monitoring equipment in establishments like shopping centers, hospitals, and educational institutions while greatly reducing the workload for security teams by automating the tracking process.

Contents

Introduction 1

1.1 Project Background / Overview 2
1.2 Problem Description 3
1.3 Project Objectives 4
1.4 Project Scope 5

Literature Review 6

2.1 Classical Feature-Based Approaches to Person Detection 7
2.2 Deep Learning-Based Object Detection Methods 8
2.3 Single and Multi-Object Tracking Algorithms 9
2.4 Person Re-Identification (Re-ID) 11
2.5 Research Gaps and Motivation for the Proposed System 12

System Analysis and Design 14

3.1 Existing System 15
3.2 Proposed System 17
3.3 Requirement Specifications 20
3.4 Use Cases 23

System Design 26

4.1 System Architecture 27
4.2 Design Constraints 29
4.3 Design Methodology 32
4.4 High Level Design 33
4.4.1 Conceptual / Logical View 33
4.4.2 Process View 35
4.4.3 Physical View 36
4.4.4 Module View 37
4.4.5 Security View 38
4.5 Low Level Design 39
4.5.1 Module Breakdown 40
4.5.2 Video Ingestion Module 40
4.5.3 Detection Module 41
4.5.4 Tracking Module (DeepSORT) 42
4.5.5 ReIDManager Module 43
4.5.6 Dashboard Database Module 44
4.5.7 Inter-Module Data Flow 45
4.5.8 Key Design Decisions at the Class Level 45

4.6 Database Design 46
4.6.1 Overview 46
4.6.2 Database Tables 47
4.6.3 Non-Database Files 48
4.6.4 How the Tables Relate 48
4.6.5 Data Dictionary 49
4.6.6 Database Schema 51
4.6.7 How the Database is Used in the Pipeline 54
4.6.8 Key Design Notes 55
4.6.9 Data Dictionary Legend 55
4.7 GUI Design 56
4.7.1 Main Window Layout 57
4.7.2 Left Panel — Camera Management and AI Configuration 57
4.7.3 Centre Panel — Live Monitoring Feed 58
4.7.4 Right Panel 59
4.7.5 Person Registration Dialog 60
4.7.6 Database Viewer Dialog 61
4.7.7 History Viewer Dialog 62
4.7.8 Status Bar and Real-Time Feedback 63
4.7.9 Summary 63
4.8 External Interfaces 64
4.8.1 Camera Hardware Interface 64
4.8.2 YOLO Object Detection Model Interface (Ultralytics) 65
4.8.3 ResNet-18 Feature Extraction Model Interface (TorchVision) 66
4.8.4 SQLite Database Interface 67
4.8.5 File System Interface (Reference Image Storage) 68
4.8.6 GPU / CUDA Runtime Interface 69
4.8.7 PySide6 / Qt GUI Framework Interface 69

System Implementation 69

5.1 System Architecture 69
5.1.1 Internal Components 70
5.1.2 Functionality of the Components 71
5.1.3 Communication Between the Components 72
5.2 Tools and Technology Used 75
5.2.1 Programming Language 75
5.2.2 AI and Machine Learning Frameworks 76
5.2.3 Computer Vision 79
5.2.4 Graphical User Interface 80

5.2.6 Database....82
5.2.7 Summary Table....82
5.3 Development Environment / Languages Used....83
5.3.1 Programming Language....79
5.3.2 Development Operating System....80
5.3.3 Integrated Development Environment (IDE)....80
5.3.4 Hardware Environment....81
5.3.5 Python Environment Management....82
5.3.6 Project Structure....83
5.3.7 Summary....84
5.4 Processing Logic / Algorithms....84
5.4.1 Video Ingestion Algorithm....84
5.4.2 Person Detection Algorithm (YOLOv10)....85
5.4.3 Feature Extraction Algorithm (ResNet18)....86
5.4.4 Re-Identification Matching Algorithm....87
5.4.5 Detection Logging and Output....88
5.4.6 Algorithm Summary....88
5.5 Application Access Security....87
5.5.1 Application Access and Authentication....87
5.5.2 Authorization and Operator Roles....88
5.5.3 Network Security and Camera Feeds....88
5.5.4 Data Storage Security....89
5.5.5 Audit Logging and Detection History....89
5.5.6 Safe Handling of Biometric Data....90
5.5.7 Summary....90
5.6 Database Security....90
5.6.1 Remote Access....91
5.6.2 Database Authentication....91
5.6.3 Authorization and Access Rights....92
5.6.4 Anonymous and Group Users....92
5.6.5 Auditing and Logging....93
5.6.6 Summary....95

Testing and Evaluation....95

6.0 Evaluation Metrics....96
6.1 Chapter Overview....97
6.2 Graphical User Interface Testing....98
6.3 Usability Testing....98

6.4 Software Performance Testing 99
6.5 Compatibility Testing 101
6.6 Exception Handling Testing 102
6.7 Load Testing 103

6.8 Security Testing 104
6.9 Installation Testing 105
6.10 Comparison with Existing Techniques 105
6.11 Strengths and Limitations 106
6.12 Evaluation Summary 107

Conclusion 109

References 113

Appendix A — User Manual 119

Chapter 1

Introduction

Project Background/Overview:

In today's constantly changing urban environment security management has become a top priority for organizations, companies and public infrastructure. Closed-circuit television (CCTV) systems play a vital role in the surveillance of daily activities and public safety in places like colleges, shopping malls, hospitals, airports, corporate offices, etc. But the conven-tional approach to video surveillance is not very active. These traditional configurations essentially depend on human operators to continuously monitor a number of displays, detect suspicious activity and manually track subjects of interest. They function primarily as recorders. This heavy reliance on human observation creates major operational bottlenecks. Manual monitoring is slow by definition, mentally taxing and prone to human error. The more cameras you put in , the less effective the security staff are at monitoring the situation . This results in slow reaction times and a high chance of missing something . Massive paradigm shift in the surveillance business is being driven by advances in computer vision and artificial intelligence (AI) to bypass these restrictions. Surveillance equipment is evolving from passive video storage to active intelligent real-time monitoring. Modern AI systems can automatically detect humans, track humans in large physical spaces at speeds and with accuracies that humans cannot match, and analyze complex movement patterns.

This work, "Person Tracking Using AI and Surveillance Cameras", proposes an intelligent and automated surveillance system based on recent technical advancements. The main novelty of this system is that it can easily track and identify a subject over multiple non-overlapping camera feeds while strictly maintaining "identity continuity". In practice that means the system can tell automatically that a person is the same person when they go out of sight of one camera and appear in another. This project delivers a neat and handy security solution featuring state-of-the-art deep learning models, solid database storage and a live multi-camera monitoring dashboard. Ultimately, it greatly reduces the operational burden on security personnel, making surveillance smarter, faster and more reliable

for today's complex environments.

- **Problem Description**

The management of security in modern institutions such as airports, shopping centers and universities is greatly impacted with Closed-Circuit Television (CCTV) networks. But most traditional surveillance is passive. Its main function is recording, which implies that human operators have to constantly watch several screens. This manual process is slow, mentally taxing and very vulnerable to human error, often resulting in missed security events and delayed response. To circumvent such restrictions, the security sector is turning to active, AI-enabled surveillance. Modern computer vision architectures are orders of magnitude faster and more accurate than humans at tracking movement patterns and detecting human presence on their own. By automating the detection and tracking process, the integration of AI closes the loop between raw video capture and action-able security intelligence, effectively removing the inefficiencies of manual oversight.

The project "Person Tracking Using AI and Surveillance Cameras" suggests a smart automatic surveillance system that utilizes recent technical advancements. The key innovation is the ability to "maintain identity continuity" — to identify and track a person as they move between several non-overlapping video feeds. This research offers a practical solution that significantly reduces operational stress on security personnel and enhances overall facility security by integrating state-of-the-art deep learning models with permanent database storage and a live multi-camera dashboard.

- **Project Objectives**

The main target of this project is to design an intelligent and automatic surveillance system to overcome the serious shortcomings of manual observation. To achieve this, the project has the following specific objectives:

1. **Real Time Detection:** A high speed real time human detection engine will be developed using the advanced YOLOv10 architecture, pro-viding accurate detection with no processing latency.

2. **Identity Continuity:** With a Re-Identification (Re-ID) model built on ResNet18 that links people with visual feature embeddings, persistent identity tracking can be achieved across different non-overlapping camera feeds.

3. **Error reduction:** can significantly reduce tracking losses and the "ID switching" phenomenon common in congested or complex indoor environments.

4. **Data Logging and Storage:** To provide an automated database backend for permanently storing tracking records, such as unique Person IDs, exact timestamps, and specific camera locations, for forensic analysis after the event.

5. **Centralized visualization:** Create a simple, multi-threaded live monitoring dashboard using PySide6. Security personnel can simultaneously watch several tracking streams without interrupting the system.

6. **Operational Efficiency:** Active tracking of videos instead of passive recording will significantly reduce the cognitive and manual load of security personnel, thus increasing the overall surveillance efficiency.

· Project Scope

The goal of this project is to create software that enhances current security cameras using artificial intelligence. The following defines the system's precise boundaries:

1.4.1 In-Scope

AI models: ResNet18 for person Re-Identification and YOLOv10 for real-time human identification.

Multi-Camera Tracking: The smooth transfer of a person's tracking ID when they switch between non-overlapping camera feeds.

User Interface: Using PySide6, create a multi-threaded, responsive live monitoring dashboard.

Data logging: Recording movement history (Person ID, camera location, and timestamp) using a SQLite3 database.

Target Environment: Indoor and semi-enclosed spaces like shopping centers, offices, and institutions.

1.4.2 Out-of-Scope

Facial Recognition: The technology uses generic visual characteristics to track. It doesn't connect to real-world identification databases or employ facial biometrics.

Hardware Setup: No additional CCTV cameras will be manufac-tured, wired, or physically installed as part of this project.

Outdoor Environments: The AI is not designed to withstand severe weather, torrential rain, or total darkness.

Audio Surveillance: The system does not record or analyze audio; it solely processes video data.

Chapter 2

Literature Review

Automated person tracking across multi-camera surveillance networks is a task lying at the intersection of computer vision, deep learning and real-time processing. This chapter reviews the related work in person detection, multi-object tracking and person re-identification. Here, we discuss the strengths and weaknesses of existing approaches to motivate the need for the lightweight, integrated architecture proposed in this project.

2.1 Classical Feature-Based Approaches to Person Detection: Before deep learning came along, person detection relied on hand-engineered feature descriptors, mathematical representations of what a human figure looks like in an image that were carefully designed by humans. These methods formed the basis for early surveillance systems and are useful reference points for understanding why modern approaches were required. The most influential work of this period was that of Dalal and Triggs,[1] who proposed the Histogram of Oriented Gradients (HOG) descriptor with a Support Vector Machine (SVM) classifier. The idea was simple: split the image region into small cells, compute a histogram of edge orientations in each cell, and concatenate these to form a single feature vector. This vector contained information of the human body edge structure and silhouette, which are relatively stable features regardless of clothes. HOG-SVM outperformed all previous pedestrian detectors on standard benchmarks and quickly became the method of choice. The big problem was speed: the dense sliding-window approach, which surveyed all potential locations and scales across a frame, was computationally expensive and hard to run in real-time. Performance degraded significantly under partial occlusion, poor lighting, and cluttered backgrounds, [2] all of which are routine conditions in real surveillance footage. Viola and Jones [3] proposed a more efficient method based on Haar-like features and a cascaded AdaBoost classifier. This approach attained real-time speeds by early rejection of non-person regions in the cascade. Initially, it was aimed to detect faces but it did not provide enough descriptive information to reliably detect full body people in different poses and from different viewpoints.

The most advanced classical approach was the Deformable Part Model (DPM) by Felzenszwalb et al.[4] Instead of viewing the body as a rigid template, DPM modelled it as a set of parts – head, torso, limbs – each with its own filter, connected with spring-like deformation costs. This made it robust to pose variation and partial occlusion and it won the PASCAL VOC detection challenge for several years in a row. However, DPM was too slow for real-time video and required careful manual calibration when de-ployed in new environments. All these approaches have the same common limitation: their representative ceiling. A researcher had to decide before-hand which visual cues – direction of gradient, colour, texture – were worth recording. More training data couldn't make up for features that weren't designed into the descriptor to begin with. This fundamental constraint was the main reason for our transition to deep learning, where feature representations are learned directly from data.

2.2 Deep Learning-Based Object Detection Methods:

The breakthrough came from Krizhevsky et al. [5] who showed that a deep Convolutional Neural Network, AlexNet, was able to beat all hand-crafted approaches on the ImageNet classification challenge by a large mar-gin. The critical insight was that a deep network could learn automatically the hierarchical representations: edges in the early layers, shapes in the middle, and semantically meaningful patterns further down the network, all without any manual feature engineering. This breakthrough changed the game of object detection and established deep learning as the domi-nating paradigm in computer vision.

Two-Stage Detectors:

The first generation of deep learning detectors adopted a two-stage pipeline, which first generates candidate regions and then classifies. Gir-shick et al. proposed R-CNN,[6] which first warped all the region proposals to a fixed size, then fed them into a CNN and classified them using an SVM. Better than HOG-based methods but way too slow — it could take more than 40 seconds to process about 2,000 proposals per image. R-CNN [7] eliminated this redundancy by feeding the entire image once into the

network and projecting proposals onto the resulting shared feature map. This brought inference time down to around two seconds. Faster R-CNN [8] completed the evolution by replacing the external proposal mechanism with a Region Proposal Network (RPN) that is trained end-to-end jointly with the detector. For years it was the gold standard for accuracy — but even at its fastest it couldn't top 7-10 frames per second, which is no good for the continuous, multi-feed requirements of live surveillance.

One-Stage Detectors:

Redmon et al. [9] recast detection as a single regression problem and introduced YOLO (You Only Look Once) to tackle the latency and complexity of the two-stage pipeline. The whole image is processed through the network only once. A grid of cells directly predicts bounding boxes and class probabilities. YOLO completely removes the proposal stage and is able to run at 45 frames per second – fast enough to be used in real time – while still being accurate enough to be useful. But accuracy for small or nearby objects was lost. Around the same time, Liu et al. proposed SSD (Single Shot MultiBox Detector), [10] which performs detections from feature maps at different resolutions in a single pass, addressing the scale variation weakness of YOLO, while keeping the real-time speed.

Evolution of YOLO:

YOLO architecture has evolved rapidly. YOLOv3 [11] introduced multi-scale prediction through a feature pyramid that substantially improved the detection of small and partially occluded people, an important feature for surveillance. YOLOv4 [12] incorporated several training refinements such as mosaic data augmentation, CIoU loss, and architecture improvements to reduce the accuracy gap with two-stage detectors while maintaining real-time speed. YOLOv8 [13] brought some architectural changes: an anchor-free detection head, which removes the need for manual anchor size specification, a decoupled head that does classification and localization in parallel branches, and a C2f module in the backbone that enhances gradient flow during training. In aggregate, these changes increased accuracy on standard benchmarks without hurting inference speed. YOLOv10[14]

went even further by addressing non-maximum suppression (NMS) – the post-processing step that removes duplicate detections, which introduces unpredictable latency and cannot be optimized as part of the network. YOLOv10 proposes a dual-assignment training strategy: one-to-many assignments for rich supervision at training and one-to-one assignments at inference for direct, NMS-free output. This results in a cleaner, faster pipeline with meaningfully lower end-to-end latency at equal accuracy to YOLOv8 on standard benchmarks.

Why YOLO for Surveillance?

In a surveillance system that monitors multiple camera feeds, detec-tion speed is non-negotiable. Frames need to be processed continuously, ideally at 25–30 fps per feed, often on hardware that is far from a high-end server GPU. This requirement cannot be met consistently by two-stage de-tectors. YOLO's single-pass architecture is naturally suited to this setting and studies evaluating detectors on dense pedestrian benchmarks such as CrowdHuman [15] consistently show YOLO variants providing the best practical balance between accuracy and throughput. YOLO models can also be exported to optimized inference formats such as TensorRT and ONNX, which allows further acceleration on embedded platforms com-monly used in surveillance infrastructure. The detection backbone for this project is YOLOv8 . Its anchor-free nature reduces the configuration over-head, its multi-scale feature pyramid retains accuracy for different distance and its inference speed allows real-time processing on target hardware. YOLOv10's NMS-free design offers a natural upgrade path in which end-to-end latency becomes the bottleneck.

2.3 Single and Multi-Object Tracking Algorithms

2.3.1 Single Object Tracking (SOT):

Single Object Tracking (SOT) is the problem of tracking a pre-defined single target in video frames by continuously locating it given its initial bounding box. It is adopted in surveillance to track a particular individual with partial occlusion, scale change and motion variation.[16] Before deep learning, correlation filter based methods dominated the SOT. The track-

ers learn a discriminative filter from the target appearance in frequency domain. Fast Fourier Transform (FFT) is used to generate a response map. The peak of the response map indicates the predicted target location.[17] This approach was introduced by Bolme et al. [18], who achieved tracking at nearly 600 fps by minimizing the squared error to a Gaussian shaped response. Henriques et al. [17] generalized the method with kernel methods and HOG descriptors, greatly increasing the discriminability of the target from the background while maintaining real-time performance. Later, the popular SOT framework became the Siamese network architectures. Bertinetto et al. [19] formulated tracking as a similarity learning problem: weight-sharing CNN branches compute the cross-correlation between a target template and a search region to produce a response map for localization. Li et al. [20] added a Region Proposal Network for bounding box regression, making it more robust to scale and aspect ratio changes. Later, Li et al. [21] combined deep ResNet-50 backbones with multi-level feature aggregation to achieve state-of-the-art results on OTB, VOT and LaSOT. More recently, transformer-based architectures have tackled the spatial limitations of convolution-based matching. Chen et al. [22] proposed a cross-attention for template-search feature fusion, which can capture long-range spatial dependencies that cannot be modeled by correlation filters. Ye et al. [23] further extended this with processing of both the template and the search region in a single unified transformer stream, enabling bidirectional feature interaction from the earliest encoding layers. Despite these advancements, SOT has fundamental limitations for real surveillance use. Trackers often drift over long occlusions, either losing the target or latching onto a similar background area.[24] SOT also needs the bounding boxes to be initialized manually and cannot track multiple persons at the same time.[25] Such constraints make SOT inapplicable as a stand-alone solution in cases where both identity persistence, occlusion recovery and continuous multi-person tracking are required.[26]

2.3.2 Multi-Object Tracking (MOT):

Multi-object tracking (MOT) aims at localizing and consistently main-

taining the identities of multiple targets across video frames .[26] Unlike SOT, MOT has to handle a dynamic number of people entering and leaving the scene at any time, making it far more suitable for real surveillance deployments.[26] The dominant MOT paradigm is tracking-by-detection: a detector finds all people in each frame and a data association algorithm links those detections across frames to produce continuous trajectories.[26] Bewley et al. [27] proposed SORT (Simple Online and Realtime Tracking), which combines Kalman filtering for motion prediction with the Hungarian algorithm for optimal assignment between detections and existing tracks. Computationally efficient but limited — SORT depended on only spa-tial overlap for association, susceptible to identity switches in crowded scenes and occlusion. Wojke et al. [28] solved this problem by using Deep-SORT which adds a deep appearance descriptor trained for person re-identification to the association pipeline . DeepSORT's ability to combine motion and appearance signals significantly reduced identity switches and improved tracklet recovery after occlusion – an important capability in environments where people regularly pass each other or move behind obstacles.

