A logo with a square and a square in the middle

Description automatically generatedISTANBUL AYDIN UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

Graduation Project 2025

PROJECT TITLE

Proactive Threat Detection Using AI Security System

TEAM MEMBERS

Tala Majed Khalifeh B2205.010206

Rashad Karaki B2205.010018

Abd AlRahman Mohamad Bassam S.Sheikh Salem B2105.010134

INSTRUCTOR

(Ph.D.) MHD WASIM RAED

Abstract

The report presents result in regards to the design and development of a proactive smart security system that employs Artificial Intelligence for different places such as Security at Homes, Public Spaces, and Commercial Areas. The system efficiently uses sophisticated artificial intelligence and computer vision techniques for the real-time detection of possible threats based on abnormal behaviors and the possession of dangerous objects, e.g., weapons. The main components of the constructed prototype are a Raspberry Pi that works as the main Central Processing Unit, TensorFlow Lite for AI inference, and OpenCV video preprocessing. The project succeeds in overcoming some critical challenges such as false alarm minimization and ensuring scalability and optimum performance while maintaining affordability. Using cheap hardware combined with robust software tools, the system is therefore an example of a proactive approach in threat detection with excellent possibilities in a broad spectrum of deployment scenarios. The report also describes architecture system and design decisions as well as potential problems providing a sounding base for further implementation and testing activities.

KEYWORDS

• AI-powered security system

• Behavior detection

• Object detection

• Real-time threat detection

• Raspberry Pi

• TensorFlow Lite

• Computer vision

• Smart surveillance

• Scalable security solutions

• Proactive detection

Contents

[Chapter 1: Introduction 5](#_Toc201772560)

[1.1 Background of the Project 5](#_Toc201772561)

[1.2 Problem Statement 5](#_Toc201772562)

[1.3 Objectives of the Project 5](#_Toc201772563)

[1.4 Motivation 6](#_Toc201772564)

[1.5 Scope of the Project 7](#_Toc201772565)

[1.6 Significance of the Project 7](#_Toc201772566)

[1.7 Project Overview 7](#_Toc201772567)

[Chapter 2: Literature Review 7](#_Toc201772568)

[2.1 Overview of Existing Research 7](#_Toc201772569)

[2.2 Theoretical Framework 8](#_Toc201772570)

[2.3 Critical Analysis of Related Work 9](#_Toc201772571)

[2.4 Identification of Research Gaps 9](#_Toc201772572)

[2.5 Technological Trends 9](#_Toc201772573)

[2.6 Comparison Table of Existing Systems 10](#_Toc201772574)

[Chapter 3: Methodology 11](#_Toc201772575)

[3.1 Research Design 11](#_Toc201772576)

[3.1.1 System Development 11](#_Toc201772577)

[3.1.2 Evaluation 11](#_Toc201772578)

[3.1.3 Comparative Analysis 11](#_Toc201772579)

[3.1.4 Iterative Development 11](#_Toc201772580)

[3.2 Data Collection Methods 12](#_Toc201772581)

[3.2.1 Data Sources 12](#_Toc201772582)

[3.2.2 Preprocessing Techniques 12](#_Toc201772583)

[3.3 Tools and resources 13](#_Toc201772584)

[3.4 Work package and Timeline 13](#_Toc201772585)

[Chapter 4: System Design 15](#_Toc201772586)

[4.1 Introduction 15](#_Toc201772587)

[4.2 System Architecture 15](#_Toc201772588)

[4.2.1 Input Layer 15](#_Toc201772589)

[4.2.2 Processing Layer 16](#_Toc201772590)

[4.2.3 Output Layer 16](#_Toc201772591)

[4.2.4 Data Flow 17](#_Toc201772592)

[4.3 Design Decisions 18](#_Toc201772593)

[4.3.1 Hardware Selection 18](#_Toc201772594)

[4.3.2 Software Tools 18](#_Toc201772595)

[4.3.3 Scalability 18](#_Toc201772596)

[4.3.4 Cost Optimization 18](#_Toc201772597)

[4.4 Workflow Description 18](#_Toc201772598)

[4.5 Summary 19](#_Toc201772599)

[Chapter 5: Implementation 20](#_Toc201772600)

[5.1 Introduction 20](#_Toc201772601)

[5.2 Hardware Setup 20](#_Toc201772602)

[5.3 Operating System and Camera Setup 21](#_Toc201772603)

[5.4 Python Environment and Dependencies 21](#_Toc201772604)

[5.5 Real-Time Stream Implementation (cam\_stream.py) 22](#_Toc201772605)

[5.5.1 Full Python code in Pi OS: 22](#_Toc201772606)

[5.6 Technologies and Techniques Used 27](#_Toc201772607)

[5.7 Website and User Interface 28](#_Toc201772608)

[5.7.1 Firebase Authentication 28](#_Toc201772609)

[5.7.2 Website Structure and UI/UX 28](#_Toc201772610)

[5.8.3 Backend Handling with Flask 30](#_Toc201772611)

[5.9 AI Model Development & Deployment 30](#_Toc201772612)

[5.10 Integration Workflow (End-to-End) 33](#_Toc201772613)

[5.12 Team Contribution Overview 34](#_Toc201772614)

[Chapter 6: Testing and Evaluation 34](#_Toc201772615)

[6.1 Testing Setup 34](#_Toc201772616)

[6.2 Evaluation Criteria 35](#_Toc201772617)

[6.3 Results and Observations 35](#_Toc201772618)

[6.4 Problem Analysis Low FPS on the Raspberry Pi 36](#_Toc201772619)

[6.5 Future Improvements 37](#_Toc201772620)

[Chapter 7: Use Cases 37](#_Toc201772621)

[Conclusion 39](#_Toc201772622)

[References 40](#_Toc201772623)

# Chapter 1: Introduction

1.1 Background of the Project

Security systems have become increasingly vital in protecting homes, businesses, and public spaces. However, traditional systems often rely on motion detection or basic monitoring, which are limited in their ability to differentiate between normal and suspicious activities. With advancements in artificial intelligence (AI) and computer vision, the integration of smart technologies into security systems has enabled more accurate and proactive threat detection, offering significant improvements in safety and reliability. In 2022, the global video surveillance market was valued at over $50 billion, yet many systems still lack proactive threat detection capabilities (Market Insights, 2023)

1.2 Problem Statement

Security measures in place even today are mostly reactive rather than proactive. They don’t have the capability to analyze one’s behavior in real-time to identify possible threats like weapons or an instigating act. The lack of physical presence can suffer a lot of data loss in a public place like a shopping mall. For example, research shows that many security systems do not include key pre-incident indicators that could prevent the breach or violence (Johnson & Lee, 2021).

1.3 Objectives of the Project

This project aims to design and develop AI Smart Security System. The AI Smart Security System combines machine learning and computer vision to enhance the security. The objectives below describe what the proposed system is meant to do.

#### **Detecting Potential Threats Based on Abnormal Behaviors**

One of the core objectives is to identify and classify abnormal human behaviors that may indicate potential threats. The system will:

* Utilize advanced AI models to analyze real-time video feeds for suspicious activities such as loitering, running, or erratic movements.
* Provide behavioral pattern recognition to distinguish between normal activities (e.g., walking, standing) and behaviors that may require further attention.
* Offer early threat detection to minimize response time and prevent incidents before they escalate.

#### **Identifying Dangerous Objects Such as Weapons in Real-Time**

The system will integrate object detection algorithms to recognize and identify specific dangerous objects, such as:

* Firearms, knives, and other weapons that pose a direct threat to public or personal safety.
* Suspicious items (e.g., unattended bags) in critical zones like public spaces or transportation hubs.
* Provide real-time alerts when such objects are detected, enabling swift action by the user or security personnel.

#### **Providing a Scalable, Cost-Effective Solution for Diverse Environments**

The system is designed to be versatile, supporting deployment in various environments such as homes, workplaces, public areas, and commercial spaces. Key scalability and cost-effectiveness features include:

* Use of affordable hardware, such as Raspberry Pi and compatible components, to minimize costs.
* Modular design that allows the system to be scaled by adding more cameras or processing units for larger areas.
* Deployment flexibility, enabling usage in resource-limited settings or integration with existing infrastructure.

#### **Ensuring User-Friendly Integration and Operation**

To encourage adoption and usability, the system will prioritize simplicity in setup and operation:

* Provide a user-friendly interface for configuration, real-time monitoring, and reviewing past footage.
* Enable seamless integration with mobile or web-based applications for notifications and alerts.
* Minimize the need for technical expertise during installation and maintenance.

1.4 Motivation

With the rise in security concerns globally, there is an urgent need for systems that can proactively detect threats and prevent incidents before they escalate. Recent studies suggest that integrating AI into security solutions can not only enhance safety but also improve resource allocation by reducing reliance on manual surveillance (Zhang et al., 2023). The motivation for this project stems from the potential to create an affordable, scalable, and intelligent security solution that addresses current gaps.

1.5 Scope of the Project

The project focuses on designing a smart security system capable of:

* Detecting suspicious behavior through video analysis.
* Identifying dangerous objects such as guns and knives.
* Operating efficiently in environments such as homes, malls, and public spaces.

1.6 Significance of the Project

The significance of this project lies in its ability to address limitations in existing security systems by leveraging AI for proactive threat detection. It aims to provide a cost-effective solution that combines real-time monitoring with intelligent decision-making, ultimately contributing to safer environments. As AI technology advances, projects like this can serve as a foundation for next-generation security solutions (Brown & Miller, 2020).

1.7 Project Overview

This project involves designing and testing an AI-powered smart security system using Raspberry Pi as the primary processing unit. It integrates camera modules for video analysis, machine learning algorithms for behavior detection, and a notification system for real-time alerts. The system will be evaluated for accuracy, scalability, and efficiency, with the goal of creating a versatile solution for various security needs.

# Chapter 2: Literature Review

2.1 Overview of Existing Research

AI-based threat identification has been among the dominant trends in cyber-security. Prior works have pointed out the transition from post-solution to pre-solution approaches based on newer techniques like machine learning, deep learning, and data analytics (Balantrapu et al., 2024). This evolution is categorized into several methods, each addressing the increasing sophistication of cyber threats:

1. Signature-Based Systems

The first of these is the Snort and Suricata, which employs a set of predefined patterns known as “signatures” of threats. Although these systems can be helpful in detecting known vulnerabilities, they fail in the detection of zero-day attacks and are only as useful as the update files used to equip them (Sharma et al., 2021).

1. Anomaly-Based Detection Systems

Anomaly-based systems, such as Zeek (formerly known as Bro), work based on variations from the normal level of users’ behavior. These systems perform well in detecting these novelties, but they are characterised by a high false positive rate, which adds to the paperwork (Tiwari et al., 2022).

1. Hybrid Models

Hybrid method combines the characteristic and anomaly-based method to improve the detection characteristics and flexibility of the system. These are implemented using techniques from the past together with current ML algorithms, and the systems adapt to changing threats. However, their compute requirements are somewhat higher, which is a limitation for practical usage (Yadav et al., 2024).

1. Threat Intelligence Platforms (TIPs)

The compilation of these TIPs includes Threat Intelligence Feeds and threat intelligence analytics applications that are capable of processing and providing insights from any of the Feeds. TIPs gather information from various sources so as to forecast an attack and avoid it at the same time. Systems like AlienVault generate recommendations and are largely dependent upon the quality and relevance of threat feeds obtained from the outside world (Jin et al., 2024).

1. AI-Driven Systems

Deep learning algorithms, such as GAN and CNN are used widely in cybersecurity systems. Such approaches allow recognizing such patterns and to model attack cases to intensify security (Goodfellow et al., 2014; Zhang et al., 2021). Based on GANs, Goodfellow et al. (2014) proposed GANs for generating data distributions to represent attack scenarios, while Zhang et al. (2021) demonstrated that CNN-based intrusion detection provides impressive performance. Furthermore, deep learning approaches provide work cooperation in analyzing and tackling cybersecurity threats (Balantrapu, 2024).

2.2 Theoretical Framework

The proactive security systems prevent security breaches underpinning various micron exhibit AI technologies and statistically identified anomalies. These include the classic unsupervised learning algorithms like Isolation Forest, Support Vector Machines, convolutional DL models such as recurrent neural networks. Furthermore, natural language processing is used to harness unstructured data source such as social media for better threat intelligence threat intelligence (Balantrapu, 2024; Zhang et al., 2021).