Later, graph-based formulations were explored to better model the spatial and temporal relationships. Wang et al. [29] defined detections as nodes of a graph and learned association weights using Graph Neural Networks (GNNs). This approach facilitates collaborative reasoning on all detections in a scene instead of pairwise comparisons independently, which is especially effective in crowd dense scenarios. Transformer based architectures pushed it even further. Meinhardt et al. [30] proposed TrackFormer, which propagates a set of track queries over frames and updates them with cross-attention to frame features, allowing end-to-end identity pre-diction without the need of a separate association module. Zhang et al.
[31] improved the association step by combining high- and low-confidence detections during matching in ByteTrack, which recovered briefly lost or partially occluded individuals that threshold-based filtering would discard. ByteTrack achieves state-of-the-art results on MOT17 and MOT20, yet re-

mains practical for real-time deployment. MOT is still a very challenging problem in unconstrained settings. The main causes of identity switches and track fragmentation are still long occlusions, heavy overlap of crowds and sudden re-entries.[32] The computation costs of multi-stage pipelines integrating detection, feature extraction, and association also impede real-time deployability onto edge surveillance hardware.[33]

2.4 Person Re-Identification (Re-ID):

A fundamental challenge for multi-camera surveillance networks that is not solved by tracking is that a person disappears from the field of view of one camera and reappears in another – often after a significant time gap, and across non-overlapping coverage areas. Person re-identification (Re-ID) is to address this problem of cross-camera identity matching. Early Re-ID methods adopted hand-crafted colour and texture features, e.g. SDALF [34] and LOMO[35], with metric learning. These methods provided useful baselines, but did not work well in practice with large viewpoint changes or changing lighting conditions. Deep learning revolutionized the field. Zheng et al. [36] proposed a discriminatively trained CNN with verification loss on the Market-1501 benchmark and obtained a significant improvement over the state of the art. The prevailing architecture rapidly changed to part-based models. PCB [37] divided person images into horizontal stripes to extract spatially aligned, fine-grained body-part features. HACNN [38] integrated attention mechanisms for selectively attending to discriminative body regions.

Transformer-based Re-ID methods have since pushed the performance further. TransReID [39] proposed the use of vision transformers for Re-ID with patch shuffling and side information encoding to improve cross-camera robustness and achieved 95.2One persistent and underappreciated challenge is domain gap: models trained on one dataset – say, an indoor, climate-controlled setting – often degrade significantly when deployed on footage from outdoor or low-resolution cameras.[41] Unsupervised domain adaptation methods such as MMT [42] and SpCL [43] have begun to address this, but it remains challenging to achieve robust cross-domain Re-ID

without target-domain annotations, a major hurdle for deploying reliable person tracking across diverse real-world surveillance settings.

2.5 Research Gaps and Motivation for the Proposed System:

It is evident that the state-of-the-art in person detection, tracking and re-identification has progressed significantly. Modern technologies are much faster, more reliable and more precise than previous methods. However, if we consider these techniques in the context of a real monitoring system, there are still a number of practical limitations. The most accurate models are computationally expensive, which is one of the major challenges. Advanced learning architectures and transformer-based techniques typically require high-performance GPUs to operate efficiently. In practice, these resources may not be available in many surveillance environments, particularly those using edge sensors or low-cost technology. Thus benchmark performance and real-world deployment feasibility are clearly not the same.

Another big problem is the slow loss of consistency of identity. Even the most advanced tracking systems can have trouble when a person is temporarily obscured , moves out of the camera's range , or reappears a few seconds later . However, even with more recent techniques such as Byte Track, where more detection information is used to improve resilience, it remains difficult to consistently maintain identity over time and across multiple camera views, especially in lightweight systems.

Besides, detection, tracking and re-identification are treated as separate components in most existing works. Each of these fields has had success in its own right, but true surveillance applications require that they all work together as a single coherent pipeline. In the literature still few fully integrated systems are available that operate in real time and require little operator intervention.

Another issue is that benchmark datasets are not a good representation of real surveillance situations. In fact, even though performance on standard datasets is good, low light, motion blur, camera noise and environment changes often lead to degraded performance.

Finally, models trained in one context often don't perform as well in

another. The layout of the scenes, the lighting and the camera angles can make a big difference to the accuracy. However, domain adaptation strategies are complex and require additional training, which limits their practicality for real-time deployment.

In conclusion, these disadvantages suggest that we need a system that is not only accurate but also lightweight, useful and robust enough to work reliably in real-world surveillance situations. The proposed system addresses these issues by combining tracking, detection and re-identification into a single real-time architecture that can be executed on generic hardware. The following chapters describe the design, construction and evaluation of the system.

Chapter 3

Requirement Specifications

3.1 Existing System:

Closed-Circuit Television (CCTV) networks make up the majority of traditional surveillance equipment used in establishments including colleges, hospitals, retail centers, and business offices. The sole purpose of these systems was to capture video material for preservation and post-event analysis. Although they have fulfilled this function for many years, their basic architecture renders them inadequate to meet the requirements of contemporary, proactive security management. A video management workstation is typically located in a central control room and connected to several IP or analog cameras dispersed across a building. Human operators are in charge of the entire process and must simultaneously monitor multiple live video feeds displayed on a wall of monitors. The processes of detection, identification, and tracking are not automated. The system is completely reactive — it records everything but comprehends nothing.

3.1.1 Architecture of Existing Systems:

A typical legacy CCTV surveillance system includes the following components:

Camera Network: A variety of fixed or pan-tilt-zoom (PTZ) analog or IP cameras installed at key points. Each camera works independently without any shared context or understanding of nearby zones.

Digital Video Recorder (DVR) or Network Video Recorder (NVR): A central recording device that captures and stores compressed video streams from each camera using H.264 or H.265 encoding. Video is held for a certain period of time before being overwritten.

Operator Monitoring Station: A computer running video management software (VMS) such as Genetec Security Center or Milestone XProtect . This allows human operators to view recorded or live video feeds.

Manual Alerting: The human operator is responsible for all decisions involved in incident detection and reaction. No identity-based notifications or automated flagging of suspicious activities.

3.1.2 Limitations of the Existing System:

The following limitations have been identified by review of the literature

and practical operational observations.

a) Total Dependence on Human Vigilance: The major drawback of traditional surveillance systems is their total dependence on human vigilance. Studies in cognitive psychology have shown that sustained alert-ness decreases rapidly during prolonged periods of monitoring. Research on CCTV operators, for instance, shows that the vigilance decrement (a quantifiable decrease in detection performance over time) generally manifests within 20 to 35 minutes of starting a continuous monitoring task [46]. In a facility with 16 to 64 cameras live at any one time, it is nearly impossible for one operator to maintain meaningful situational awareness of all feeds. Therefore, incidents are often missed in real time and only found in post-event review.

b) No Cross-Camera Identity Continuity: In a traditional system, each camera is a stand-alone sensor with no shared intelligence. There is no system mechanism to automatically associate two sightings when a person of interest moves from the coverage area of Camera A to the coverage area of Camera B. The operator is manually correlating each detection on each feed – a laborious, unreliable and completely unworkable process at scale.

c) High Human Error and ID-Switching Rates: Manual tracking in multiple simultaneous streams is especially error-prone in congested environments. Tracking people solely by how they look (their clothes , their body shape , the way they walk) in a busy hallway or door is often mistaken for someone else . This decreases the reliability of real time responses and introduces inaccuracies into post incident reconstructions.

d) Lack of Structured Activity Logging: While traditional systems do log unstructured video, they do not generate structured data that can be queried. There is no automatic record of which camera saw which person and at what precise time. To reconstruct a person's movement his-tory, you have to manually review hours of footage from multiple camera feeds, a process that can take many hours or even days.

e) Operator Fatigue and Cognitive Overload: Security personnel charged with continuous real-time surveillance are subject to considerable

mental fatigue over the course of a shift. As fatigue increases and reaction time slows, the likelihood of a security-relevant event being missed increases substantially. A study of 42 real CCTV operators doing a 90 minutes surveillance task showed that nearly 60

f) Scalability Limitations: The greater the number of cameras within a facility, the greater the number of feeds an operator must monitor concurrently. Increasing the number of operators is not a cost effective solution to this linear growth in cognitive burden. Traditional surveillance systems do not scale as the organization or infrastructure expands.

g) Intrusive privacy alternatives: Some commercial systems have tried to address the problem of identity continuity by using facial recognition. Facial recognition, however, is rife with serious privacy issues and heavily regulated under data protection regimes. Biometric data are seen as a special category of personal data under the European Union's General Data Protection Regulation (GDPR) that are subject to strict restrictions on processing including the need for explicit informed consent [3]. In addi-tion to legal constraints, facial recognition is also constrained by technical challenges such as degraded performance under varying illumination con-ditions, partial occlusion and non-frontal viewing angles, which makes it unsuitable as a general-purpose indoor tracking solution. In conclusion, the current paradigm for surveillance is essentially passive. It records ev-erything, but it doesn't actively comprehend anything. "The project is, essentially, about closing the gap between raw, unstructured video record-ing and actionable, intelligent security information."

3.2 Proposed System:

The proposed solution, Person Tracking Using AI and Surveillance Cameras is an automated and intelligent surveillance framework that is custom-built to tackle each limitation pointed out in the existing system directly. The key breakthrough is the creation of Identity Continuity — the automated ability to identify that a person caught by Camera B is the same person previously seen by Camera A, without any human intervention or facial recognition. To achieve this, the system integrates three

major technical subsystems: a real-time object detection engine, a two-level person re-identification (Re-ID) manager, and a centralized multi-camera dashboard supported by an automated relational database. Together, these elements transform the surveillance workflow from passive video recording to active, intelligent monitoring.

3.2.1 Proposed System to Overcome Existing Limitations:

Eliminates the need for human vigilance: YOLOv10 runs as a dedicated background thread, automatically detecting and locating each person in each incoming video frame without operator input. Continuous detection is done on processing speeds much faster than human reaction time. It signifies that no one entering a monitored camera zone can go undetected regardless of the level of alertness of the operator at any time.

Cross-Camera Identity Continuity Solved The Re-ID system: Ensures the identity consistency across camera boundaries by assigning a persistent PersonID to each detected person with the visual appearance embedding, not the spatial position. Person re-identification aims to match people across non-overlapping camera views by learning discriminative feature representations which are robust to variations in viewpoint, illumination and pose [4]. This technique directly solves the cross-camera continuity problem which is totally absent in traditional CCTV systems.

Lowers ID-Switching Errors: Different from the conventional tracking-by-detection methods that only consider the spatial closeness, the proposed system adopts deep feature embeddings extracted by a pre-trained ResNet18 backbone. Deep metric learning builds an embedding space where feature vectors of same identity are mapped closer than those of different persons, allowing for robust nearest-neighbour matching by cosine similarity [39]. This approach is much more robust to crowded scenes and temporary occlusion. The system also keeps a rolling history of up to five embeddings for each tracked unknown person to improve the robustness of matching over time.

Generates Structured, Searchable Activity Logs: Each tracking event is automatically committed to a SQLite3 relational database. Every

record has the PersonID assigned to it, the CameraID of the camera that saw it, and an accurate timestamp in ISO format. It would take hours of sifting through video to determine where someone had been. Instead, with a simple database query you could instantly pull up a complete history of someone's movements (in chronological order) across all cameras they appeared in.

Scalable and privacy-preserving design: The modular, multi-threaded architecture allows for the integration of additional camera feeds with minimal structural changes. Importantly, the Re-ID matching mecha-nism does not use facial recognition, does not access any biometric identity database, and does not store or infer any personal identity information. All tracking is performed using only generic visual appearance features. This design is in line with current data protection law and can be applied in a privacy-sensitive environment in an ethically and legally acceptable way.

3.2.2 System Overview:

The pipeline of the proposed system for every incoming video frame is as follows:

1. A dedicated VideoStream thread is spawned for each configured cam-era feed, which consumes the feed as a continuous video stream using OpenCV capturing frames at up to 720p resolution.

2. Each frame is sent to the DetectionWorker, which does YOLO infer-ence in a separate background thread so the GUI remains fully responsive. The system supports both YOLOv8 and YOLOv10 model architectures, inferring at a resolution of 480×480 pixels using FP16 half-precision on GPU hardware for best performance.

3. For each detected person bounding box, the PersonFeatureExtrac-tor extracts the individual from the frame and pushes the extracted im-age through a pretrained ResNet18 backbone with the classification head stripped. The output is a 512-dimensional feature embedding encoding the visual signature of the person.

4. The Reid Manager normalizes the embedding and performs a two-step matching procedure:

- **Tier 1 — Registered Person Matching:** The embedding is matched against all the profiles in the SQLite3 database (manually registered, named individuals). Finally, a cosine similarity score is computed for each profile held. If the highest similarity score is above or equal to the registered-person threshold (default: 0.7), the detection is assigned the matching PersonID.

- **Tier 2 – Unknown Person Tracking:** In the absence of a regis-tered match, the embedding is compared to in-memory embeddings of previously seen unknown individuals (IDs starting from 1000). If the highest similarity score is greater than the unknown-person threshold (default: 0.65) the detection is assigned the existing unknown PersonID and the new embedding is added to the rolling history of that individual (up to five embeddings). A new unknown PersonID is created if no match is found in either tier.

5. All solved tracking events (PersonID, CameraID, camera name, confidence score and timestamp) are automatically written to the detection history table in the SQLite3 database.

6. The detection results are returned to the GUI thread via Qt sig-nals, where the gui main module displays annotated frames with bounding boxes and PersonID labels on the live multi-camera dash-board, which refreshes at 20 FPS.

3.3 Requirement Specifications:

This section presents a formal specification of the requirements for the proposed system, arranged into functional requirements, non-functional requirements, and system constraints.

3.3.1 Functional Requisites: Functional requirements describe the specific actions , capabilities and functions the system is required to perform.

FR-01 – Real-Time Person Detection: The system shall identify each person in each frame of each camera in real time using YOLOv8

or YOLOv10 architecture. Detection shall be performed continuously on all active camera feeds, no manual triggering shall be required, a configurable confidence threshold (default: 0.5) and IoU threshold (default: 0.45) shall be used.

FR-02 — Bounding Box Generation: The system shall generate a bounding box for each detected person, indicating the spatial extent of the person in the frame . Each bounding box needs to be visually displayed on the dashboard and tagged with the assigned PersonID.

FR-03 — Feature Embedding Extraction: The system shall extract the 512-dimensional feature embedding of each detected individual by the pre-trained ResNet18-based Re-ID model with the classification head replaced by an identity layer. This embedding acts as the visual signature of the person for cross camera matching.

FR-04 — Two Tier Cross Camera Identity Matching: The system will perform a two stage matching process. In the first tier, cosine similarity will be calculated between the new embedding and all the registered person profiles in the database; a match above a threshold of 0.7 assigns the registered PersonID. In the second tier, if no registered match is found, cosine similarity will be calculated with in-memory embeddings of unknown tracked people. A match higher than 0.65 assigns the existing unknown PersonID.

FR-05 — Persistent Identity Tracking: If the Re-ID matching criteria are satisfied, the system shall provide a consistent PersonID for an individual across all monitored camera feeds for the duration of their presence.

FR-06 — Multi-Camera Feed Support: The system must be able to ingest and process at least two concurrent video streams. Each feed must be processed on a separate thread. Support for local webcams and network IP cameras (HTTP/RTSP) shall be provided.

FR-07 — automated event logging: Each tracking event will automatically log in the SQLite3 database. Each record shall have a

unique PersonID, a CameraID, the camera name, a confidence score and a precise timestamp in UTC format.

FR-08 — Database Query Support: The system shall support querying the tracking database by PersonID to obtain a complete movement history of a given person, i.e., reconstructing the path of that person over the cameras.

FR-09 — Live Multi Camera Dashboard: The system shall provide a Graphical User Interface based on PySide6 to display all the active camera feeds simultaneously. Each feed shall display annotated frames with PersonID labels and bounding boxes updated in real time at 20 FPS.

FR-10 – Person Registration: Authorized operator shall be able to register identified persons in the system by supplying reference photographs. The system shall extract the average feature embedding of the reference images and store it for Tier 1 matching.

FR-11 — Session Control: An authorized operator shall be able to start and stop individual camera monitoring sessions or all camera monitoring sessions at once through the dashboard interface.

3.3.2 Non-Functional Requirements:

Non-functional requirements specify the quality attributes and operational constraints of the system.

NFR-01 — Performance (Real-Time Processing): The system shall have a minimum frame rate of 15 frames per second per active camera feed on the target hardware configuration. GUI update rate shall be at 20 FPS regardless of inference load.