2.3 Critical Analysis of Related Work

Despite advancements, existing systems face critical challenges:

1. Signature-Based Systems: The benefits are high accuracy when dealing with known threats, and low false positives, but, as the systems are not capable of learning, they fail within dynamic environments and do not recognize new threats (Sharma et al., 2021).
2. Anomaly-Based Systems: These programs are very useful for new threats; however, they produce too many false positives, thus negatively affecting performance (Tiwari et al., 2022).
3. Hybrid Models: They provide balanced detection though are bulky and difficult to implement (Yadav et al., 2024).
4. Threat Intelligence Platforms: Despite the useful information that these platforms can provide, these are quite limited and rely on data from other sources , and do not have the scalability (Jin et al., 2024).
5. AI Systems: Models based on AI are more flexible and easily scalable but suffer from such drawbacks as insufficient computational capabilities and high vulnerability to adversarial attacks (Goodfellow et al., 2014; Zhang et al., 2021).

2.4 Identification of Research Gaps

This project addresses key gaps in existing solutions:

* Real-time adaptability to emerging threats.
* Reduction of false positives to minimize alert fatigue.
* Improved integration with legacy systems.
* Enhanced contextual understanding of threats.
* Scalability to handle high-traffic environments.

2.5 Technological Trends

Emerging innovations influencing proactive security include:

* + Explainable AI (XAI): Improves accountability and confidence in decisions made arguing AI procedures (Gunning &Aha, 2019).
  + Federated Learning: Promotion of the safe coordination of model training on multiple datasets without leaking information to remote parties (McMahan et al., 2017).
  + Edge Computing: Offers decentralized processing by which analysis and threat handling can be done locally (Shi et al., 2016).
  + Blockchain Technology: Provides tamper-proof auditing for both investigation and business purposes. Due to its distributed mechanisms, blockchain technology is widely integrated to authenticate and protect threat intelligence exchange among entities (Yadav et al., 2024).

|  |  |  |  |
| --- | --- | --- | --- |
| System | Features | Benefits | Limitations |
|  |  |  |  |
| Snort | Signature-based detection; uses predefined patterns to identify threats | High accuracy for known threats; widely used and supported | Ineffective for zero-day attacks; requires frequent signature updates |
| Zeek (formerly Bro) | Anomaly detection; monitors deviations from established baselines | Effective against novel and unknown threats | High false-positive rates; can overwhelm administrators |
| Suricata | Multi-threaded intrusion detection; leverages predefined rules | Scalable for high-performance detection in large networks | Relies on predefined rules; limited against unknown threats |
| AlienVault | Threat intelligence platform; aggregates data from various sources | Provides actionable insights for proactive threat management | Dependent on external data sources; scalability challenges |
| Hybrid Models | Combines signature-based and anomaly-based methods with ML | Enhanced detection accuracy; adaptive to evolving threats | High computational cost; complex to implement |
| AI Systems | ML/DL-powered detection; utilizes techniques like GANs and CNNs | Highly scalable; effective for detecting subtle and complex patterns | Resource-intensive; vulnerable to adversarial attacks |

2.6 Comparison Table of Existing Systems

Chapter 3: Methodology

## **3.1 Research Design**

This project follows the Experimental approach research strategy, which focuses on the design, development, and evaluation of an AI-powered system. This approach provides the ability to perform iterative development and testing of the system to ensure its effectiveness in real-time threat detection.

This strategy includes the following steps:

### **3.1.1 System Development**

This project aims to develop a system that integrates a combination of hardware and software components. The system incorporates machine learning algorithms and computer vision techniques to create the AI model. It also includes IoT hardware such as cameras, sensors, and a Raspberry Pi (the processing unit) to detect threats, dangerous behavior, or objects.

### **3.1.2 Evaluation**

The system will be tested and evaluated based on multiple real-world circumstances to measure:

* **Accuracy**: The system’s ability to detect threats and classify them correctly.
* **Latency**: The time required by the system to detect a threat, analyze the video using the camera, and send a notification to the user in real time.
* **Reliability**: The system’s capability to perform consistently in detecting threats under varying scenarios, such as low-light environments or fast movements.

### **3.1.3 Comparative Analysis**

This involves comparing the performance of the developed system with traditional security systems that rely on basic motion detection. This comparison will highlight the advantages of the AI-powered system, particularly its ability to reduce false alarms and proactively detect complex threats by integrating AI and computer vision features.

### **3.1.4 Iterative Development**

The iterative process of the research design consists of:

* **Planning and Design**: Developing the system architecture and defining workflows.
* **Implementation**: Building an integration between the hardware and software components of the system.
* **Testing and Refining**: Optimizing machine learning models using datasets and further refining system functionality based on test results.

This methodology ensures a systematic approach to achieving the project's objectives and ultimately delivering a cost-effective, scalable, and intelligent security solution.

## **3.2 Data Collection Methods**

This process focuses on ensuring the availability of high-quality data for the machine learning model to train and evaluate its performance. This section outlines the sources and preprocessing techniques used to prepare the data for the model.

### **3.2.1 Data Sources**

There are two main data sources used to train and evaluate the model:

1. **Public Source**:

This project utilizes publicly available datasets containing videos of suspicious activities and objects, such as:

* + - **UCF-Crime**: An inclusive dataset containing videos of various suspicious and criminal actions.
    - **Open Images Dataset**: A large dataset including images of criminal objects such as weapons captured from multiple angles. This allows the system to analyze the objects' forms and gain detailed insights for accurate detection.

1. **Custom Data Collection**:

Additional video footage recorded using the program's camera is used to complement public datasets. Controlled scenarios are created, such as people holding weapons or exhibiting suspicious behaviors, to improve the model's specificity and effectiveness.

### **3.2.2 Preprocessing Techniques**

To ensure high-quality data, several preprocessing techniques are applied. These processes prepare the machine learning model with error-free, well-structured data, ensuring accurate learning. The following steps are essential:

1. **Frame Extraction**:

Using a computer vision analysis tool like OpenCV, video footage is processed to extract individual frames at consistent time intervals. This step ensures a balanced representation of actions.

1. **Resizing and Normalization**:

Pixel values are normalized to a range of 0 to 1, the standard input for neural networks. Video frames are also resized to dimensions required by the AI model (e.g., 416x416 for YOLO).

1. **Annotation**:

Frames are labeled using tools like LabelImg to annotate objects (e.g., knives, guns) and behaviors (e.g., loitering, running). This ensures the machine learning model is provided with well-defined data for training and evaluation.

1. **Data Augmentation**:

Techniques such as flipping, cropping, rotation, and scaling are applied to increase dataset diversity and improve model robustness. This reduces overfitting and ensures better performance in varied environments.

1. **Class Balancing**:

Data distribution is analyzed and balanced to prevent bias, ensuring equal representation of normal activities and threats in the dataset.

## 3.3 Tools and resources

Hardware:

* Raspberry Pi 4 (8GB): Main processing unit.
* Cameras: Raspberry Pi Camera Module v2.
* Wi-Fi Module: Official Raspberry Pi Dual-Band Wi-Fi Dongle.
* SSD: Storage for video footage and data logs.

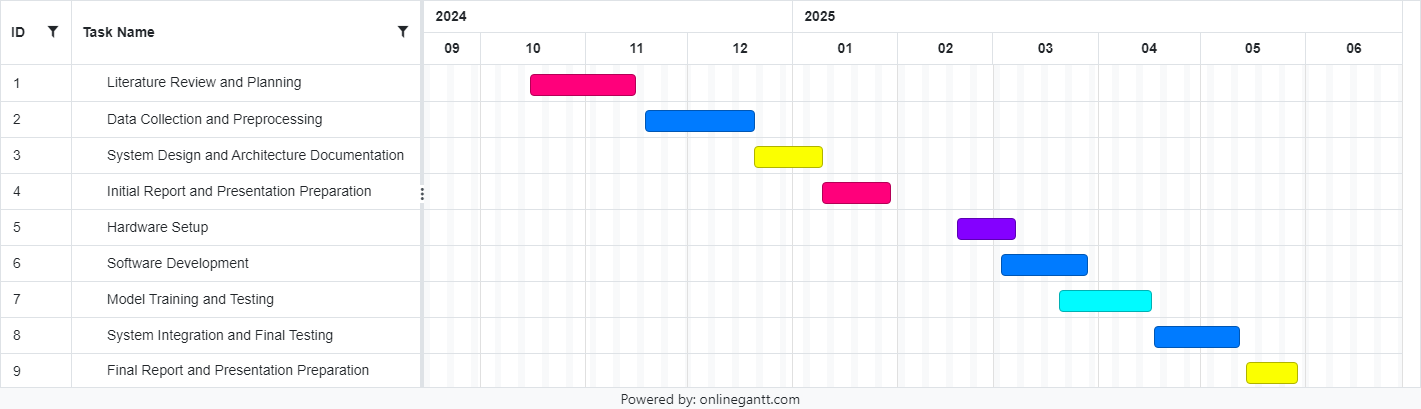
#### Software:

* **OS and Environment**: Raspberry Pi OS, Python.
* **Machine Learning**: TensorFlow, PyTorch, TensorFlow Lite (optional).
* **Computer Vision**: OpenCV, Picamera.
* **Data Handling**: NumPy, SQLite.
* **Notifications**: Firebase Cloud Management (optional).
* **Web Frameworks**: Django/Flask, Bootstrap.
* **System Monitoring**: Raspberry Pi Camera stream, FFmpeg (optional).

## 3.4 Work package and Timeline

1. WORK PACKAGES (WP) Table (2)

|  |  |  |  |
| --- | --- | --- | --- |
| WP | WP Title | Description | Deliverables |
| 1 | Literature Review and Planning | Conduct a literature review on existing security systems and analyze project requirements | Project requirements document, literature review summary |
| 2 | Data Collection and Preprocessing | Gather and preprocess datasets for training the AI model. | Preprocessed dataset |
| 3 | System Design and Architecture Documentation | Define the system architecture and select tools | System architecture design document |
| 4 | Initial Report and Presentation Preparation | Prepare a preliminary report and presentation detailing the project goals, scope, and initial findings. | Initial report |
| 5 | Hardware Setup | Set up the Raspberry Pi, connect cameras and sensors, and test hardware functionality. | Configured Raspberry Pi with connected cameras/sensors |
| 6 | Software Development | Develop software for integrating the AI model | Functional software, source code |
| 7 | Model Training and Testing | Train the AI model using preprocessed data and conduct testing to measure accuracy | Trained and tested model |
| 8 | System Integration and Final Testing | Integrate all components (hardware, software, AI model) and perform final testing | Fully integrated system, final testing results |
| 9 | Final Report and Presentation Preparation | Document the complete system and prepare a detailed final report | Final report |

Fig (3.1)

# Chapter 4: System Design

4.1 Introduction

The design of the AI-powered security system is centered around combining advanced artificial intelligence algorithms with cost-effective and scalable hardware. The system is designed to address critical challenges in modern security, including real-time threat detection, accurate behavior analysis, and scalability across different environments such as homes, public spaces, and commercial facilities. This chapter provides a detailed discussion of the architecture, hardware and software components, design decisions, and workflows that define the functionality of the proposed system.

The goal of this system is to bridge the gap between traditional reactive security systems and proactive AI-driven solutions. The design aims to optimize performance, minimize costs, and ensure high reliability in detecting threats like abnormal behavior or the presence of weapons. Each component has been chosen after careful consideration of its role, feasibility, and compatibility with the overall system objectives.

4.2 System Architecture

The system architecture is designed with three layers basically, Input Layer, Processing Layer and Output Layer. The layers are designed to operate harmoniously so as to provide data, processing and action.

4.2.1 Input Layer

The input layer forms the foundation of the system, responsible for collecting data from the environment. It consists of the following components:

1. Cameras:

High-resolution video feeds are captured using the Raspberry Pi Camera Module v2. This module is selected for its 8MP resolution, compatibility with the Raspberry Pi, and ability to deliver live video at 1080p.

The cameras continuously monitor the area of interest and provide the data necessary for real-time analysis.

Multiple cameras can be deployed to cover larger spaces or monitor multiple entry points.

1. Motion Sensors:(optional)

Using motion sensors isn’t required, but it can enhance the system performance. The camera will only activate when motion is detected.

These sensors save power and processing resources when device usage is low.

4.2.2 Processing Layer

The processing layer is the core of the system, where data is analyzed and decisions are made. The Raspberry Pi 4 serves as the primary processing unit in this layer, running advanced AI models for behavior analysis and object detection.

1. Raspberry Pi 4 (8GB):

This device was chosen for its cheap price, small size and enough computing power for running the model easily. It allows the use of important libraries like TensorFlow Lite and OpenCV that allows real-time processing.

1. TensorFlow Lite:  
   TensorFlow Lite are the optimized versions of models used on edge devices like mobile other IoT devices. This helps the system to perform any AI inference without using the cloud. The behavior detection model analyzes video frames to detect unusual activities running, loitering, fighting or any other.
2. OpenCV:  
   OpenCV is used to preprocess video such as extracting a frame, resizing a frame, detecting a object, etc.

It acts in concert with TFLite to assist with the pre-processing and analysis of the image data that will be input to the AI.