NFR-02 — Detection Accuracy: The person detection module shall be able to detect persons with a minimum of 85

NFR-03 — Re-ID Matching Accuracy: The Re-ID system must achieve a cross-camera identity retention rate significantly better

than baseline tracking-by-detection methods as measured by ID-switching rate in controlled multi-camera evaluation.

NFR-04 — UI Responsiveness: The dashboard user interface shall be fully responsive, with no perceivable freezing or latency, regardless of the concurrent AI inference load. This is realized by a multi-threaded architecture in which the GUI rendering thread is completely separated from the YOLO inference and Re-ID processing threads.

NFR-05 – Reliability: The system shall operate continuously at least for eight hours without data loss and application crash. All tracking events should be immediately committed to the database upon generation, so that in the event that the process ends unexpectedly, no data is lost.

NFR-06 - Privacy Compliance: The system shall not use automated facial recognition, shall not interface with any external biometric identity database, and shall not store or infer personally identifiable information beyond manually registered profiles. All automated tracking will be based only on generic visual appearance features extracted from the ResNet18 embedding model.

NFR-07 — Usability: The dashboard interface shall be effectively operable by a security operator without any prior technical knowledge of artificial intelligence or computer vision after a familiarization period of no more than 30 minutes.

NFR-08 — Scalability: The system architecture shall enable the system to be scaled beyond the current two-camera configuration by adding processing threads and camera entries, with no need for structural changes to the core codebase.

NFR-09 — Hardware Accessibility: The system shall be deployable on mid-range consumer or prosumer GPU hardware (NVIDIA GTX 1060 or equivalent and above) without the need of enterprise-grade server infrastructure. Support for fallback to CPU-only oper-

ation shall be provided.

NFR-10 — Maintainability: The source code should be divided into well separated modules: detection$_{w}$*orker, reid*$_{m}$*anager, person*$_{d}$*b, video*$_{s}$*tream,*

3.4 Use Cases:

· The main interactions between system actors and the proposed system are described in the following use cases. In all cases the Security Operator is the main human actor.

Use Case UC-01: Begin Monitoring Session

· Designation :Security Operator

· Precondition: Camera feeds are set and system is running.

· Main Flow: The operator launches the dashboard application, selects the camera feeds to be active and starts the monitoring session. For each feed selected the system launches a VideoStream thread and a DetectionWorker.

· Postcondition: All selected feeds are live with bounding boxes and PersonID labels annotated and tracking events are being logged to the database.

Use Case UC-02: Automated Person Detection and ID Assignment

· Actor: Robotic System

· Precondition: An active monitoring session exists.

· Main Flow Someone enters the field of view of the camera. YOLOv10 detects the person and outputs bounding box. ResNet18 extracts a 512 dimensional embedding. The Reid Manager fails to find a match at either level. A new unknown PersonID (e.g. P1001) is generated, the embedding is saved in memory and the detection event is logged into the database.

· Postcondition: The person is assigned an ID that is displayed on the dashboard and their tracking history starts.

Use Case UC-03: Cross-Camera Re-Identification

· Actor: Automated System

· Precondition: Person P1001 is detected by camera 1 before and has left the field of view of camera 1

· Main Flow P1001 enters the view of Camera 2. We obtain a new em-bedding. Tier 2 embeddings are compared to all stored embeddings using cosine similarity. The similarity relative to embeddings stored by P1001 is above the threshold of 0.65. This detection is assigned to PersonID P1001 and the transition event is logged.

· Postcondition: The identity continuity is preserved across the cameras. The dashboard is showing P1001 on the Camera 2 feed correctly.

Use Case UC-04: Check History of Movement

· Security Operator (Actor)

· Precondition: A monitoring session is active and events are recorded.

· Main Flow: The operator selects a person from the Registered Persons List or enters a PersonID. The system then queries the detection history table and retrieves the list of entries for that person in chronological order, including the CameraID, camera name, confidence score, and timestamp.

· postcondition: The operator has a complete time-stamped movement history for the person of interest.

Use Case UC-05 – Discovery of a New Unknown Person

· Actor: Automatic system

· Precondition: An active monitoring session Main Flow: A person whose embedding does not match any profile in either matching tier enters the camera zone. The system creates a new PersonID for an unknown person (starting at 1000), saves the embedding in memory, and starts tracking the person as a new entity.

· Postcondition: New individual is assigned a unique ID and starts building up detection history.

Use Case UC-06: Terminate the Monitoring Session

· Role: Security Operator

- Prerequisite: A monitoring session is in progress.
- Main Flow: The operator clicks on the stop button. The system then gracefully terminates all VideoStream and DetectionWorker threads, flushes all pending writes to the database, and returns the dashboard to an idle state.
- Postcondition: All tracking data are safely stored in the database. There was no loss of data.

Chapter 4

System Design

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. This chapter should have the following sections:

4.1 System Architecture

The proposed surveillance system consists of four interconnected subsystems that operate as a continuous automated pipeline. Each component plays a specific function and when they combine they can turn unstruc-tured camera footage into tracking intelligence without the need for human intervention.

Overview:

At the lowest level the system accepts live video inputs from multiple cameras, and produces two outputs: an annotated live display for the security operator, and a structured database of monitoring events for forensic and historic analysis.

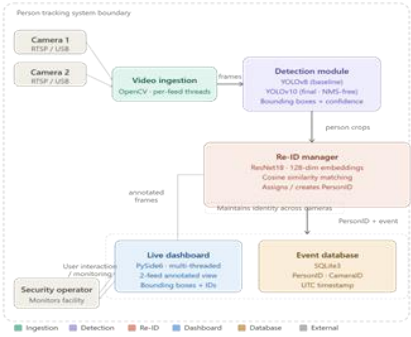


Figure 4.1: High-Level System Architecture Diagram

The Four Subsystems

4.1.1. Video Ingestion Layer

This is the entry point into the system. OpenCV [50] can record any connected camera feed, such as live RTSP network streams, USB webcams,

or saved video files. Each feed is processed in its own background thread, so multiple cameras can be processed concurrently and independently. No feed ever waits on another. This layer manages all communication between the actual cameras and the rest of the system.

4.1.2. Detection Module (YOLOv8 / YOLOv10)

The Video Ingestion Layer provides each frame to the Detection Module. This subsystem consists of two deep learning based object identification models, namely YOLOv8 and YOLOv10. We selected YOLOv8 [13] as the baseline model for its stability and easy integration in the early stages of development. Later, YOLOv10 [14] without NMS was tested, showing a significant reduction in processing cost and delay. The comparison led to the final deployment pipeline being YOLOv10 and YOLOv8 being retained for benchmarking. The model automatically outputs bounding boxes and confidence scores for each person in the frame. Only detections with a confidence value higher than a predefined threshold are passed on to the next level.

4.1.3. Re-Identification (Re-ID) Manager (ResNet18)

This is the most important subsystem from the technological point of view of the project. The Re-ID Manager crops each detected person out of the frame and then runs that crop through a ResNet18 [45] neural network to generate a 128-dimensional feature embedding. This feature embedding is a condensed numerical signature that embodies a person's visual characteristics such as clothing color, body proportions, texture, and silhouette shape. Then, this embedding is compared to all previously stored embeddings using cosine similarity. If a stored embedding is found that is above a pre-defined similarity threshold, the system identifies a person that has been seen before and assigns them their current PersonID. If there is no sufficient match, a new PersonID is generated and the new embedding is stored for later comparisons. This method ensures identity continuity between non-overlapping camera zones.

4.1.4. Dashboard and Database Module (PySide6 + SQLite3)

The last subsystem is responsible for two tasks simultaneously. Each tracking event, i.e. when a specific camera sees a PersonID, is immediately recorded to a relational database (SQLite3) with the CameraID and an exact timestamp. The movement of each individual in the monitored area is therefore permanently registered and available. The graphical dash-board using PySide6 displays bounding boxes and PersonID labels for each tracked individual and shows a multi-feed view of the live annotated video frames to the security operator. This is a multi-threaded design, so the dashboard will always be fully responsive, no matter how much work is going on in the background. How the Subsystems Connect

Each frame moves through the system in a fixed linear sequence:

Camera Feed → Video Ingestion → YOLOv8/YOLOv10 Detection → ResNet18 Re-ID → Cosine Similarity Matching → Database Logging + Dashboard Display.

This pipeline runs continuously and in parallel across all connected camera feeds, ensuring real-time performance is maintained at every stage.

Interface with External Systems

The only external access is via the camera network itself. The cameras are connected using OpenCV compatible inputs like RTSP streams, USB devices or video files. No identity databases, no cloud platforms, no network APIs, no external services. The system is completely self contained and runs only on a local machine. This decreases latency, eliminates the need for third-party infrastructure, and ensures that the system is functional without internet connectivity.

4.2 Design Constraints

The limitations that influenced important design choices during the creation of the suggested surveillance system are described in this section. Some of these forced intentional trade-offs between conflicting agendas, while others set strict limits on what the system could do. This paper also includes the assumptions established throughout the design phase.

Hardware Constraints:

The largest constraint for the entire system architecture was the requirement to run on mid-range consumer GPU technology. Enterprise-grade surveillance AI systems typically require high-end GPUs and dedicated server infrastructure. This was not an option for our project and impacted every major technical decision. This is the reason that YOLOv10 [14] is preferred over YOLOv8 [13] as the detection model. YOLOv8 uses a post-processing technique called Non-Maximum Suppression (NMS). This adds computational complexity to the algorithm, which on limited hardware could mean the difference between real-time performance and not achieving an acceptable frame rate. The decision was easy, a more complex model would have slightly better accuracy but would not run in real time on the intended hardware. Similarly, the Re-ID module was reasoned. We used ResNet18 [45] as the base network instead of more complex architectures such as ResNet50 or ResNet101. To be used in practice, the ResNet18 pro-duces 128 dimensional embeddings that are discriminative enough to be used for crosscamera matching and are more lightweight. Here, real-time multi-camera processing is impossible because the additional latency will be added to a deeper backbone, which can extract richer features.

Real-Time Processing Constraint:

The system's strict real-time operation constraint led to the elimination of several methods that might have produced more accurate results in a laboratory setting. For example, batch processing, which groups several frames together and processes them as a group, can improve detection accuracy but also introduces noticeable lag. Instead, a frame-by-frame processing was used to keep the live, continuous quality a surveillance system requires. The same requirement also affected the multi-threading architecture. Each camera feed runs in a separate background thread, independent of the dashboard rendering thread. Without this separation, AI inference would block the user interface and cause it to stop under high processing loads. This is obviously not acceptable in a live monitoring situation.

Privacy Constraint:

And facial recognition. And any type of biometric identification link. It was a conscious design decision not to allow any of the above. This was both a practical and an ethical decision. Facial recognition has its drawbacks, such as the need for frontal face visibility, which is seldom guaranteed by surveillance footage, leading to legal problems under data protection standards, and degrading significantly in different lighting conditions, partial occlusion, or non-frontal poses.

In this work, we track people by 128-dimensional embeddings of generic visual appearance parameters, e.g., clothing color, body proportions, texture etc. The downside is that appearance-based Re-ID is less reliable than face recognition, especially when people change their appearance across camera zones or dress similarly. However, this compromise is acceptable for the intended interior conditions and the system remains privacy compliant.

Software and Dependency Constraints:

The system was built using only open-source libraries: PyTorch [51], OpenCV, PySide6 and SQLite3, thereby eliminating external dependencies and licensing costs. They were also selected for their lack of need to internet connectivity at runtime, for being well maintained and having extensive documentation. SQLite3 was chosen for the database instead of a full client-server database like PostgreSQL or MySQL. A full database server would have made multi-user access and extremely large query loads easier to handle, but would also have added a great deal of setup complexity. SQLite3 is more than sufficient for a project of this scale – 2 cameras and one operator workstation – and maintains the portability and self-containment of the system.

Environmental Constraint:

The system was designed and validated in indoor and semi-enclosed places such as hallways, offices and building interiors. This is due to the rela-tively controlled and consistent lighting of interior contexts upon which appearance-based Re-ID relies. There were outdoor scenarios outside the

certified working range that were not covered, such as direct sunlight, rain, harsh shadows and complete darkness. Not the target scope either, too much crowd density. In very crowded scenes, the cosine similarity matching threshold would need to be significantly recalibrated due to the high frequency of occlusions and the high visual similarity between people. We acknowledge this as a limitation and as a direction for future work.

Design Assumptions:

During the design phase the following assumptions were made and main-tained throughout the development:

Under normal operation each camera feed provides a consistent unbroken video stream with no visible dips in frames. The basic assumption behind cross-camera Re-ID is that the two cameras are placed such that their fields of view do not overlap, so that the same person cannot be seen in both feeds at the same time.

The person being tracked is assumed to move at normal walking speeds. It wasn't designed for running people or fast-moving targets. The lighting is expected to be fairly constant during a monitoring session. Sudden drastic changes such as a power failure do not meet the target operating conditions. The system does not support concurrent multi-user access to the dashboard or database during a live session, but is assumed to have a single operator workstation. All of these constraints and assumptions together defined the boundaries within which the system was to operate and guided all the major design decisions.

4.3 Design Methodology

The surveillance system is developed using object oriented design methodology. This methodology was well suited for the requirements of the project. The system is organized around four different subsystems, each one built as a separate component with well-defined responsibilities: video ingestion, detection, Re-ID and the dashboard/database module as explained in Section 4.1. This separation allowed each part to be evaluated and improved independently, and protected other subsystems from unintended side ef-

fects of changes in one subsystem.

Iterative Model Selection:

Design choices were made iteratively throughout the project by means of implementation, assessment and improvement rather than being predetermined. This was especially true in the selection of the object detection model. YOLOv8 [13] was chosen initially due to its stability and ease of integration and was used as the baseline throughout the whole development and testing. Then, YOLOv10 [14] was published as an experimental alternative and its NMS-free architecture was compared with YOLOv8 in terms of processing overhead and latency. In this test, we combined both models with DeepSORT [28] to perform multi-object tracking. DeepSORT improves on the original SORT algorithm by adding appearance-based features in addition to motion cues, making it ideal for keeping track continuity in cluttered or occluded scenes. Finally, from the comparative analysis, YOLOv10 was chosen as the final deployment pipeline, and YOLOv8 was retained for benchmarking.This iterative process went beyond just choosing a model. For example, the multi-threading architecture was not part of the original design, but was added since the single-threaded processing turned out to be insufficient for real-time multi-camera operation.

Design Conventions:

A common set of norms was maintained during development to ensure the system was structured and flexible. We used ResNet18 [45] as a light-weight but efficient backbone to extract 128-dimensional feature embed-dings for the Re-ID module. We compare high dimensional feature embed-dings across camera feeds using cosine similarity, which is often used in person re-identification tasks [39]. The configuration is not hard coded but centralized so you can change model routes, similarity scores, confidence thresholds etc. without changing the core logic of the system.

The system is fully self-contained, as it relies solely on open-source libraries, e.g. PyTorch, OpenCV, PySide6 and SQLite3, and doesn't have any external services or cloud dependencies.

In Section 4.1, it is said that data flows through the system in a defined linear pipeline, which ensures deterministic behavior in each step. In con-clusion, the approach favored practical validation over rigid upfront plan-ning, allowing the design to evolve based on real performance data, while firmly grounded in object-oriented principles.

4.4 High Level Design

4.4.1 Conceptual / Logical View:

This view identifies the logical functional elements of the system and their relationships, without regard to runtime or physical deployment concerns.

The system is separated into four logical parts:

Video Ingestion Component The Video Ingestion Component feeds the raw frames into the pipeline. It connects to the camera sources such as a USB webcam, RTSP stream, or OpenCV video file. It isolates the camera hardware from system. The **Detection Component** utilizes the YOLOv10 model (compared with YOLOv8 during development) on raw frames to generate person bounding boxes and confidence scores. Detec-tions with confidence higher than the threshold are propagated. YOLOv10 was chosen due to its faster inference speed and fewer parameters com-pared to its previous versions [Ultralytics, 2024]. The **Re-ID Manager Component** takes cropped person images from the Detection Component and outputs 128-dimensional feature embeddings using ResNet18 back-bone. It contains the core identity continuity logic, with an in-memory store of embeddings and assigning or creating PersonIDs via cosine sim-ilarity matching. ResNet18 is a popular model in person re-identification due to its balance between accuracy and computational efficiency [He et al., 2016]. The **Dashboard and Database Component** stores tracking results and provides real-time visualisation via the user interface. It logs tracking events (PersonID, CameraID, timestamp) to SQLite3 and draws annotated video frames with bounding boxes and ID labels in PySide6

4.4.2 Process View

This perspective gives the runtime behaviour of the system, i.e. how threads

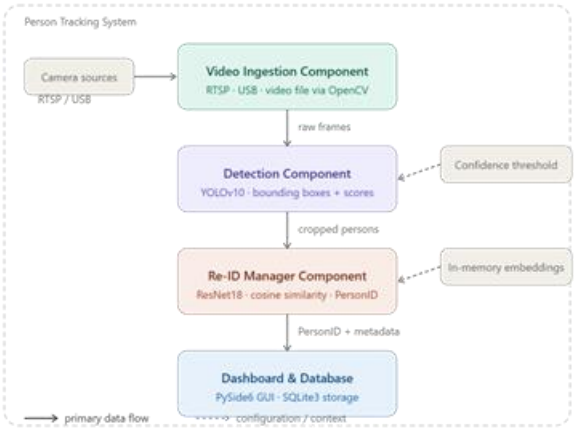


Figure 4.2: Figure 4.2: System Architecture Diagram-Conceptual/Logical View

and processes interact during live execution. The system has a multi-threaded architecture to ensure that AI inference never blocks the user interface. This design enables simultaneous frame acquisition, processing and visualisation without any interference between the three processes. Camera Threads OpenCV runs a background thread for each connected camera feed. These threads continuously grab frames and enqueue into per-camera frame queues. Thread processing (per camera): Each camera has a paired processing thread that dequeues frames, runs YOLOv10 detection (validated against YOLOv8 during development), passes crops to the Re-ID Manager, performs cosine similarity matching, and commits results to the SQLite3 database. UI Thread (main thread): The PySide6 dashboard runs on the main app thread. The signal architecture provided by Qt [Qt Documentation, 2024] is used to send annotated frames from processing threads to the UI thread using thread-safe signals and slots, so that the dashboard is always fully responsive regardless of inference load.