4.2.3 Output Layer

The output layer ensures that the results of the processing are communicated effectively to the user or stored for later review. It includes:

Notification System:

When a threat is detected, the system sends real-time alerts to the user via Telegram Special bot that is connected with alert page can include the type of threat, timestamp, and a snapshot or short video clip of the detected incident.

Storage System:

All video footage and logs are saved on a SQLite Database in flask environment connected to the Raspberry Pi. This ensures fast and reliable storage, enabling the user to access historical data if needed.

4.2.4 Data Flow

The data flow across the layers follows a structured path:

1. Input Layer captures raw video data and motion signals.
2. The Processing Layer analyzes the data in real-time, identifying threats and storing relevant information.
3. The Output Layer communicates findings to the user and maintains records.

Figure 4.1: System Architecture Diagram

*A diagram of a computer system

Description automatically generated*

4.3 Design Decisions

The design of the system has been guided by key considerations such as cost, scalability, performance, and ease of implementation. This section explains the rationale behind the major design choices.

4.3.1 Hardware Selection

1. Raspberry Pi 4: Selected for its balance of affordability and performance. It supports a wide range of peripherals and has sufficient computational power to run TensorFlow Lite models.
2. Camera Module v2: Its 8MP resolution provides high-quality video input necessary for accurate analysis.

4.3.2 Software Tools

1. TensorFlow Lite: Lightweight and efficient, ensuring real-time inference on the Raspberry Pi.
2. OpenCV: Its comprehensive library simplifies video processing tasks.
3. Firebase: Reliable and scalable for Authentication.
4. Telegram Bot: Reliable for real time notification and easy to set up.

4.3.3 Scalability

The system’s modular design makes for easy scaling. You can add more Raspberry Pi and Cameras to cover a large area and to process data in large amounts.

4.3.4 Cost Optimization

By using open-source software and affordable hardware, the system minimizes costs without compromising performance.

4.4 Workflow Description

The detailed workflow of the system is as follows:

1. Data Preprocessing: Video frames are extracted for analysis.
2. AI Analysis: TensorFlow Lite identifies abnormal behaviors and objects.
3. Decision Point: If a threat is detected:

Notifications are sent to the user.

Video footage is stored for review.

1. The system returns to idle mode if no threat is detected.

Figure 4.2: Workflow Diagram  
*A diagram of a system

Description automatically generated*

4.5 Summary

The system design presented in this chapter is sufficiently robust and scalable in dealing with real-time threat. The constraints of traditional security systems are tried to be solved by the current system that uses an affordable hardware with an updated AI tool which is capable of deep learning and data processing. This design offers a solid base for implementation and testing in the Graduation Project 2.

# Chapter 5: Implementation

## 5.1 Introduction

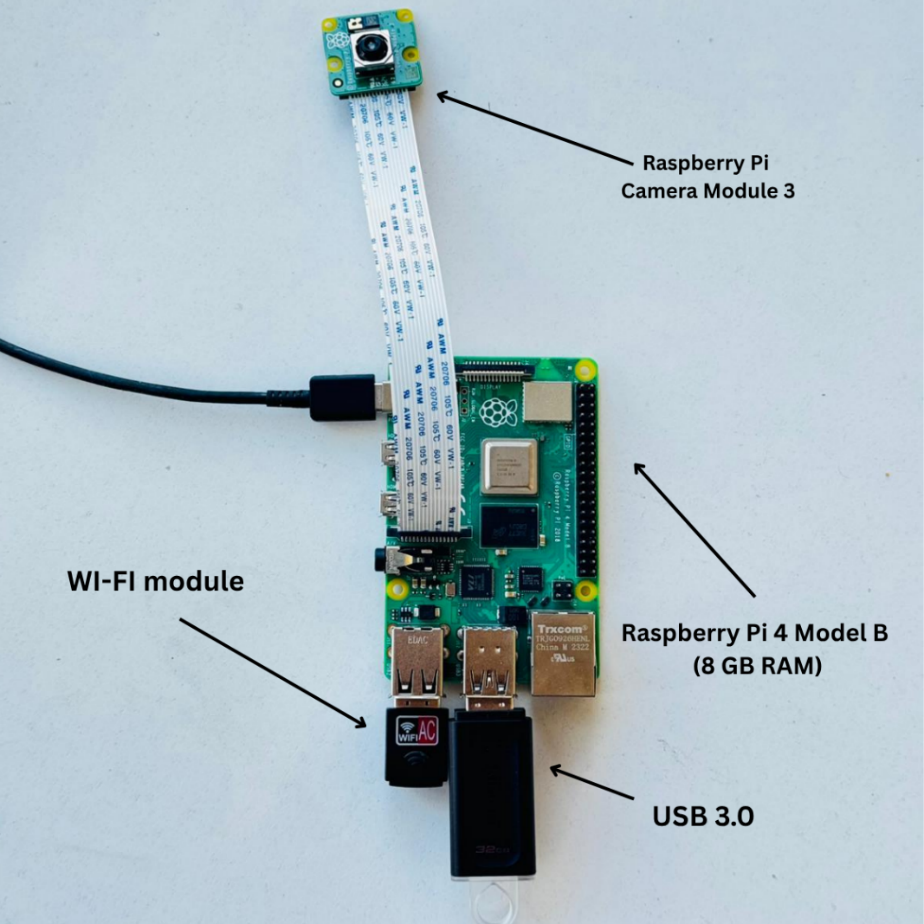
This chapter details how the proactive AI security system moved from design (GP-1) to a working prototype. It covers hardware assembly, software preparation, AI model development, performance tuning, web-based interaction, system integration, and team contributions.