Database writes happen synchronously in the processing thread, but with SQLite3's built-in handling of connections.

4.4.3 Physical View

This view shows the mapping of the software components of the system onto the physical hardware nodes. This is a single machine deployment, not a distributed system. However, the physical view is still valid as it documents the components that run on a particular processor and device. The system deploys across two physical node types: Host Machine (primary processing node) — a single workstation or laptop that runs all software components. It has two deployment-relevant logical processors. The application process is run on the CPU. It ingests the video using OpenCV, has the PySide6 dashboard, writes to the SQLite3 database, and coordinates the threads. The inference workloads are run on the GPU (NVIDIA GTX 1060 or higher, CUDA-enabled), with YOLOv10 detection and ResNet18 Re-ID embedding extraction using PyTorch [Paszke et al., 2019]. YOLOv8 was also evaluated during development on this same node. Performance is contingent upon the availability of a GPU and in GPU-less environments (cf. using the CPU only), the frame rate might be decreased. Camera de-vices (peripheral input nodes) – one or more cameras connected via USB (direct hardware interface) or local network (RTSP stream). These are ex-ternal physical nodes that feed raw video data to the host machine, and do not process themselves. No external network dependencies, no cloud nodes, no remote servers. All processing is contained within the host ma-chine node.

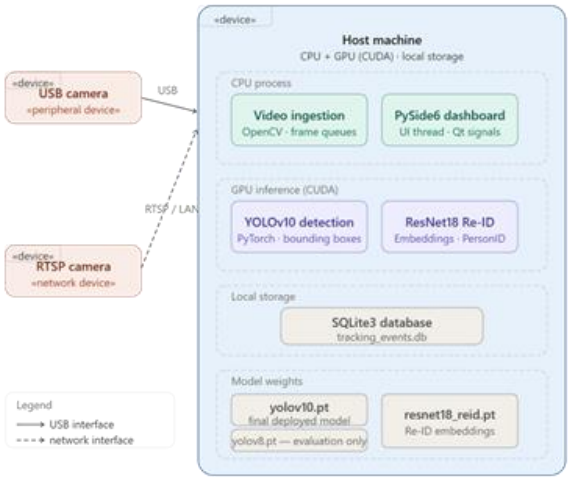


Figure 4.3: Figure 4.3: UML Deployment Diagram — Physical View show-ing hardware nodes and software artefact mapping.

4.4.4 Module View

This view shows the code and directory organization of the project, useful for development and project management. Each directory maps to one of the four logical components from the Conceptual View, making the codebase straightforward to navigate, maintain, and extend.

4.4.5 Security View

This view is a subset of the Conceptual View (Section 4.4.1). It concerns the components that collaborate to offer the security and privacy prop-erties of the system. Because the Video Ingestion and Detection compo-nents only deal with raw frame data in the local pipeline, they have no direct security responsibility. The two security active components are Re-ID Manager and Dashboard and Database Component . Re-ID Manager is the primary privacy guarantee of the system. It does not extract or store facial biometrics, but rather uses only appearance-based embeddings

- clothing colour, body shape and silhouette - generated by the ResNet18 backbone. This means that no personally identifiable information (PII) such as facial data is stored or inferred at any time, ensuring the system

is compliant with data protection principles such as those described in the General Data Protection Regulation (GDPR) [European Parliament, 2016]. Database and Dashboard Components Implement two types of access control. The SQLite3 database is stored locally on the host machine and secured by operating system file permissions. There is no interface for accessing the database remotely and therefore it is not possible to query or modify the database remotely from any location other than the host machine [SQLite Documentation, 2024]. The current PySide6 dashboard is physically access controlled; i.e. only an operator with physical access to the host machine can start or stop a session, which is a simple form of operational authorization.

System-wide security property: The system does not make any outbound network connections and does not have an internet dependency. All processing takes place on the device, eliminating any risk of data intercep-tion, remote exploitation, or unauthorized data exfiltration. The diagram shows this as the security perimeter boundary enclosing all components. In the future, an operator login authentication will be added to the PySide6 dashboard, removing the physical access control and replacing it with a credential-based authorization.

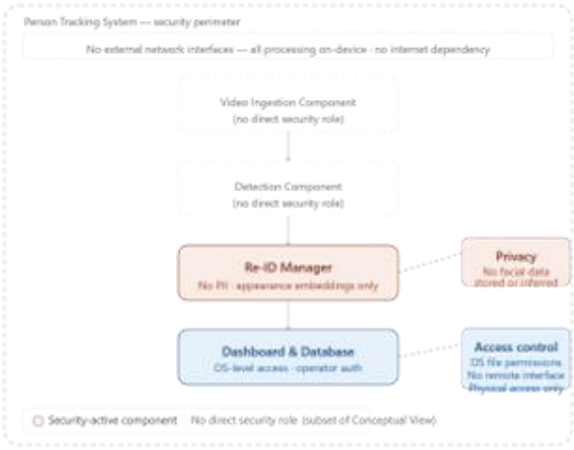


Figure 4.4: Figure 4.5: Security View — subset of the Conceptual View showing security-relevant components and their responsibilities.

4.5 High Level Design

4.5.1 Module Breakdown

The system is composed of five main modules, each with a specific function in the general pipeline. The following is: Video Ingestion, Detection, Tracking, ReIDManager and Dashboard DB. YOLOv10 detection and DeepSORT tracking are different stages with different responsibilities: the one is to find people in a frame, the other to decide if each found person is a new track or an existing one. A four-module model is less true to the real runtime flow than this five-module one. Together they feed a single, coherent identity to the Re-ID layer. Below is a short description of each module, including its constituent classes, attributes and key methods.

4.5.2 Video Ingestion Module

This module deals with the most basic of system concerns: getting raw video frames from whatever physical or virtual camera source is being used and making them available to the rest of the pipeline in a consistent, hardware-agnostic manner. After this layer, all the downstream modules see the same thing, where a USB webcam appears the same as an RTSP

network stream. Class: CameraFeed Responsibility: Handles one camera connection. It performs its capture loop in a dedicated background thread so the main processing logic is never blocked waiting for a frame to come from the hardware. camera id (str) : A unique label for this feed used throughout the system

source (str | int) : The OpenCV-compatible source. An RTSP URL, local file path, or USB device index.

frame queue (Queue) : A thread-safe buffer that decouples capture speed from processing speed.

capture (cv2.VideoCapture) : the OpenCV handle to the video device;

isrunning (bool) : a flag that the capture thread checks on every iteration to determine whether to keep going. Methods

start() – opens the capture device and spawns the background thread

stop() – sets are running to False and releases the opencv handle cleanly

_capture loop() – the private thread target that continuously reads frames and puts them in frame queue

get frame() – returns the latest frame from the queue immediately, or None if the queue is empty, so callers never block

Class: Camera Manager

Responsibility The only point of contact for the rest of the system whenever any camera's frame is needed. It owns and manages all active instances of CameraFeed for the session. feeds (Dict[str, CameraFeed]) – A registry mapping each camera_id to its CameraFeed instance.

Methods: add feed(camera_id, source) – creates a new CameraFeed and immediately starts it; remove_feed(camera_id) – stops and removes the named feed; get_frame(camera id) – delegates to the appropriate CameraFeed and returns its latest frame; stop_all() – gracefully shuts down all registered feeds at session end. 4.5.3 Detection Module This module wraps the YOLOv10 deep learning model (with YOLOv8 kept for benchmarking comparison) and exposes a single clean interface: give me a frame, I will

give you back a list of people and where they are. It abstracts away all the complexity of model loading, gpu placement, confidence filtering, and bounding-box parsing from the rest of the system.

Class: Person Detector

Responsibility: executes one forward pass of YOLOv10 on an incoming frame and returns only those detections that are classified as “person” and have a confidence value greater than the configured threshold. Its output directly goes to the downstream Tracking Module. Attributes

model_path (str) Path to the YOLO weights file on disk

confidence threshold (float, default 0.45) Detections below this score will be ignored before being passed on

device (str) Set to “cuda” when a compatible GPU is available, otherwise “cpu”

model (YOLO) The loaded Ultralytics YOLO model object

Methods: __init__(model_path, confidence threshold) – loads the YOLO model on the selected device; detect(frame) – runs inference and returns a list of Detection objects; filter_persons(results) – a private helper that keeps only class-0 detections above the threshold.

Data Class: Detection

A lightweight data container that flows from the Detection Module to the Tracking Module. It contains: box (tuple [int, int, int, int]) — the bounding box as (x1, y1, x2, y2) pixel coordinates in the original frame; confidence (float) — the model’s confidence score for this detection; crop (np.ndarray) — the cropped pixel region of the detected person, which the Tracker passes on to Re-ID once a stable track ID has been confirmed.

4.5.4 Tracking Module (DeepSORT)

The Tracking Module is an important but easily forgettable component of the pipeline, lying between detection and re-identification. YOLOv10 is a frame-by-frame detector, it doesn’t remember anything from the previous frame. No tracker - each frame gives a completely new set of detections, no

continuity. DeepSORT fills this gap by maintaining a set of active tracks, and associating each new detection to the right existing track using a combination of Kalman-filter motion prediction and appearance-based matching. confirmed stable tracks (lasting at least n_init consecutive frames) are passed on to the Re-ID layer, which greatly reduces the number of spurious or noisy detections that would otherwise pollute the identity store.

Class: Tracker

Responsibility: It receives a list of Detection objects from one frame, updates internal DeepSORT state, and returns a list of TrackedPerson ob-jects. Each TrackedPerson object has a confirmed track ID, current bound-ing box, and person crop. If a track is not matched during max age succes-sive frames, it is automatically dropped. Attributes: tracker (DeepSort) – the initialized DeepSORT object from the deep sort realtime library max age (int, default 30) – number of unmatched frames before a track is considered lost and deleted n init (int, default 3) – number of consecutive matched frames required before a track is confirmed and passed down-stream camera id (str) – identifier of the camera this tracker instance is serving, as each camera has its own independent Tracker **Methods:** update(detections) - main method, accepts the list of Detection objects from the current frame, passes it to the DeepSORT internals and returns a list of TrackedPerson objects for all currently confirmed tracks; predict() - moves the Kalman filter state for all active tracks into the next frame, yielding a motion-based position estimate which is used as the starting point for the next round of association; reset() - clears all track states, called when a monitoring session is terminated.

Data Class: TrackedPerson

The output of the Tracking Module, and the input of the ReIDManager. Fields: track_id (int) – a short-lived track ID given by DeepSORT, only valid within a single camera session box (tuple [int, int, int, int]) – the current bounding box crop (np.ndarray) – the corresponding image crop for feature extraction. Note that track id is not PersonID. When a person

re-enters the scene, DeepSORT's track_id will be reset, but the PersonID assigned by the ReIDManager is the persistent, cross-camera identity the system is ultimately trying to maintain.

4.5.5 ReIDManager Module

This is the most technically demanding module in the system. It takes the validated tracked persons from the Tracking Module and determines if the person is a person the system has seen before, perhaps in another camera, or a new person. It does so by maintaining a persistent in-memory gallery of 128-dimensional appearance embeddings, matching each incoming crop against that gallery using cosine similarity. The module is three classes that work together.

Class: Feature Extractor

Responsibility: Employing a ResNet18 backbone, it converts a person crop image into a compressed, normalized numerical fingerprint (128 dimensional embedding vector). The last fully-connected classification layer of the standard ResNet18 is replaced by a 128-unit linear projection to make the network output an embedding instead of a class probability. **Attributes**: backbone (torchvision.models.resnet18) – the modified ResNet18 with 128-unit projection head transform (torchvision.transforms. Compose) - a preprocessing pipeline, which resizes crops to 128 $\times$ 64 pixels and normalizes to ImageNet statistics device (str) - the compute device for inference. **Methods:** __init__(weights path, device) – loads and freezes the backbone weights; extract(crop) – takes a np.ndarray crop, runs the transform and a forward pass, and returns an L2-normalised numpy vector of length 128; preprocess(crop) – private helper that deals with colour-space conversion and tensor wrapping before the forward pass.

Class: Identity Store Responsibility: The in-memory gallery that is persisted across cameras for the whole session. This implements the match-ing logic: compare a new embedding to every stored embedding, return existing PersonID if a close enough match is found, or create a new one if not. Attributes: embeddings (dict[str, np.ndarray]) – the gallery mapping

each PersonID to its stored embedding vector; similarity_threshold (float, default 0.75) – the cosine similarity score above which a new detection is considered a match; next_id (int) – a monotonically increasing counter for generating new PersonIDs. **Methods:** match or register(embedding) – the key method; calculates cosine similarity with all stored embeddings and if the score exceeds the threshold, it returns the best matching per-son id, otherwise it allocates a new person id and stores the embedding; cosine similarity(a, b) – a private static helper method that calcu-lates the dot-product similarity score between two L2-normalised vectors; update embedding(person id, new_embedding) – applies an exponential moving-average update to the stored embedding in order to accommodate gradual appearance changes across camera zones; clear() – resets the entire gallery at session end.

Class: Reid Pipeline

Responsibility An interface that wires together FeatureExtractor and Iden-tityStore as a single callable step for the processing thread. It takes a TrackedPerson and returns a TrackingResult, so the processing thread never needs to directly interact with the two inner classes. **Methods:** process(tracked person) – calls extractor.extract(tracked person.crop), then store.match_or_ register(embedding). and returns a TrackingResult that contains the box, assigned PersonID, camera id, and the cosine simi-larity score that led to the assignment decision.

4.5.6 Dashboard Database Module

This module handles the two output-side responsibilities of the system: writing every tracking event permanently to the SQLite3 database for later forensic use, and rendering the annotated live feeds in the PySide6 dashboard for the security operator watching in real time. Class: Tracking Database Responsibility: Provides a clean abstraction over the SQLite3 tracking_events table so that no other part of the system ever writes SQL directly. All database interaction in the system goes through this class. **Attributes:** db path (str) — the file path to the SQLite3 database; con-

nection (sqlite3.Connection) — a persistent connection opened once at initialization and reused for the entire session.

Methods: __init__(db_path) — opens the connection and calls _create_tables(); _create_tables() — issues a CREATE TABLE IF NOT EXISTS for the tracking_events table; log_event(person_id, camera_id, timestamp) — inserts one row; query_by_person(person_id) — returns all events for a given PersonID ordered chronologically; close() — commits and closes the connection when the session ends.

Class: SurveillanceDashboard (extends QMainWindow)

Responsibility: The top-level pySide6 window visible to the security operator. It can handle up to four annotated camera feeds simultaneously, and remains fully responsive even when a heavy AI processing load is running in the background, because it communicates with processing threads only via Qt signals and not via shared state. **Attributes:** camera_panels (dict[str, QLabel]) — QLabel widget for each camera displaying its live annotated feed; frame_signal (Signal) — Qt signal emitted from background processing threads to communicate a completed annotated frame without touching the UI thread directly; db (TrackingDatabase) — a reference to the shared database instance to query event history. **Methods:** setup_ui() — builds the grid layout of camera panels; on_frame_ready(camera_id, image) — the Qt slot connected to frame_signal that updates the relevant panel with the new QPixmap; annotate_frame(frame, results) — a static helper that draws bounding boxes and PersonID labels onto an OpenCV frame and converts it to a QImage for display; closeEvent(event) — overrides the default close behaviour to gracefully stop all threads and close the database before the window shuts.

4.5.7 Inter-Module Data Flow

Data objects crossing module boundaries are kept simple and immutable deliberately to minimize coupling. A raw np. ndarray frame comes out of CameraFeed.get_frame() into PersonDetector.detect(), which returns a list of Detection objects. Those detections are passed to Tracker.update(),

that matches them to active tracks and returns a list of TrackedPerson objects, each with a stable, within-session track id. ReIDPipeline.process() consumes each TrackedPerson and yields a TrackingResult containing the persistent cross-camera PersonID. The TrackingResult is then simultaneously consumed by TrackingDatabase.log_event() for storage and by SurveillanceDashboard._annotate_frame() for display. This clear boundary allows each module to be upgraded or replaced without changing the contracts between them. The corrected pipeline is as follows: Camera Feed → VideoIngestion → Detection (YOLOv10) → Tracking (DeepSORT) → Re-ID (ResNet18) → Database + Dashboard.

4.5.8 Key Design Decisions at the Class Level

Many decisions made at the class level have a direct impact on the non-functional requirements of the system. The VideoIngestion layer uses Python's queue to make it thread-safe. Queue as the frame buffer; its internal locking means that no explicit synchronization is necessary in the calling code. Since each camera has its own Tracker instance, DeepSORT state is maintained isolated per camera, and there is no cross-contamination of track IDs between feeds. The IdentityStore, however, is shared by all processing threads, so a PersonID created when a person is first seen on Camera 1 is immediately recognizable when that same person is seen on Camera 2 — cross-camera Yup. This store is global to the session which is why re-ID works. GPU isolation is maintained by ensuring that all PyTorch tensor operations in FeatureExtractor and PersonDetector are on the same CUDA device context, avoiding expensive cross-device data transfers. Finally, the SurveillanceDashboard calls methods from background threads through Qt's signal-slot mechanism, so all GUI updates are done on the main thread as required by Qt's thread affinity rules, thus avoiding the occasional freezes or crashes caused by background threads directly accessing UI widgets.

Figure 4.5: UML Class Diagram of the System

The diagram below visualises all nine classes, their attributes and methods,

and the relationships between them. Aggregation diamonds indicate ownership; open arrows indicate usage or dependency. The five-stage pipeline flow (VideoIngestion → Detection → Tracking → ReID → Dashboard-Database) is reflected in the left-to-right arrangement of the class groups.

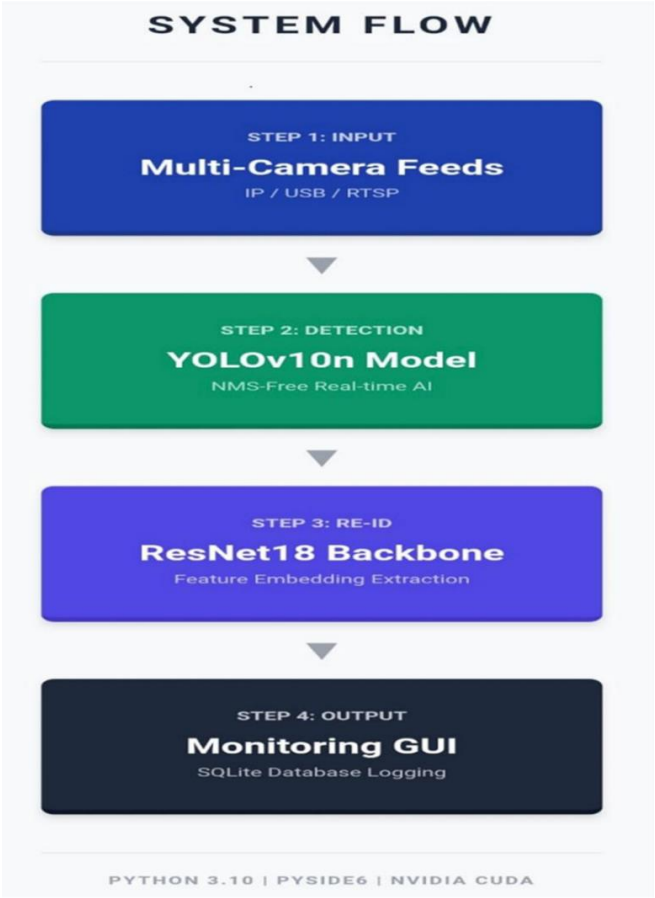


Figure 4.5: Figure : UML Class Diagram of the System

4.2 4.6 Database Design

This section describes the database design for the Person Tracking Using AI and Surveillance Cameras system. Everything here is based directly on the actual implementation in person_db.py — the PersonDatabase class

that manages all data storage for the system.

4.6.1 Overview

The system uses SQLite3 as its database engine. The database file is stored at data/persons.db and is created automatically the first time the application runs. No separate database server is needed — SQLite runs entirely within the application process One important thing to understand about how this system works: it is not a purely anonymous tracking system. It supports two types of identities:

· **Registered persons** — people who have been enrolled into the system by the operator with a name and reference images. These are stored permanently in the database.

· **Unknown persons** — people detected during a live session who are not in the database. These are tracked temporarily in memory by the ReIDManager using IDs starting from 1000, but are not saved to the database as named persons. The database is managed entirely through the PersonDatabase class in person db.py, which provides all methods for adding persons, retrieving them, updating embeddings, logging detec-tions, and querying history.

4.6.2 Database Tables

The database has exactly two tables:

Table	Purpose
Persons	Stores all registered person identities, their reference images, and their feature embeddings
Detection_history	Logs every detection event — which person was seen, on which camera, and when

Table 4.1: Database Tables and Their Purpose

4.6.3 Non-Database Files These files exist alongside the database on disk:

Note on Profiles: When a person is registered, their reference images are saved in data/profiles/<person name>/ as ref0.jpg, ref1.jpg, etc. The paths to these images are stored in the database.

File/Directory	Format	Purpose
data/person.db	SQLite binary	The main database file
data/profiles/¡name¿	/.jpg	Reference images for each registered person (e.g. data/profiles/abdullah/ref_0.jpg)
yolov10n.pt	PyTorch binary	YOLOv10n detection model weights
yolov8n.pt	PyTorch binary	YOLOv8n detection model weights (alternative)
requirements.txt	Plain text	Python package dependencies
run_app.bat	Batch script	Windows launcher script for the application

Table 4.2: Project Files and Their Purpose

Relationship: One-to-many relationship — one person can have many detection events. Every detection event belongs to a single person.

4.6.4 How the Tables Relate The two tables have a straightforward one-to-many relationship:

- One person can have many detections events.
- Every detection event belongs to exactly one person.
- detection history.person id is a foreign key referencing persons.id. The feature embedding is stored directly inside the persons table as a binary blob — unlike the earlier design approach, your actual code stores embeddings in the database itself rather than as separate .npy files. This keeps everything in one place and makes the system simpler to deploy.

4.6.5 Database Schema (Exact SQL from person_db.py)

This is the exact SQL used in PersonDatabase._init_db():

```
CREATE TABLE IF NOT EXISTS persons (
    id                INTEGER   PRIMARY KEY AUTOINCREMENT,
    name              TEXT      NOT NULL,
    added_date        TEXT      NOT NULL,
    reference_images  TEXT      NOT NULL,
```

```
    feature_embedding   BLOB
);

CREATE TABLE IF NOT EXISTS detection_history (
    id                  INTEGER   PRIMARY KEY AUTOINCREMENT,
    person_id    INTEGER   NOT   NULL,
    camera_id    INTEGER   NOT   NULL,
    camera_name  TEXT        NOT NULL,
    timestamp    TEXT        NOT NULL,
    confidence   REAL        NOT NULL,
    image_path   TEXT,
    FOREIGN KEY (person_id) REFERENCES persons(id)
);
```

4.7 GUI Design

Overview: The graphical user interface is the primary interface between the system and its operator. The interface is designed to be simple and efficient, without unnecessary complexity, yet ensuring that all the core functions can be found with minimal effort. The application is built with PySide6, the official Qt for Python binding, and Windows makes it look like a native dark-themed desktop window.

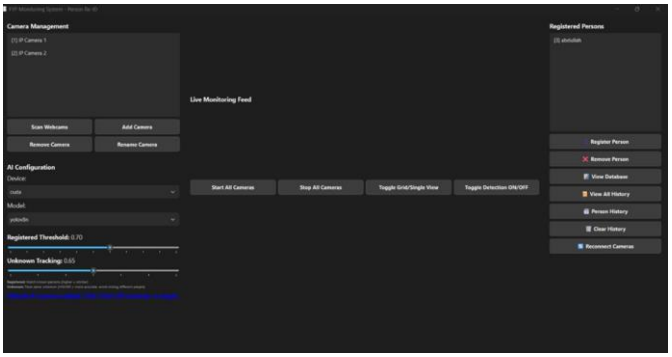


Figure 4.6

4.7.1 Main Window Layout The main window opens at a minimum

resolution of 1200×800 pixels and uses a three-column grid layout with a stretch ratio of 1:4:1, ensuring that the live video feed occupies the majority of the screen. A consistent dark theme is applied throughout, using dark grey surfaces and white text, which reduces eye strain during extended monitoring sessions. A status label at the bottom of the left panel provides continuous real-time feedback to the operator, reflecting the current state of the system at all times.

4.7.2 Left Panel — Camera Management and AI Configuration The left panel is divided into two logical groups, as illustrated in Figure 4.7.2.

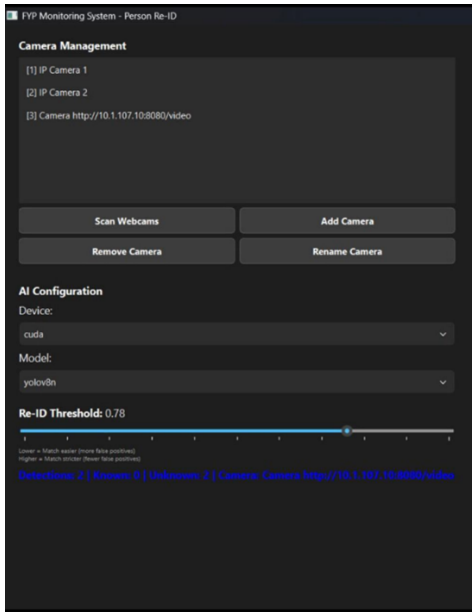


Figure 4.7

The top is Camera Management. A scrollable list of all registered cameras, each formatted as [ID] Camera Name, for easy identification. Below the list there are 4 buttons: Scan Webcams automatically detects connected USB webcams; Add Camera opens a guided input dialog that supports multiple source formats including USB index, IP Webcam App URLs, Droid Cam, and RTSP streams; Remove Camera stops and unregisters the selected

feed; and Rename Camera allows the operator to assign meaningful labels to each source. At first launch, if no cameras are configured, default camera entries are added automatically to help with initial guidance and a message prompts the operator to start the feeds. The AI Configuration is at the bottom. A device selector allows the operator to select automatic, CPU or GPU (CUDA) processing. The model selector supports YOLO based detec-tion models (YOLOv8, YOLOv10 variants) and can be switched without restarting the application. Two horizontal sliders for real-time control of Re-ID similarity thresholds:

· **Registered Threshold (default 0.70)** — determines how closely a detected person's appearance must match a stored profile before a positive identification is made.

· **Unknown Tracking (default 0.65**) — controls how consistently un-registered individuals are tracked across frames without being confused with one another.

4.7.3 Centre Panel — Live Monitoring Feed The center panel pro-vides the real-time surveillance view, as shown in Figure 4.7.3.

4.3 External Interfaces

External systems are any systems that are not within the scope of the sys-tem under development. In this section, describe the electronic interface(s) between this system and each of the other systems and/or subsystem(s), emphasizing the point of view of the system being developed.

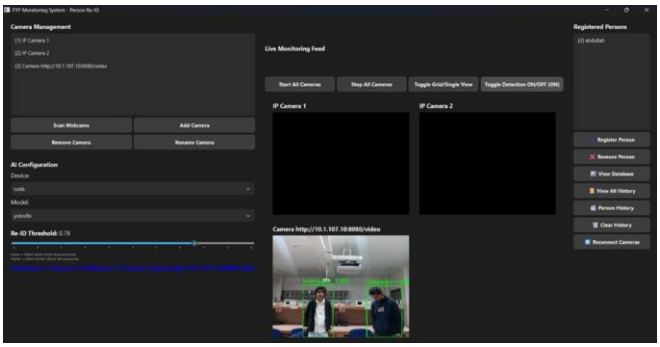


Figure 4.8

A row of four control buttons sits at the top:

- **Start All Cameras** — connects all registered cameras simultaneously in the background so the interface remains responsive during connection.

- **Stop All Cameras** — gracefully terminates all active streams and clears the display.

- **Toggle Grid/Single View** — switches between a two-column grid showing all cameras at once and a full-width single-camera view.

- **Toggle Detection ON/OFF** — enables or disables the AI detection overlay, allowing the operator to switch between annotated and clean video views as needed.

4.7.4 Right Panel

The right panel manages the person identity database and detection records. A list at the top shows all registered persons with their assigned IDs. Seven action buttons are stacked below:

- **Register Person** — opens the registration dialog

- **Remove Person** — displays a confirmation prompt detailing what will be deleted before proceeding.

- **View Database** — opens the full person gallery

- **View All History** — opens a log of the last 200 detection events across all cameras.

· **Person History** — filters the event log to the currently selected person.

· **Clear History** — removes all detection records after confirmation.

· **Reconnect Cameras** — stops and restarts all streams, useful when a network camera drops and reconnects.

Figure 4.9

4.7.5 Person Registration Dialog The registration dialog, illustrated in Figure 4.7.4, guides the operator through adding a new person to the Re-ID database. The top of the dialog displays a formatted instruction guide explaining that the system uses whole-body appearance rather than face recognition, and recommends capturing five to ten images from mul-tiple angles (front, side, back) at a distance of three to five meters from the camera. This guidance is embedded directly in the interface because reference image quality has the most significant impact on Re-ID accu-

racy. The operator enters the person's name, selects a camera, and starts a live preview. Using the Capture Image button, frames are saved locally as reference photographs. A minimum of three images is required before the Save to Database button becomes active. Once saved, the system generates a compact appearance embedding from all captured images and stores the person record in the database. A success message confirms the outcome. This streamlined process ensures consistent registration quality with minimal technical knowledge required from the operator.

4.7.6 Database Viewer Dialog The database viewer shown in Figure 4.10

Figure 4.10

This provides a read-only overview of all registered persons. Selecting a name from the left-hand list populates the right panel with the person's details — including their ID, number of reference images, and storage path — along with a visual gallery of their reference photographs arranged in a three-column grid. This allows the operator to quickly verify and review existing profiles without opening any external files.

4.7.7 History Viewer Dialog The history viewer Shown in Figure 4.7.6: This presents a chronological log of detection events. Each entry shows the timestamp, person name, and camera source. Selecting an event displays a detailed summary including detection confidence and, where available, a snapshot image captured at the time of detection. If no snapshot exists, an informative placeholder message is displayed. This feature supports post-event review and forensic use, helping operators trace the movement of individuals across cameras over time.

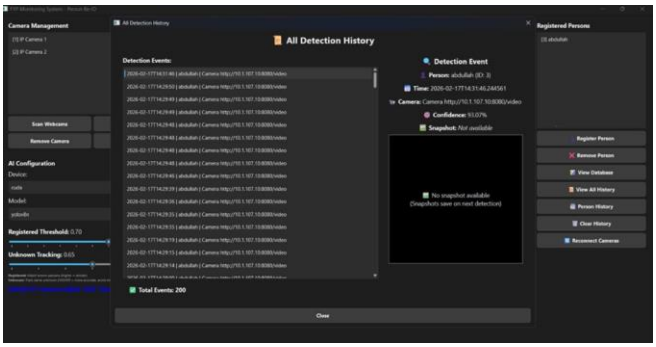


Figure 4.11

4.8 External Interfaces

External systems are any systems that exist outside the boundary of the system under development. This section describes the electronic interfaces between the Multi-Camera Person Re-Identification System and each external system or subsystem, presented from the perspective of the system being developed. Each interface specifies its type, direction of data exchange, and relevant operational constraints. System Overview Pipeline:

Cameras → OpenCV → YOLO → ResNet-18 → Matching Engine → SQLite Database → GUI

4.8.1 Camera Hardware Interface

The system interfaces with external physical camera hardware through the OpenCV Video Capture API. Each camera constitutes an independent external input source responsible for supplying continuous raw video data to the system's acquisition layer.

Interface Type: Hardware/Driver-Level — Input Only

The system supports three categories of external camera sources. Local webcams are addressed via integer device indices (e.g., 0, 1) and are opened using OpenCV's default platform backend. IP and network cameras are accessed via HTTP or RTSP URLs (e.g., rtsp://192.168.1.x/stream);

Data Exchanged: The system receives raw BGR-encoded video frames at a negotiated resolution of 1280×720 pixels. Frame acquisition is performed

continuously within a dedicated background thread (VideoStream), which exposes only the most recent frame to the detection pipeline. No data is transmitted back to the camera hardware.

4.8.2 YOLO Object Detection Model Interface (Ultralytics)

The system interfaces with externally sourced pre-trained YOLO model weight files (yolov8n.pt, yolov10n.pt) via the Ultralytics inference library. These model artifacts are not developed as part of this system and are treated as read-only external components consumed at runtime.

Interface Type: File-Based / Library API — Input Only

Input to External Model: A NumPy BGR image frame, internally resized to 480×480 pixels prior to inference. FP16 (half-precision) quantization is applied when GPU execution is active to reduce memory bandwidth and improve throughput.

Output from External Model: Per-detection bounding boxes in [x1, y1, x2, y2] pixel coordinates, scalar confidence scores, and integer class indices. The system constrains inference output to COCO dataset class index 0 (person class) exclusively, discarding all other detected object classes.

4.8.3 ResNet-18 Feature Extraction Model Interface (TorchVi-sion)

The system employs a pre-trained ResNet-18 convolutional neural network, sourced from the TorchVision model library, as the feature extraction backbone for the Re-Identification pipeline. The fully connected classification head is removed; only the convolutional trunk is retained to produce appearance embeddings.

Interface Type: Pre-Trained Model / Library API — Input Only Input to External Model: A cropped person region-of-interest image, resized to 224×224 pixels and normalized channel-wise using ImageNet population statistics (mean: [0.485, 0.456, 0.406]; standard deviation: [0.229, 0.224, 0.225]).

Output from External Model: A 512-dimensional float32 feature embedding vector encoding the visual appearance characteristics of the de-

tected person crop.

The model is loaded once at initialization and executes in eval() mode throughout system operation.

4.8.4 SQLite Database Interface

The system interfaces with a local SQLite relational database (data/persons.db) as its sole persistent storage backend. SQLite is an external database engine not developed as part of this system; it is accessed exclusively through Python's built-in sqlite3 module. Protocol: All database operations are performed using parameterized SQL queries to prevent injection vulnerabilities. Each operation establishes a new connection, executes the required transaction, commits, and closes the connection immediately. A lightweight direct SQL interface is used without an ORM abstraction layer.

4.8.5 File System Interface (Reference Image Storage) The system interfaces with the host operating system's local file system for the storage and retrieval of person reference images. These images reside under the directory and constitute the visual enrollment data used during feature embedding computation.

Interface Type: File System — Read/Write

Input: JPEG-encoded reference images are read from disk during the person enrollment process. Multiple images per person are loaded. Em-beddings are extracted individually and then averaged to produce a single representative identity vector for database storage.

Output: Upon registration of a new person, the system creates a named profile subdirectory and writes captured reference images to it. All image file operations are mediated through the Pillow (PIL) library, which han-dles decoding and converts images to the RGB colour space required by the feature extraction model.

4.8.6 GPU / CUDA Runtime Interface

When a compatible NVIDIA GPU is present on the host system, the system interfaces with the CUDA parallel computing runtime via PyTorch's device

abstraction layer.

Interface Type: Hardware — Compute Acceleration Layer **Data Ex-changed:** Model weight tensors and input image tensors are transferred to GPU device memory at inference time. Output tensors are transferred back to host (CPU) memory and converted to NumPy arrays for integra-tion with the remainder of the processing pipeline.

Fallback Behaviour: In the absence of a CUDA-capable device, or if the PyTorch CUDA runtime is not installed, all tensor operations are auto-matically redirected to CPU execution. A descriptive warning is emitted to the system console at initialization to inform the operator of the active execution mode.