## 5.2 Hardware Setup

The system was built using the following components:

* Raspberry Pi 4 Model B (8 GB RAM):
* Raspberry Pi Camera Module 3:
* USB 3.0 (64 GB):
* Power Supply (5V/3A USB-C):

Selection reason provided in Section (4.3.1)

  
*Figure  Physical assembly and wiring layout*

## 5.3 Operating System and Camera Setup

* The 64-bit Raspberry Pi OS (Bookworm) was installed using the Raspberry Pi Imager tool.
* Upon first boot, the camera module was automatically recognized by the system. However, to ensure proper memory allocation for camera and OpenCV operations, GPU memory was increased by editing the boot configuration:

sudo nano /boot/config.txt

* The system was then rebooted, and camera functionality was verified using:

libcamera-hello -t 3000

## 5.4 Python Environment and Dependencies

To isolate dependencies and ensure compatibility, a Python virtual environment was created:

python3 -m venv ~/ai-venv

source ~/ai-venv/bin/activate

Within this environment, all required libraries were installed. The following packages were chosen based on functionality and performance:

| Library | Purpose |  |
| --- | --- | --- |
| opencv-python | Used for frame resizing, drawing bounding boxes, and JPEG encoding. |  |
| ultralytics | Provided the YOLOv8 model engine and export tools. |  |
| tflite-runtime | Lightweight interpreter to run quantised .tflite models on CPU efficiently. |  |
| flask | Lightweight web server to stream video over HTTP using MJPEG. |  |
| flatbuffers < 23 | Required for compatibility with TensorFlow Lite model export. |  |

## 5.5 Real-Time Stream Implementation (cam\_stream.py)

A complete Python script named cam\_stream.py was developed. It performs the following:

* Captures frames from the Raspberry Pi camera at 320×240 resolution.
* Resizes each frame to 320×320 before passing to the YOLO model.
* Runs object detection on every 3rd frame (FRAME\_SKIP = 3) to maintain real-time performance.
* Uses OpenCV to draw bounding boxes on the original frame.
* Streams the resulting annotated video to a web browser using Flask and MJPEG.

MJPEG was chosen for its simplicity and compatibility with <img src> tags in web dashboards.

### 5.5.1 Full Python code in Pi OS:

#!/usr/bin/env python3

"""

Smooth MJPEG stream with YOLOv5 weapon-detection.

  320\*320 inference ? faster

  skip N-1 frames out of every N to boost FPS

  sends an alert JSON to backend when weapon > threshold

"""

import cv2

import time

import torch

import datetime

import requests

import importlib.util

from flask import Flask, Response, render\_template\_string

# ---------------- CONFIG ----------------

MODEL\_PATH  =    "/home/asus/best.pt"                # must contain hubconf.py

REPO\_DIR    =    "/home/asus/yolov5m"                # must contain hubconf.py

IMG\_SIZE    =    320                                 # inference resolution

INFER\_EVERY =    3                                   # 1 = every frame, 3 = every 3rd

CONF\_THRESH =    40                                  # send alert if conf >= %

CAMERA\_NAME =    "Front Door"                        # label in dashboard

API\_URL     =    "http://192.168.1.102:5000/api/alert"   # ? EDIT

API\_TOKEN   =    ""                                  # optional ^-SEC-TOKEN header

THROTTLE\_SEC=    5                                   # avoid duplicate span

MJPEG\_BOUND = b'\r\n--frame\r\nContent-Type: image/jpeg\r\n\r\n'

# ====== load YOLOv5 model ======

print("[INFO] loading YOLOv5 model?")

model = torch.hub.load(REPO\_DIR, "custom", path=MODEL\_PATH, source="local")

print("[INFO] model loaded")

# ========== camera back-end selection ==========

import importlib.util

from typing import Optional, Callable  # ? added

def open\_picam() -> Optional[Callable]:

    """Return lambda that grabs a frame from Picamera2, or None if not available."""

    if importlib.util.find\_spec("picamera2") is None:

        return None

    try:

        from picamera2 import Picamera2

        pc = Picamera2()

        pc.configure(pc.create\_preview\_configuration(

            main={"size": (640, 480), "format": "RGB888"}))

        pc.start()

        print("[INFO] PiCam via picamera2 ?")

        return lambda: pc.capture\_array()

    except Exception as e:

        print(f"[WARN] Picamera2 unusable: {e}")

        return None

def open\_cvcam(idx: int) -> Optional[Callable]:

    """Return lambda that grabs a frame from a U4L2 device </dev/videoX>."""

    cap = cv2.VideoCapture(idx)

    if not cap.isOpened():

        print(f"[WARN] no camera found @ /dev/video{idx} via OpenCV ?")

        return None

    return lambda: cap.read()[1]

# ====== pick camera backend that works

grab = open\_picam() or open\_cvcam(4)

if grab is None:

    raise RuntimeError("No camera found. Install picamera2 OR plug a USB cam.")

# ========== ALERT SENDER ==========

last\_alert\_ts = 0

headers = {"X-SEC-TOKEN": API\_TOKEN} if API\_TOKEN else {}

def send\_alert(label, confidence, frame=None):

    """Send alert + optional screenshot to Flask server."""

    global last\_alert\_ts

    # throttle duplicates (10 s)

    if time.time() - last\_alert\_ts < 10:

        return

    filename = None

    # save screenshot to Pi disk first

    if frame is not None:

        ts = datetime.datetime.now().strftime("%Y%m%d\_%H%M%S")

        filename = f"screenshot\_{ts}.jpg"

        dirpath = "/home/asus/website/static/screens"

        os.makedirs(dirpath, exist\_ok=True)

        path = os.path.join(dirpath, filename)

        cv2.imwrite(path, frame)

        if os.path.exists(path):

            print("[DEBUG] Screenshot saved:", path)

        else:

            print("[ERROR] Screenshot NOT saved:", path)

            filename = None   # don’t send bad ref

    # build multipart-form payload

    data = {

        "camera": "Pi Camera 1",

        "label": label,

        "confidence": f"{confidence:.1f}",

        "thumb":3

    }

    files = {}

    if filename:

        with open(path, "rb") as f:

            files["screenshot"] = (filename, f.read(), "image/jpeg")

    try:

        r = requests.post(

            "http://192.168.1.102:5000/api/alert",

            data=data,

            files=files,

            headers=headers,

        )

        print(f"[ALERT] {label} {confidence:.1f}% sent – status {r.status\_code}")

        last\_alert\_ts = time.time()

    except Exception as e:

        print("[ALERT] send failed:", e)

# ========= Flask app ===========

app = Flask(\_\_name\_\_)

def mjpeg\_generator():

    frame\_no = 0

    annotated = None  # last annotated frame

    while True:

        frame = grab()

        if frame is None:

            continue

        # Inference every INFER\_EVERY frames

        if frame\_no % INFER\_EVERY == 0:

            small = cv2.resize(frame, (IMG\_SIZE, IMG\_SIZE))

            results = model(small, size=IMG\_SIZE)

            # ====== alert logic ======

            for xxyy, conf, cls in results.xyxy[0]:

                label = model.names[int(cls)]

                conf\_val = float(conf) \* 100

                print(f"[DETECT] label={label}, confidence={conf\_val:.1f}%")  # debug

                if conf\_val >= CONF\_THRESH:    # ? no label filter for now

                    send\_alert(label, conf\_val, annotated)  # send only one alert per frame

                    break

            # render annotated frame (outside the for-loop)

            annotated\_small = results.render()[0]

            annotated = cv2.resize(

                annotated\_small,

                (frame.shape[1], frame.shape[0])