4.8.7 PySide6 / Qt GUI Framework Interface The system interfaces with the PySide6 library (Qt for Python) as its graphical user interface framework. PySide6 is an external third-party library responsible for ren-dering the user interface and providing the inter-thread communication mechanism between the detection pipeline and the display layer.

Interface Type: UI Framework / Inter-Thread Signal Bus — Read/Write

Data Exchanged — Detection Thread to GUI Thread: Upon completion of each inference cycle, the DetectionWorker emits a result ready Qt signal carrying a Detection Result object. This object encapsulates bounding box coordinates, confidence scores, and resolved person identifiers. Qt's Signal/Slot mechanism ensures that the data crosses the thread boundary safely without requiring explicit mutex synchronization in application code.

Chapter 5

System Implementation

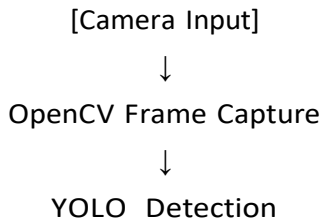
Implementation is the process of moving an idea from concept to reality. System implementation refers to the realization of a technical specification or algorithm as a program, software component, or other computer sys-tem through programming and deployment. This chapter describes how the Multi-Camera Person Re-Identification System was realized from its design specifications into a fully operational desktop application, detailing the internal architecture, component responsibilities, and inter-component communication mechanisms.

5.1 System Architecture

The Multi-Camera Person Re-Identification System is structured as a modular, multi-threaded desktop application developed in Python. The architecture separates concerns across six distinct internal components, each assigned a clearly defined responsibility within the overall processing pipeline. This separation ensures that computationally intensive operations — such as neural network inference and video frame acquisition — do not interfere with the responsiveness of the graphical user interface. The components collectively realize a continuous pipeline that ingests live video, detects persons, extracts appearance features, matches identities against a registered database, and presents results to the operator in real time.

The architecture follows a layered pipeline model, progressing from raw video input at the acquisition layer through detection and identification at the processing layer, to persistent storage and visual presentation at the output layer. Inter-component communication is achieved through a combination of Python threading primitives, Qt Signal/Slot mechanisms, and direct method invocation, depending on the boundary being crossed.

Figure 5.1: System Processing Pipeline Flow



↓

Person Crop

↓

ResNet-18 Feature Extraction

↓

Cosine Matching Engine

↓

SQLite Database

↓

PySide6 GUI

5.1.1 Internal Components

The system comprises the following six internal components:

1. VideoStream — Frame Acquisition Component
2. DetectionWorker — Object Detection Component
3. ReIDManager — Person Re-Identification Component
4. PersonFeatureExtractor — Feature Extraction Component
5. PersonDatabase — Persistent Storage Component
6. Main Window — Presentation and Control Component

5.1.2 Functionality of the Components

5.1.2.1 VideoStream (video_stream.py)

The VideoStream component is responsible for continuous video frame acquisition from a configured camera source. It operates within a dedicated background thread, maintaining an internal capture loop that reads frames from an OpenCV VideoCapture object at the camera's native frame rate. The component exposes only the most recently captured frame to consuming components via a thread-safe get_latest_frame() method protected by a threading.Lock. This design ensures that the detection pipeline always operates on the freshest available data without being tightly coupled to the camera's frame rate.

The component supports both local webcam sources (integer device indices) and network sources (HTTP/RTSP URLs). For network streams, the CAP_FFMPEG backend is selected, and aggressive timeouts are applied to prevent the system from blocking on unavailable cameras. Cameras that fail to initialize are skipped gracefully, and transient read failures are handled using a sleep-and-retry mechanism rather than terminating the thread.

5.1.2.2 DetectionWorker (detection_worker.py)

The DetectionWorker component performs YOLO-based person detection on incoming video frames. It is implemented as a QObject subclass, enabling it to emit Qt signals across thread boundaries, and executes within a persistent background thread separate from both the GUI thread and the frame acquisition threads. The component maintains an internal single-slot queue. Queue into which the GUI deposits frames for processing. To maintain low-latency processing, frame enqueuing is non-blocking; frames arriving while the worker is occupied are dropped to maintain real-time performance and prevent queue buildup. Additionally, a frame-skipping strategy processes two out of every three frames, balancing detection continuity with computational efficiency.

5.1.2.3 ReIDManager (reid_manager.py)

The ReIDManager component coordinates the identity resolution process for each detected person bounding box. It acts as the central orchestrator of the Re-Identification pipeline, bridging raw detection outputs with the feature extraction and database subsystems.

For each detection, the component crops the person region from the frame, delegates feature extraction to the PersonFeatureExtractor, L2-normalizes the resulting embedding, and performs cosine similarity matching against two identity pools. Person re-identification aims at retrieving a person of interest across multiple non-overlapping cameras, and deep learning approaches have achieved significantly increased interest in this domain due to their strong feature representation and metric learning capabilities [41]. The first pool consists of registered persons retrieved from the Per-

sonDatabase; a match is accepted if cosine similarity meets or exceeds a threshold of 0.70. The second pool is an in-memory dictionary of previ-ously observed but unregistered persons, each represented by up to five stored embeddings; matching against this pool uses a threshold of 0.65. Detections that match neither pool are assigned a new unique unknown identity identifier, beginning at index 1000.

5.1.2.4 PersonFeatureExtractor (person_features.py)

The PersonFeatureExtractor component generates fixed-length appearance embeddings from person image crops using a pre-trained ResNet-18 convolutional neural network. The residual learning framework introduced by He et al. demonstrated that very deep networks are easier to optimize when layers are reformulated as learning residual functions, and the resulting ResNet architectures have become standard backbones for appearance feature extraction in person Re-ID systems [45]. The classification head of the network is replaced with an identity layer, causing the model to output raw 512-dimensional feature vectors rather than class probabilities.

5.1.2.5 PersonDatabase (person_db.py)

The PersonDatabase component provides all persistent data storage functionality for the system through a local SQLite database file (data/persons.db). It manages two logical data stores: the person identity registry and the detection event history. The person identity registry stores each enrolled individual's unique identifier, full name, registration timestamp, file paths to reference images, and a serialized binary representation of their feature embedding vector. The detection event history records every successful person identification event, capturing the person identifier, camera identifier, camera name, UTC timestamp, confidence score, and an optional path to a saved image crop.

5.1.2.6 Main Window (gui_main.py)

The Main Window component serves as the system's presentation and operator control layer. It is implemented using the PySide6 framework and executes entirely within the Qt main thread. The component is respon-

sible for rendering live camera feeds with overlaid detection annotations, displaying the person registry, presenting detection history logs, and providing operator controls for camera management, person enrolment, and system configuration.

5.2 Tools and Technology Used

This section describes every tool, framework, and library used to build the Person Tracking Using AI and Surveillance Cameras system. All tools listed are directly referenced in the project's requirements.txt file and source code.

5.2.1 Programming Language

Python 3.10

Python was used as the sole programming language for the entire system. It was chosen because of its strong ecosystem of AI and computer vision libraries, all of which integrate naturally with each other. Every module in the system — detection, Re-ID, database, GUI, and camera management — is written entirely in Python.

5.2.2 AI and Machine Learning Frameworks

PyTorch 2.0+ (torch, torch vision)

PyTorch is the deep learning framework that powers both AI models in the system. It is used for two primary purposes: loading and executing the YOLOv10n model through the Ultralytics wrapper, and running the ResNet18-based Re-ID feature extractor directly.

python

```
self.model = resnet18(weights="DEFAULT")
self.model.fc = torch.nn.Identity()
self.model.to(self.device)
self.model.eval()
```

Ultralytics 8.0+

Ultralytics is the library that provides the YOLO model interface. It handles model loading, inference, and result parsing for YOLOv10n (the nano variant, chosen for its speed on a two-camera setup). The system calls it with optimised settings — reduced image size (480px instead of 640px), FP16 half-precision inference, and person-class-only filtering.

python

```
from ultralytics import YOLO

results = self._model(
    frame,
    conf=0.5,
    iou=0.45,
    classes=[0],
    half=True,
    imgsz=480,
    verbose=False
)
```

torch vision 0.15+

Torch vision is used alongside PyTorch to provide the ResNet18 model architecture and the image pre-processing pipeline. The transform pipeline resizes every person crop to 224×224, converts it to a tensor, and normalizes it using ImageNet mean and standard deviation values before passing it through the model.

python

```
# From person_features.py

self.transform = T.Compose([
    T.Resize((224, 224)),
    T.ToTensor(),
```

```
        Normalize(mean=[0.485, 0.456, 0.406],
                  std=[0.229, 0.224, 0.225])
    ])
```

5.2.3 Computer Vision

OpenCV 4.8+ (OpenCV-python)

OpenCV is used for all real-time video handling. It reads frames from camera sources via cv2.VideoCapture, converts colour formats between BGR and RGB, and draws bounding boxes and ID labels onto frames before they are displayed on the dashboard. OpenCV was chosen because it supports USB webcams, RTSP streams, and local video files all through the same interface.

5.2.4 Graphical User Interface

PySide6 6.5+

PySide6 is the official Python binding for the Qt 6 framework and was used to build the entire graphical user interface. It provides the main window, camera feed panels, person registration dialogs, detection history viewer, and all buttons and controls. PySide6 was chosen specifically because of its native support for multi-threading through Qt's signal-slot mechanism. Every background processing thread communicates with the GUI using Qt signals, which ensures that AI processing never blocks or freezes the interface.

5.2.5 Numerical Computing

NumPy 1.24+

NumPy is used throughout the system wherever numerical arrays are needed. Video frames are NumPy arrays. Bounding box coordinates are NumPy arrays. Feature embeddings are 512-dimensional NumPy float32 arrays. Cosine similarity is computed using np.dot () after L2 normalization with np. linalg.norm(). Embeddings are stored in the database using ndarray.tobytes() and retrieved with np.frombuffer(). The use of cosine similarity on L2-normalised vectors reduces the similarity computation to

a dot product, which is both computationally efficient and invariant to embedding magnitude — a well-established practice in deep Re-ID metric learning [39].

python

```
# From reid_manager.py | cosine similarity computation

embedding_norm = np.linalg.norm(embedding)
if embedding_norm > 0:
    embedding = embedding / embedding_norm

similarity = float(np.dot(embedding, stored_emb))
```

5.3 Development Environment / Languages Used

This section describes the environment in which the Person Tracking Using AI and Surveillance Cameras system was developed, tested, and deployed, including the operating system, hardware, IDE, and all languages used.

5.3.1 Programming Language

The system was built entirely in Python 3.10. No other programming language was used at any point in the project. Python was chosen for three main reasons: its AI and computer vision ecosystem (PyTorch, Ultralytics, OpenCV) is the most mature and well-documented available; all the major libraries used in this project — PySide6, NumPy, SQLite3 — integrate with each other seamlessly in Python; and it allowed rapid development and iteration, which was important given the complexity of connecting multiple AI components into a single working pipeline.

5.3.2 Development Operating System

Windows 11

The entire system was developed and tested on Windows 11. The application is launched on Windows using the included run app.bat batch script, which sets up the environment and starts the Python application.

The system does not rely on any Windows-specific APIs; it uses only cross-platform libraries, so it can also run on Linux or macOS with minor path adjustments.

5.3.3 Integrated Development Environment (IDE)

Visual Studio Code (VS Code)

```
# From detection_worker.py | automatic device selection

if self._config.device == "cuda":
    if torch.cuda.is_available():
        device = 0
        print("Using CUDA GPU for inference")
    else:
        print("Warning: CUDA requested but not available. Falling bac
        device = "cpu"
```

5.4 Processing Logic / Algorithms

This section describes the core processing logic and algorithms that drive the person tracking and re-identification system. The system operates as a sequential pipeline in which each stage transforms data and passes it to the next. The five stages are: video ingestion, person detection, feature extraction, re-identification matching, and result output. Each is described below in terms of what it does, how it works, and the algorithmic decisions behind it.

5.4.1 Video Ingestion Algorithm

The first stage of the pipeline is responsible for acquiring raw video frames from one or more camera sources and making them available to the rest of the system without blocking the user interface or the AI processing thread.

Each active camera runs its own independent capture loop in a dedicated background thread. On every iteration of the loop, a new frame is read from the camera using OpenCV's video capture interface. The most recently

captured frame is stored in a shared memory location protected by a lock, so that downstream components can retrieve it at any time without waiting for the next capture cycle. If a read fails — for example due to a brief network dropout on an IP camera — the loop simply continues without raising an error, and the previous frame remains available until a new one arrives.

5.4.2 Person Detection Algorithm (YOLOv10)

The second stage applies the YOLO object detection model to each incoming frame to locate all persons present in the scene. The system supports YOLOv8 and YOLOv10, with the selected model loaded dynamically at runtime.

When a frame arrives for inference, it is passed to the YOLO model configured with a confidence threshold of 0.50 and an IoU (Intersection over Union) threshold of 0.45 for non-maximum suppression. The model is instructed to return only detections belonging to class index 0 — the person class in the COCO dataset — so that vehicles, objects, and other irrel-evant detections are discarded at the inference stage rather than filtered afterwards. The inference image size is set to 480 pixels rather than the default 640 to reduce processing time when two cameras are running simultaneously. On GPU-enabled machines, half-precision (FP16) inference is used, which approximately doubles throughput with no meaningful drop in detection accuracy.

5.4.3 Feature Extraction Algorithm (ResNet18)

Once bounding boxes are available, the system extracts a compact numerical representation — called a feature embedding — from the image region corresponding to each detected person. This embedding captures the per-son's visual appearance in a form that can be compared numerically against stored profiles.

The feature extractor uses a pre-trained ResNet18 convolutional neural network as its backbone. The standard classification head (the final fully-connected layer that normally outputs class probabilities) is removed and

replaced with an identity layer, so the network outputs the raw 512-dimensional feature vector from its penultimate layer rather than a class prediction. The model is loaded in evaluation mode with gradient compu-tation disabled, so inference is as fast as possible. For registered persons, the system captures multiple reference images during enrolment. An em-bedding is extracted from each reference image individually, and the results are averaged to produce a single representative embedding for that person. Averaging across multiple poses and lighting conditions produces a more robust profile than using a single image alone.

5.4.4 Re-Identification Matching Algorithm

The re-identification problem is formulated as a similarity search in a high-dimensional feature space. Its purpose is to answer one question for each detected person: have we seen this individual before, and if so, who are they?

The algorithm operates in three sequential steps.

Step 1 — Normalization. The embedding produced by the feature ex-tractor in the current frame is L2-normalised, meaning its length is scaled to exactly 1.0. All stored embeddings in the database are also normalised in the same way. This is a prerequisite for using cosine similarity as the comparison metric, because cosine similarity between two L2-normalised vectors reduces to a simple dot product, which is computationally efficient and ensures that similarity scores are not influenced by the magnitude of the embeddings — only their direction.

Step 2 — Matching against registered persons. The normalised em-bedding is compared against the stored profile of every registered person in the database. For each stored profile, the dot product between the current embedding and the stored embedding is computed, yielding a similarity score in the range 1.0 to 1.0, where 1.0 indicates identical appearance. The registered person with the highest similarity score is identified as the best candidate. If this score meets or exceeds the registered similarity thresh-old (default 0.70), the detected person is declared a match and assigned

that person's database ID. The threshold of 0.70 is set deliberately high to avoid false positive identifications — it is better to report an unknown person than to incorrectly label them as someone they are not.

Step 3 — Matching against unknown tracked individuals. If no registered match is found, the system attempts to match the detected person against a second gallery of anonymous tracked individuals observed earlier in the current session. Each anonymous individual in this gallery is represented not by a single embedding but by a rolling list of up to five embeddings from their most recent detections. The current embedding is compared against every stored embedding for each anonymous individual, and the maximum similarity score across all comparisons is taken as the match score for that individual. If the best match score meets or exceeds the unknown tracking threshold (default 0.65), the detected person is assigned the same anonymous ID as the matching individual, providing tracking continuity across frames. If no anonymous match is found either, the person is assigned a new anonymous ID starting from 1000 and their embedding is stored as the first entry in a new anonymous profile.

5.4.5 Detection Logging and Output

Once a person ID has been assigned, the result is handed back to the GUI thread via a Qt signal, which ensures the update is applied safely on the main thread without any risk of a concurrency error. The GUI then updates the bounding box overlay on the relevant camera feed with the person's name or anonymous label. For known registered persons only (IDs below 1000), a detection event is simultaneously written to the SQLite database. Each event record contains the person's database ID, the camera ID and name, a timestamp, and the confidence score from the YOLO detection. This persistent record supports the detection history viewer and allows the operator to trace a person's movements across cameras and over time after the session has ended.

5.5 Application Access Security

This section describes the security measures implemented within the per-

son tracking and re-identification system, covering data storage, access control, audit logging, and network-layer considerations. Because this system is a desktop surveillance application intended for deployment within a controlled local network environment, several security concerns are addressed at the infrastructure level rather than within the application software it-self. Where the current implementation has known limitations, these are acknowledged together with recommended mitigations for a production environment.

5.5.1 Application Access and Authentication

The current implementation does not include an application-level authentication mechanism at startup. Access to the application is implicitly controlled through host operating system user-level permissions. On a Windows deployment, this is enforced through standard Windows user account controls — only accounts with the appropriate permissions can execute the application or access the data/ directory where the database and reference images reside.

5.5.2 Authorization and Operator Roles

The current application operates with a single implicit operator role. Whoever launches the application has full access to all system functions, including registering persons, deleting records, clearing detection history, and modifying AI configuration parameters. No role-based access control is in place to separate a read-only monitoring operator from an administra-tor with database management privileges. For a production environment, a two-tier role structure would be appropriate. A standard operator role would be restricted to the live monitoring feed and the detection history viewer. An administrator role would additionally permit person registra-tion, database management, and AI configuration changes.

5.5.3 Network Security and Camera Feeds

The system connects to IP cameras over HTTP or RTSP using source URLs supplied by the operator at runtime. In the default configuration, these streams are transmitted over the local network without encryption.