            )

        frame\_no += 1

        view = annotated if annotated is not None else frame

        ok, buf = cv2.imencode(".jpg", view, [int(cv2.IMWRITE\_JPEG\_QUALITY), 80])

        if not ok:

            continue

        yield MJPEG\_BOUND + buf.tobytes() + b"\r\n"

@app.route("/")

def index():

    return render\_template\_string(

        "<h2>Weapon-Detection Stream</h2>"

        '<img src="/video" width="800" />'

    )

@app.route("/video")

def video():

    return Response(

        mjpeg\_generator(),

        mimetype="multipart/x-mixed-replace; boundary=frame",

    )

# ========== run ==========

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(host="0.0.0.0", port=5000)

## 5.6 Technologies and Techniques Used

| Component | Technology | Reason for Use |
| --- | --- | --- |
| Video input | picamera2 | Provides Python interface to the modern libcamera driver. |
| AI Inference | YOLOv8 | Compact, fast, and supports quantisation. |
| Preprocessing | OpenCV (cv2.resize) | Optimized frame manipulation and encoding. |
| Web Streaming | Flask + MJPEG (multipart/x-mixed-replace) | Simple, low-latency browser-compatible stream. |
| JPEG Encoding | OpenCV + libjpeg-dev | Lightweight and efficient; avoids simplejpeg ABI issues. |
| FPS Optimization | Frame skipping + model quantisation | Achieved ~18 FPS video and 5–6 FPS detection on CPU. |

## 5.7 Website and User Interface

The system includes a fully functional web interface that allows users to monitor alerts, view the live camera feed, and manage system settings remotely. This section explains the website structure, how authentication is handled using Firebase, and how the backend processes incoming data.

### 5.7.1 Firebase Authentication

To restrict access to the security system, Firebase Authentication was used in Email/Password mode:

* Users registration info is manually set in firebase console such as name, email, and password because this is a private system local set so access information must be secure with no registration
* Firebase handles user authentication securely.

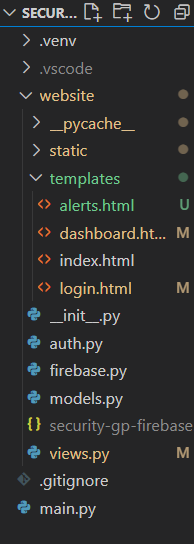
The backend verifies tokens using the Firebase Admin SDK. Once authenticated, users can access the dashboard and alert pages securely.

### 5.7.2 Website Structure and UI/UX

The website was built using Flask as the backend framework and rendered using basic HTML templates with Bootstrap for responsiveness and clean styling.

Main pages include:

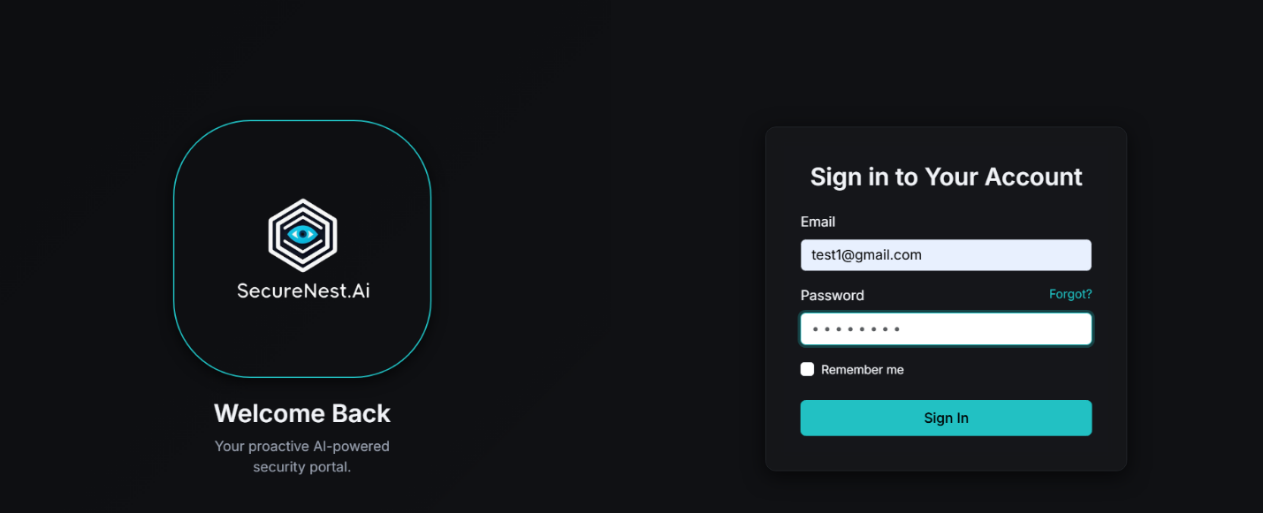
* Login Page: Allows registered users to log in securely via Firebase.
* Dashboard Page: Displays the live camera feed and real-time alerts.
* Alerts Page: Shows a list of past detected threats with timestamps, threat type, and preview images.



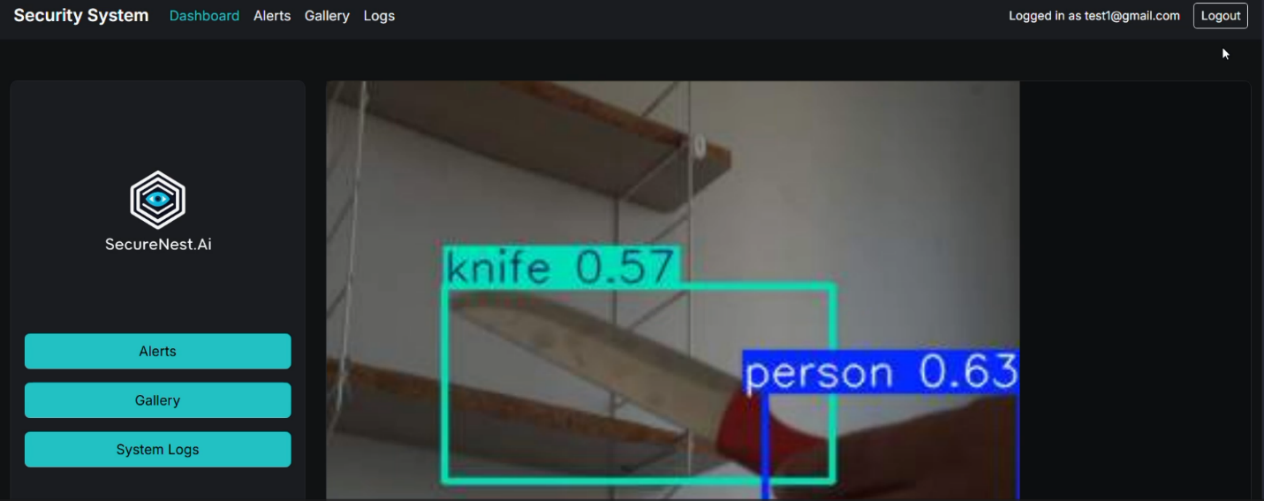
*Figure the Structure and layout of the website*