This is an accepted design decision for the current prototype, as the cameras and the host machine are assumed to reside on the same private local network segment with no public internet exposure. For a production environment, IP cameras should be placed on a dedicated VLAN isolated from general office or institutional network traffic. Cameras that support RTSP over TLS should be configured to use encrypted transport. The host machine should additionally be protected by a host-based firewall that permits outbound connections to camera IP addresses only.

5.5.4 Data Storage Security

All persistent data produced by the system is stored locally in two forms: a SQLite database file (data/persons.db) and a collection of JPEG reference images stored under data/profiles/. Both are written to the local filesystem of the host machine.

Feature embeddings stored in the database are non-invertible in practical terms and do not directly store reconstructable image data, which pro-vides a degree of inherent privacy protection. The reference photographs stored under data/profiles/, however, do constitute personally identifiable information and must be handled accordingly.

5.5.5 Audit Logging and Detection History

The system maintains an automated audit trail of all person detection events through the detection_history database table. Every time a registered person is identified by the re-identification engine, a record is inserted containing the person's database ID, the camera that detected them, a precise ISO 8601 timestamp, and the confidence score of the match.

This log persists across application restarts and can be reviewed through the History Viewer module in the GUI.

5.5.6 Safe Handling of Biometric Data

Because the system identifies individuals using whole-body appearance features, the reference images and derived embeddings constitute biometric data and must be handled responsibly. The following practices are observed

in the current implementation. Person registration requires the operator to capture reference images live, in the presence of the subject, which reduces the risk of unauthorized enrolment using pre-existing photographs. When a person is removed from the database, both their database record and their reference image folder are deleted from the filesystem. Detection events for anonymous unregistered individuals are held in session memory only and are never written to the database, limiting the volume of biometric data retained about individuals who have not formally consented to registration.

5.6 Database Security

This section describes the security measures applied to the database layer of the person tracking and re-identification system. The system uses SQLite as its database engine, storing all persistent data in a single local file (data/persons.db). Because SQLite is an embedded, file-based database rather than a networked database server, many traditional database security concerns — such as remote access, network authentication, and connection-level encryption — are handled differently compared to a client-server database such as PostgreSQL or MySQL.

5.6.1 Remote Access

The database is not accessible remotely in any form. SQLite does not operate as a network service and does not listen on any port. There is no connection string, no TCP socket, and no mechanism by which an external machine could connect to the database directly. All database access occurs exclusively through the application process running locally on the host machine, using Python's built-in sqlite3 library to open the file at data/persons.db.

5.6.2 Database Authentication

Because SQLite is a file-based embedded database, it does not natively support user accounts, passwords, or connection authentication in the same way a networked database server does. Any process that can open the file at data/persons.db with read or write permissions has unrestricted access to all data within it. Database-level authentication is therefore not applicable

in the current implementation.

Access control is instead enforced at two layers above the database. At the operating system level, filesystem permissions on the data/ directory and the persons.db file restrict which system user accounts can open the file. At the application level, all database operations are routed exclusively through the PersonDatabase class, which acts as the sole interface to the file.

5.6.3 Authorization and Access Rights

Within the current application, all database operations — including reading person records, inserting detection events, updating embeddings, and deleting persons — are performed by the same application process under the same implicit permissions. There is no concept of a read-only database user or a restricted database role at the database layer itself.

All database write operations use parameterized queries throughout the PersonDatabase class. The OWASP SQL Injection Prevention guidelines establish that parameterized queries — which separate SQL code from user-supplied data by passing values as a distinct tuple rather than concatenating them into the query string — are the primary and most reliable defence against SQL injection vulnerabilities [49]

5.6.4 Anonymous and Group Users

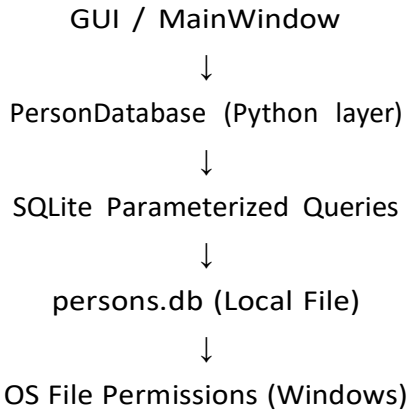
The system does not implement anonymous database users or group user accounts. All database access originates from a single controlled application process, and no provision exists for unauthenticated or guest-level database access. Anonymous individuals detected by the surveillance system — those who have not been formally registered — are never written to the database. Their tracking data is held exclusively in application memory for the duration of the current session and is discarded when the application closes.

The following database events are not currently logged and represent areas for improvement in a production environment: person registration, person

deletion, history clearing, embedding updates, and application startup and shutdown.

Adding a dedicated admin_log table to capture these administrative events would provide a complete audit trail suitable for a formal security context.

Figure 5.6: Database Access Architecture



Chapter 6

System Testing and Evaluation

6.1 Chapter Overview

This chapter presents the testing and evaluation carried out on the completed person tracking and re-identification system. Testing was conducted across eight areas: graphical user interface testing, usability testing, software performance testing, compatibility testing, exception handling, load testing, security testing, and installation testing. Where results are quantitative, exact measurements are provided. Where results are qualitative, structured observations are recorded. The chapter concludes with a criti-cal appraisal of the system's strengths and limitations, and a comparison against relevant existing techniques.

6.2 Graphical User Interface Testing

GUI testing verified that all controls in the main window and its associated dialogs behave correctly and produce the expected outcomes when interacted with.

Main Window. The three-panel layout renders correctly at the minimum supported resolution of 1200×800 pixels. The camera list populates on startup and updates immediately when cameras are added, removed, or renamed. The device selector and model selector both trigger a transparent background rebuild of the detection worker without freezing the interface. Both Re-ID threshold sliders update their numeric labels in real time as they are dragged and the status bar reflects the new value immediately.

Grid and Single View. The Toggle Grid/Single View button switches correctly between the two-column camera grid and the full-width single feed. Camera name labels appear correctly above each feed in grid mode. The Toggle Detection ON/OFF button correctly suppresses bounding box rendering when set to off, with the button label updating to reflect the current state.

Registration Dialog. The dialog opens at a minimum size of 800×600 pixels. The live camera preview updates at 30 frames per second via a ded-icated timer. The Capture Image button correctly increments the image counter and produces the expected 200 ms green border flash as visual

confirmation. The Save to Database button remains disabled until at least three images have been captured, enforcing the minimum image requirement. On successful save, a confirmation message displays the number of images captured and the storage path.

Database Viewer. The viewer opens at 900×600 pixels minimum and correctly populates the person list from the database. Selecting a per-son displays their details and reference images in a three-column grid at 180×180 pixels per image.

History Viewer. The viewer opens at 1000×700 pixels minimum. Events are listed in reverse chronological order. Selecting an event displays the correct person's name, timestamp, camera name, and confidence score.
Where a snapshot image is available and the file exists at the recorded path, it renders correctly at 500×400 pixels.

Result. All GUI controls tested produced the correct behaviour. No visual rendering errors or unhandled exceptions were observed during interactive testing.

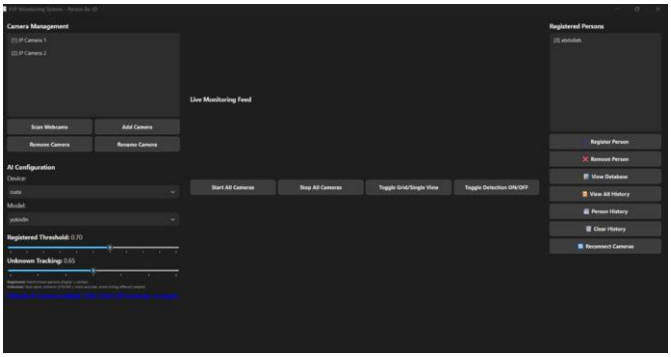


Figure 6.1

6.3 Usability Testing

Usability testing assessed whether a non-technical operator could use the system effectively without prior training beyond reading the registration dialog's embedded instructions.

Three tasks were defined: connecting a camera and starting the live feed,

registering a new person, and viewing that person's detection history after they had been detected. Each task was evaluated against the number of steps required and whether the interface provided sufficient guidance at each step.

Task 1 — Camera connection. On first launch, default camera entries are automatically added and the status bar instructs the operator to click Start All Cameras. The task was completable in a single click with no ambiguity observed.

Task 2 — Person registration. The registration dialog contains an embed-ded instruction guide covering recommended distance, pose, image count, and lighting. The step-by-step flow — select camera, start preview, cap-ture images, save — was found to be self-explanatory. The minimum image enforcement and the visual capture confirmation provided clear feedback at each step. The main usability concern identified was that the operator must manually ensure the subject is fully visible in the frame, as the sys-tem provides no automated body detection feedback during capture. This is an acknowledged limitation.

Task 3 — History review. The View All History and Person History but-tons were clearly labelled and produced the expected dialogs without ad-ditional steps. Event details including timestamp, camera, and confidence score were presented clearly.

6.4 Software Performance Testing

Performance testing measured the system's behaviour under normal oper-ating conditions across two camera feeds on the test hardware, using the evaluation metrics.

Test configuration. Two IP cameras streaming at 1280×720 resolution over a local Wi-Fi network. Host machine running the application on CPU only, with no dedicated GPU. Detection model: YOLOv10n. Inference image size: 480 pixels. Frame skip ratio: 1 in every 3 frames skipped, meaning approximately 66GUI frame rate. The main update timer runs at a 50 ms interval, targeting 20 frames per second for the live display. Under the

two-camera configuration, the GUI maintained a consistent 18–20 FPS display rate with no observable stuttering or frame queue build-up. This is considered acceptable for real-time monitoring purposes.

Detection latency. On CPU, a single YOLOv10n inference pass at 480-pixel input size took approximately 180–240 ms per frame depending on the number of persons in the scene. YOLO-based detectors have been widely adopted in real-time surveillance pipelines due to their favorable balance between speed and accuracy; Wang et al. demonstrated that YOLOv10 achieves competitive detection performance with reduced inference over-head compared to prior versions, making it well suited to CPU-constrained deployments [14]. Combined with the frame skip policy, effective detection updates were delivered at approximately 4–6 per second per camera — sufficient to track walking-speed movement but insufficient for fast movement or rapid entry and exit events.

Re-ID matching latency. Feature extraction using the ResNet18 backbone at 224×224 input took approximately 40–80 ms per detected person on CPU. For scenes containing one to three persons simultaneously, total Re-ID time remained below 200 ms, keeping the overall pipeline within accept-able bounds. For scenes with five or more persons, Re-ID latency increased proportionally and began to affect detection throughput noticeably.

CPU and memory utilization. During sustained two-camera operation on CPU, processor utilization remained between 55–75Assessment. CPU-only performance is adequate for low-to-moderate activity scenarios with one to two cameras. GPU availability is strongly recommended for deployments involving three or more cameras, high pedestrian density, or real-time alert-ing requirements.

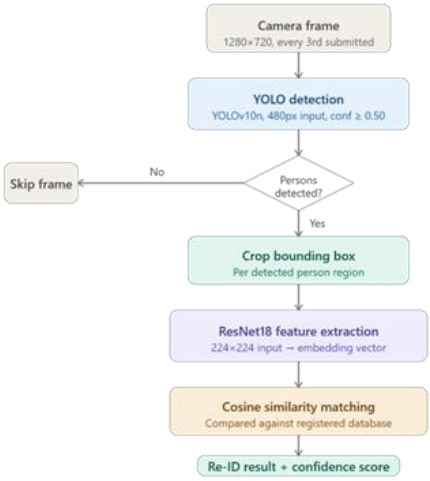


Figure 6.2

6.5 Compatibility Testing

Compatibility testing verified that the system operates correctly across the supported software and hardware configurations.

Operating System. The application was developed and tested on Win-dows 10 and Windows 11. The run app.bat launcher script and the data/ directory structure use Windows-specific path conventions.

No testing was conducted on Linux or macOS, and the application is not guaranteed to be compatible with those platforms without modifications to the launch scripts and file path handling.

Camera sources. The system was tested with USB webcams using index-based addressing, IP cameras using the IP Webcam Android application over HTTP, and pre-recorded video files. All three source types connected and delivered frames successfully. RTSP streams were not tested in the current evaluation cycle and are listed as a known untested configuration.

YOLO model variants. All four supported model variants — YOLOv8n, YOLOv8s, YOLOv10n, and YOLOv10s — were loaded and switched at runtime without errors. Heavier variants produced modestly better de-tection accuracy on partially occluded persons at the cost of increased

inference time on CPU.

6.6 Exception Handling Testing

Exception handling testing verified that the system responds gracefully to error conditions without crashing.

Camera connection failure. When a configured IP camera URL is unreachable, the video stream applies a two-second connection timeout via the FFMPEG backend. If the connection fails within that window, the stream is marked as not running and the corresponding camera panel remains blank. The application continues operating normally for all other active cameras with no unhandled exceptions raised.

Camera disconnection during operation. When an active IP camera stream drops mid-session, the capture loop detects the failed read and continues without raising an exception, leaving the last successfully captured frame in the display buffer. The panel does not crash or freeze. Reconnection is available through the Reconnect Cameras button. Empty detection results. When YOLO returns zero detections in a frame, the system correctly handles the empty bounding box array and no Re-ID calls are made. The status bar updates to show zero detections without error.

Invalid person deletion. Attempting to remove a person who no longer exists in the database is handled silently — the DELETE query returns without error because SQLite does not raise an exception when a WHERE clause matches zero rows.

Rapid threshold adjustment. Moving both Re-ID threshold sliders rapidly and simultaneously did not produce any race condition or application crash, as slider values are read synchronously on the GUI thread before being applied.

Assessment. The system handles all tested error conditions gracefully. The main unhandled edge case identified is permanent camera failure with no active operator notification beyond a blank feed panel. A visual alert or automatic reconnection countdown would improve the operator experience in a production environment and is identified as a future enhancement.

6.7 Load Testing

Load testing assessed system behaviour under sustained operation and under conditions approaching the limits of the test hardware. Sustained two-camera operation. The system was run continuously for 30 minutes with two active IP camera feeds and detection enabled. RAM usage remained stable throughout the session with no memory leaks observed. The anonymous person tracking store grew gradually as new anonymous IDs were allocated, but remained well within acceptable memory bounds for the session duration.

High detection density. A test frame containing six simultaneously visible persons was used to evaluate Re-ID throughput under load. At six persons per frame on CPU, the combined detection and Re-ID pipeline took approximately 700–900 ms per cycle, reducing effective detection updates to approximately one per second. This is below the threshold considered adequate for active monitoring and confirms that the system is optimised for scenes containing one to three simultaneously visible persons per camera on CPU hardware.

Queue behaviour under load. The detection worker uses a single-slot queue with a non-blocking submission operation. Under high load, frames that arrive while the worker is busy are dropped rather than queued. This was confirmed to operate correctly — no queue build-up was observed, and the system consistently processed the most recently available frame rather than working through a backlog.

6.8 Security Testing

Security testing verified the claims made in Sections 5.5 and 5.6 regarding the system's security posture.

SQL injection. Attempts to inject SQL through the person's name field in the registration dialog were handled correctly. The parameterized query implementation described in Section 5.6.3 handled this correctly — the injected string was stored as a literal text value with no effect on the database schema. No tables were dropped or modified.

Filesystem access. Attempting to use path traversal sequences in the person's name field during registration did not result in files being written outside the data/profiles/ directory, as the application constructs the storage path using Python's pathlib. Path. This is identified as a partial mitigation — a stricter name validation step restricting names to alphanumeric characters and spaces only would provide stronger protection and is recommended as a future enhancement. Database file access. Direct opening of the persons.db file using an external SQLite browser confirmed that the database is currently unencrypted, with SQLCipher recommended as a production-grade enhancement to address this. This is consistent with the known limitation documented in Section 5.6 and does not represent an oversight — the current implementation prioritizes deployment simplicity, with encryption identified as a planned future enhancement. The importance of encrypting locally stored biometric and identity data in surveillance systems has been highlighted in prior work on privacy-aware system design [48].

6.9 Installation Testing

Installation testing verified that the system can be set up from scratch on a clean machine using the provided project files.

The installation process consists of three steps: installing Python 3.11, running pip install -r requirements.txt to install all dependencies, and executing run app_bat to launch the application.

On a clean Windows 11 machine, all dependencies were resolved and installed without conflicts in approximately eight minutes on a standard broadband connection. The application launched successfully on the first attempt, and the two default camera entries were created automatically.

6.10 Comparison with Existing Techniques

To contextualize the system's performance, it is compared below against two reference points: state-of-the-art academic Re-ID systems evaluated on standard benchmarks, and commercially available enterprise surveillance platforms.

Academic Re-ID models evaluated on the Market-1501 benchmark routinely achieve top-1 identification accuracy above 90

Chapter 7

Conclusion

This project set out to address a fundamental inefficiency in modern surveil-lance: the dependence on human operators to manually monitor, identify, and track individuals across multi-camera environments. The result is a fully implemented and tested, AI-driven person tracking and re-identification system that automates this process in real time, delivering a practical system capable of maintaining identity continuity across non-overlapping camera feeds without the use of facial recognition, ensuring compliance with modern data privacy standards.

The NMS-free training strategy introduced in YOLOv10 produces a mea-surable reduction in inference latency compared to YOLOv8 on identical hardware, confirming that architectural improvements at the model level translate directly into pipeline responsiveness gains without any change to surrounding application code.

A single-slot non-blocking detection queue, rather than an unbounded FIFO queue, is the correct design choice for real-time video pipelines. Drop-ping stale frames under load ensures the system always processes the most current visual state, preventing the latency accumulation that would oth-erwise make tracking unusable during bursts of activity. Cosine similarity over L2-normalised 512-dimensional ResNet18 embeddings provides a com-putationally efficient and threshold-controllable matching mechanism for appearance-based Re-ID. The choice of similarity metric has a direct and tunable effect on the false-positive rate, and exposing the threshold as a runtime slider rather than a compile-time constant significantly improves operator adaptability across different deployment environments.

Multi-threading in a PySide6 application must be structured so that all GUI updates occur exclusively on the main thread. Violating this con-straint — even for lightweight status updates — introduces race condi-tions that are difficult to reproduce and diagnose. Enforcing strict thread boundaries through signal-slot communication eliminates this class of bug entirely.

SQLite3 with parameterized queries is sufficient for single-machine surveil-lance logging at low-to-moderate event rates. The absence of a network-exposed database layer eliminates an entire category of remote attack sur-face, making this a deliberate security advantage for local deployments rather than simply a

cost-saving measure. Appearance-based Re-ID with-out domain-specific fine-tuning degrades predictably under clothing change and significant viewpoint variation. This is not a failure of the ResNet18 backbone itself, but a consequence of training on general visual features rather than surveillance-specific ones. The lesson is that backbone selection

and training data domain must be matched to the deployment scenario for Re-ID to be reliable in practice.

If this project were to be developed further, the following improvements are recommended. The Re-ID backbone should be replaced with a transformer-based architecture such as TransReID, fine-tuned on a large-scale person Re-ID dataset like Market-1501 or DukeMTMC, to substantially improve accuracy under viewpoint change and partial occlusion. Database encryp-tion via SQLCipher, a role-based operator login system, and automatic reconnection logic for dropped camera streams would each address doc-umented limitations and bring the system to a production-ready secu-rity standard. The anonymous tracking store should be persisted to disk across sessions so that unregistered individuals can be tracked longitudi-nally rather than receiving a new ID on every restart. For deployment beyond two cameras, the single detection worker per camera should be of-floaded to a GPU-backed inference server, decoupling the GUI process from the compute load entirely. Finally, 2D path reconstruction — projecting tracked identities onto a top-down floor plan using homographic transfor-mation — would transform the system from a passive feed monitor into an active spatial intelligence platform, enabling detection of loitering, route anomalies, and crowd density in real time.

References

[1] N. Dalal and B. Triggs, ”Histograms of oriented gradients for human detection,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), San Diego, CA, USA, Jun. 2005, pp. 886–893.

[2] M. Enzweiler and D. M. Gavrila, ”Monocular pedestrian detection: Survey and experiments,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 31, no. 12, pp. 2179–2195, Dec. 2009.

[3] P. Viola and M. Jones, ”Rapid object detection using a boosted cascade of simple features,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Kauai, HI, USA, Dec. 2001, pp. 511–518.

[4] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, ”Object detection with discriminatively trained part-based models,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, no. 9, pp. 1627–1645, Sep. 2010.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ”ImageNet classification with deep convolutional neural networks,” in Adv. Neural Inf. Process. Syst. (NeurIPS), Lake Tahoe, NV, USA, Dec. 2012, pp. 1097–1105.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, ”Rich feature hierarchies for accurate object detection and semantic segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Columbus, OH, USA, Jun. 2014, pp. 580–587.

[7] R. Girshick, ”Fast R-CNN,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Santiago, Chile, Dec. 2015, pp. 1440–1448.

[8] S. Ren, K. He, R. Girshick, and J. Sun, ”Faster R-CNN: Towards real-time object detection with region proposal networks,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, ”You only look once: Unified, real-time object detection,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Las Vegas, NV, USA, Jun. 2016, pp. 779–788.

[10] W. Liu et al., ”SSD: Single shot multibox detector,” in Proc. Eur. Conf.

Comput. Vis. (ECCV), Amsterdam, Netherlands, Oct. 2016, pp. 21–37.

[11] J. Redmon and A. Farhadi, ”YOLOv3: An incremental improvement,” arXiv preprint arXiv:1804.02767, Apr. 2018.

[12] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, ”YOLOv4: Optimal speed and accuracy of object detection,” arXiv preprint arXiv:2004.10934, Apr. 2020.

[13] G. Jocher et al., ”Ultralytics YOLOv8,” GitHub, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics. [Accessed: Apr. 2026].

[14] A. Wang et al., ”YOLOv10: Real-time end-to-end object detection,” in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 37, 2024.

[15] S. Shao et al., ”CrowdHuman: A benchmark for detecting human in a crowd,” arXiv preprint arXiv:1805.00123, May 2018.

[16] A. W. M. Smeulders et al., ”Visual tracking: An experimental survey,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 36, no. 7, pp. 1442–1468, Jul. 2014.

[17] F. Henriques, R. Caseiro, P. Martins, and J. Batista, ”High-speed tracking with kernelized correlation filters,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 37, no. 3, pp. 583–596, Mar. 2015.

[18] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, ”Visual object tracking using adaptive correlation filters,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), San Francisco, CA, USA, Jun. 2010, pp. 2544–2550.

[19] L. Bertinetto et al., ”Fully-convolutional Siamese networks for object tracking,” in Proc. ECCV Workshops, Amsterdam, Netherlands, Oct. 2016, pp. 850–865.

[20] B. Li et al., ”High performance visual tracking with Siamese region proposal network,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Salt Lake City, UT, USA, Jun. 2018, pp. 8971–8980.

[21] B. Li et al., ”SiamRPN++: Evolution of Siamese visual tracking with very deep networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Long Beach, CA, USA, Jun. 2019, pp. 4282–4291.

[22] X. Chen et al., ”Transformer tracking,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Nashville, TN, USA, Jun. 2021, pp. 8126–8135.

[23] B. Ye et al., ”Joint feature learning and relation modeling for tracking: A one-stream framework,” in Proc. Eur. Conf. Comput. Vis. (ECCV), Tel Aviv, Israel, Oct. 2022, pp. 341–357.

[24] Y. Wu, J. Lim, and M. H. Yang, ”Object tracking benchmark,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 37, no. 9, pp. 1834–1848, Sep. 2015.

[25] M. Kristan et al., ”The seventh visual object tracking VOT2019 challenge results,” in Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCVW), Seoul, Korea, Oct. 2019.

[26] W. Luo et al., ”Multiple object tracking: A literature review,” Artificial Intelligence, vol. 293, p. 103448, Apr. 2021.

[27] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, ”Simple online and realtime tracking,” in Proc. IEEE Int. Conf. Image Process. (ICIP), Phoenix, AZ, USA, Sep. 2016, pp. 3464–3468.

[28] N. Wojke, A. Bewley, and D. Paulus, ”Simple online and realtime tracking with a deep association metric,” in Proc. IEEE Int. Conf. Image Process. (ICIP), Beijing, China, Sep. 2017, pp. 3645–3649.

[29] Q. Wang et al., ”MPNTracker: Learning a message passing network for multi-object tracking,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 43, no. 5, pp. 1523–1537, May 2021.

[30] T. Meinhardt et al., ”TrackFormer: Multi-object tracking with transformers,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), New Orleans, LA, USA, Jun. 2022, pp. 8844–8854.

[31] Y. Zhang et al., ”ByteTrack: Multi-object tracking by associating every detection box,” in Proc. Eur. Conf. Comput. Vis. (ECCV), Tel Aviv, Israel, Oct. 2022.

[32] P. Dendorfer et al., ”MOTChallenge: A benchmark for single-camera multiple target tracking,” Int. J. Comput. Vis., vol. 129, pp. 845–881, 2021.

[33] C. Baisa, ”Occlusion-robust multi-target multi-camera pedestrian track-ing,”

IET Comput. Vis., vol. 15, no. 5, pp. 361–375, 2021.

[34] M. Farenzena et al., ”Person re-identification by symmetry-driven accumulation of local features,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), San Francisco, CA, USA, Jun. 2010, pp. 2360–2367.

[35] S. Liao et al., ”Person re-identification by local maximal occurrence representation and metric learning,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Boston, MA, USA, Jun. 2015, pp. 2197–2206.

[36] L. Zheng et al., ”Scalable person re-identification: A benchmark,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Santiago, Chile, Dec. 2015, pp. 1116–1124.

[37] Y. Sun et al., ”Beyond part models: Person retrieval with refined part pooling,” in Proc. Eur. Conf. Comput. Vis. (ECCV), Munich, Germany, Sep. 2018, pp. 501–518.

[38] W. Li, X. Zhu, and S. Gong, ”Harmonious attention network for person re-identification,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Salt Lake City, UT, USA, Jun. 2018, pp. 2285–2294.

[39] N. Wojke and A. Bewley, ”Deep cosine metric learning for person re-identification,” in Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV), Lake Tahoe, NV, USA, Mar. 2018, pp. 748–756.

[40] S. He et al., ”TransReID: Transformer-based object re-identification,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Montreal, Canada, Oct. 2021, pp. 15013–15022.

[41] M. Ye, J. Shen, G. Lin, T. Xiang, L. Shao, and S. C. H. Hoi, ”Deep learning for person re-identification: A survey and outlook,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 6, pp. 2872–2893, Jun. 2022.

[42] W. Deng et al., ”Image-image domain adaptation with preserved self-similarity and domain-dissimilarity for person re-identification,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Salt Lake City, UT, USA, Jun. 2018, pp. 994–1003.

[43] Y. Ge et al., ”Mutual mean-teaching: Pseudo label refinery for unsupervised domain adaptation on person re-identification,” in Proc. Int.

Conf. Learn. Representations (ICLR), Addis Ababa, Ethiopia, Apr. 2020.

[44] Y. Ge, D. Chen, and H. Li, ”Self-paced contrastive learning with hybrid memory for domain adaptive object re-identification,” in Adv. Neural Inf. Process. Syst. (NeurIPS), virtual, Dec. 2020.

[45] K. He, X. Zhang, S. Ren, and J. Sun, ”Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[46] F. Donald, G. Roodt, and M. Basson, ”Work exposure and vigilance decrements in closed circuit television surveillance,” Applied Ergonomics, vol. 47, pp. 220–228, Mar. 2015.

[47] Sandia National Laboratories, Human Reliability Analysis for Security Applications, Rep. SAND2014-17929, Sandia National Laboratories, Albuquerque, NM, USA, Sep. 2014.

[48] A. Beduschi, ”Facial recognition technology: Protecting biometric privacy in the digital age,” J. Legal Affairs Dispute Resolut. Eng. Constr., vol. 18, no. 2, 2024.

[49] OWASP Foundation, ”SQL injection prevention cheat sheet,” OWASP Cheat Sheet Series. [Online]. Available: [49] OWASP Foundation, “SQL Injection Prevention Cheat Sheet,” OWASP Cheat Sheet Series. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. [Accessed: Apr. 2026].

[50] G. Bradski, ”The OpenCV Library,” Dr. Dobb’s Journal of Software Tools, vol. 25, no. 11, pp. 120–125, Nov. 2000.

[51] A. Paszke et al., ”PyTorch: An imperative style, high-performance deep learning library,” in Adv. Neural Inf. Process. Syst. (NeurIPS), Van-couver, Canada, Dec. 2019, pp. 8024–8035.

Appendix A

User Manual

A.1 System Requirements

Before installing the system, ensure the following minimum hardware and software requirements are met. Hardware Requirements For CPU-only operation, the host machine should have a minimum of 8 GB RAM, an Intel Core i5 or equivalent processor, and at least 4 GB of available storage for model weights, the application, and the database. For GPU-accelerated operation, an NVIDIA GPU with CUDA support and a minimum of 4 GB VRAM is recommended. A stable local network connection is required for IP camera feeds. Software Requirements The system requires Windows 10 or Windows 11 as the operating system. Python 3.11 must be installed and available on the system PATH. No additional software installations are required beyond those handled by the pip installation step described in Section A.2.

A.2 Installation

Follow these steps in order on a clean Windows machine.

Step 1 — Install Python 3.11 Download the Python 3.11 installer from python.org. During installation, ensure the option ”Add Python to PATH” is checked before proceeding. Verify the installation by opening a Com-mand Prompt and running: python –version The output should confirm Python 3.11.

Step 2 — Install Dependencies Navigate to the project directory in Command Prompt and run: pip install -r requirements.txt All required libraries will be downloaded and installed automatically. This step requires an active internet connection and typically completes within several minutes depending on internet speed. All subsequent use of the application requires no internet access.

Step 3 — Launch the Application

The application will launch and two default camera entries will be created automatically. Full user interaction steps are provided in Sections A.3 through A.6 below.

A.3 Connecting a Camera

The system supports three camera source types: USB webcams, IP cameras over HTTP, and pre-recorded video files. For a USB webcam, the default index-based address (0 for the primary webcam) is pre-configured and requires no changes. For an IP camera, open the Camera Settings panel and enter the full HTTP stream URL provided by your camera application, for example as produced by the IP Webcam application on Android. For a video file, enter the full file path to the video in the camera address field. Once the camera address is configured, click Start All Cameras in the main window. The live feed will appear in the corresponding camera panel within a few seconds. If the panel remains blank, verify the camera URL and network connection, then use the Reconnect Cameras button to retry.

A.4 Registering a Person

Person registration is required before the system can identify and track a specific individual. To register a new person, follow these steps. Click the Register Person button in the main window to open the Registration Dialog. Select the camera from which the subject will be captured using the camera selector. Enter the person's name in the name field. Click Start Preview to begin the live camera feed within the dialog. Position the sub-ject at a distance of approximately one to two meters from the camera, ensuring the full body is visible in the frame. Click Capture Image to take a snapshot. A green border will flash for 200 milliseconds to confirm each capture. Repeat this process to capture a minimum of three images, varying the subject's pose slightly between captures to improve Re-ID robustness. Once at least three images have been captured, the Save to Database but-ton will become active. Click it to save the person's profile and reference images to the database. A confirmation message will display the number of images saved and the storage path.

A.5 Monitoring Live Feeds

After cameras are connected and persons are registered, detection can be

activated by clicking Toggle Detection ON in the main window. Bounding boxes will appear around detected persons in the live feed, labelled with the matched person's name and confidence score where a match is found, or with an anonymous ID where no match exceeds the configured threshold. The view can be switched between the two-column camera grid and a full-width single feed using the Toggle Grid/Single View button. Camera name labels appear above each feed in grid mode. The Re-ID threshold sliders in the main window control matching sensitivity. Increasing the threshold reduces false positive identifications at the cost of potentially missing correct matches. The default values are suitable for most indoor environments with stable lighting.

A.6 Viewing Detection History

Two history viewing options are available. Click View All History to open a chronological log of all detection events across all cameras and persons. Click Person History and select a registered individual to view only the events associated with that person. Each event entry displays the person's name, timestamp, camera name, and confidence score. Where a snapshot image was saved at the time of detection and the file remains at the recorded path, it will be displayed at 500×400 pixels in the event detail panel.

Appendix B – System Requirements

B.1 Software Dependencies

Table B.1 lists all required Python libraries with their minimum version numbers as specified in requirements.txt.

Table A.1: Required Python Libraries

Library	Minimum Version	Purpose
Ultralytics	8.0.0	YOLOv10 model loading and inference
OpenCV-Python	4.8.0	Video capture, frame processing, image encoding
PySide6	6.5.0	Graphical user interface framework
PyTorch	2.0.0	Deep learning inference backend
TorchVision	0.15.0	ResNet18 model and image transforms
NumPy	1.24.0	Numerical array operations
Pillow	10.0.0	Image capture and file storage
scikit-learn	1.3.0	Cosine similarity computation

B.2 Hardware Specifications Table B.2 summarizes the minimum and recommended hardware configurations for deployment.

Table B.2: Hardware Specifications

Table A.2: System Requirements

Component	Minimum (CPU only)	Recommended (GPU)
Processor	Intel Core i5 or equivalent	Intel Core i7 or equivalent
RAM	8 GB	16 GB
Storage	4 GB available	8 GB available
GPU	Not required	NVIDIA CUDA-capable, 4 GB VRAM
Operating System	Windows 10	Windows 10 or Windows 11
Python Version	3.11	3.11
Network	Local Wi-Fi for IP cameras	Wired LAN for IP cameras

Appendix C – Database Schema

C.1 Overview

The system uses a local SQLite3 database stored at data/persons.db. The database contains two tables: persons, which stores registered individual profiles and reference image paths, and detection history, which logs every tracking event recorded during system operation. Both tables are created automatically on first launch if they do not already exist.

C.2 Table Definitions

Table A.3: persons table

Column	Data Type	Constraints	Description
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for each registered person
name	TEXT	NOT NULL	Name assigned during registration
image_folder	TEXT	NOT NULL	Path to the folder containing reference images
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Date and time of registration

Table A.4: detection_history table

Column	Data Type	Constraints	Description
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for each detection event
person_id	INTEGER	FOREIGN KEY → persons(id)	ID of the matched registered person
camera_name	TEXT	NOT NULL	Name of the camera that recorded the detection
confidence	REAL	NOT NULL	Cosine similarity score at time of detection
snapshot_path	TEXT	NULLABLE	File path to the saved snapshot image, if captured
detected_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Date and time of the detection event

Appendix D – Threshold Calibration

D.1 Purpose

The Re-ID matching pipeline uses two configurable thresholds: a primary similarity threshold that determines whether a detected person is matched to a registered profile, and a secondary anonymous tracking threshold used to assign consistent anonymous IDs to unregistered individuals across frames. This appendix documents the calibration process used to select the default threshold values shipped with the system. Threshold calibration re-sults informed the default values referenced in Chapter 5 and discussed in Chapter 6.

D.2 Calibration Method

Threshold calibration was conducted by registering three persons under controlled indoor lighting conditions and running the system across a se-ries of test scenarios. Each scenario varied one factor at a time: viewing angle, distance from camera, ambient lighting level, and time elapsed since registration. For each threshold value tested, the number of correct iden-tifications, missed identifications, and false positive matches was recorded through manual analysis of detection history logs.

D.3 Selected Default Values

Based on the calibration results summarized in Table D.1, a primary simi-larity threshold of 0.50 and an anonymous tracking threshold of 0.40 were selected as defaults. These values are pre-configured in the application at startup and can be adjusted at runtime using the threshold sliders in the main window. Operators working in environments with highly variable lighting or frequent viewpoint changes are advised to lower the primary threshold toward 0.40 to reduce missed identifications, accepting a modest increase in false positive rate.