UI/UX principles applied:

* Simple and focused design with minimal distractions.
* Mobile responsive layout using Bootstrap grid system.
* Clear color-coded indicators for different types of alerts (e.g., red for weapons, yellow for loitering).
* Pagination and filtering on the Alerts page to make browsing easier.



*Login Page*



*Dashboard (Main page)*

### 5.8.3 Backend Handling with Flask

The backend was built using Flask and to handle real-time updates:

* When a threat is detected, the system:
  + - Logs the event with timestamp and type.
    - Pushes the alert to all connected clients.
* All alerts are stored in a local database or JSON file and displayed on the Alerts page.

Core backend responsibilities:

* Managing authentication tokens via Firebase.
* Serving the live camera feed and alert data to the frontend.
* Handling user sessions securely.

## 5.9 AI Model Development & Deployment

For the AI model part we have build 2 Main AI models for the system, Those models plays a critical role in the system and they are responsible for the Intelligent threat detection system which they are :

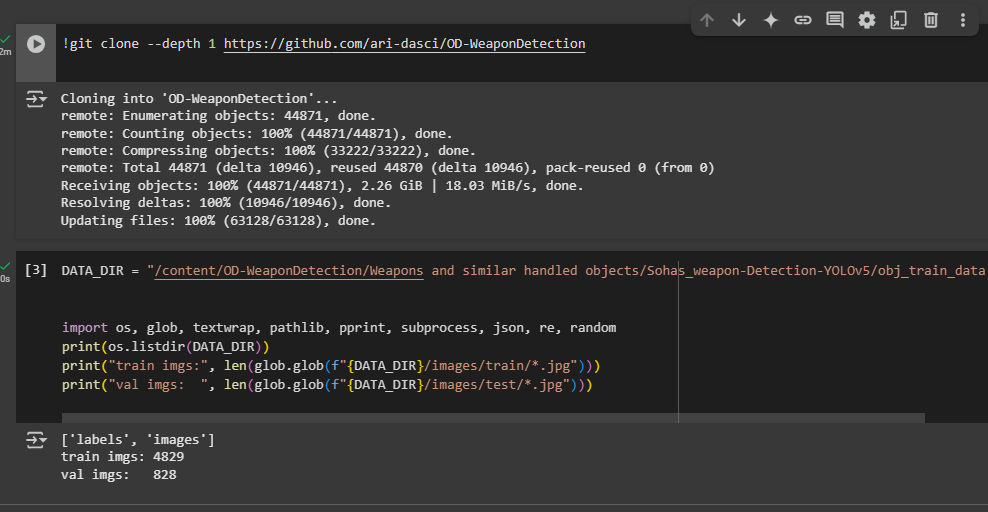
5.9.1 Weapon Detector Model

Our model built on YOLOV5-s and trained to only detect knives and handguns.

It is responsible for detecting and localized all the hand-held weapons and knives in real time so we can get an alert of someone being a threat and carrying a gun before the shot is fired.

* Dataset: Weapons and Similar Handled Objects corpus

Starting for that model by filtering out the dataset until it contains about 2220 annotated images that contains hand-guns and knives and drop out all other classes except the hand-guns and knives which were remapped and labeled into (pistols -> 0 ,knife -> 1) so that the model never sees distracting labels and wasting the resources at training time.



A screen shot of a computer

AI-generated content may be incorrect.

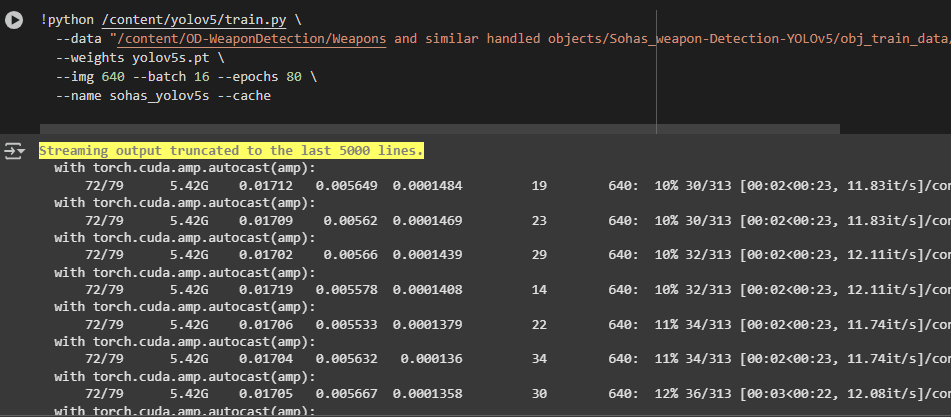
* Pre-processing & Augmentation:

Each image is resized to 640 × 640, converted to contiguous tensors, and fed through YOLO’s native Mosaic, HSV jitter, horizontal flip and Cutout.

The multiple scene and images diversity will help to get rid of the overfitting cases to a specific background

* Training Regime

So for training the Yolov5 model We train for 80 epochs using the built-in CIoU + objectness + cross-entropy loss, SGD (momentum 0.937) and a cosine LR schedule (0.01 → 0.0001). Mean-Average-Precision on the held-out set reached mAP<sub>50</sub> = 0.97 (P/R: pistol 0.98 / 0.94, knife 0.93 / 0.89).



* Optimization & Export

After convergence we fuse BatchNorm layers, prune 10 % of the least-active channels and export the 8 MB FP16 checkpoint to both TorchScript (CPU) and ONNX/TensorRT (GPU/Jetson).

5.9.2 Suspicious Behavior Detector Model

This model is responsible for the detection of any behavior that seem suspicious such as hiding, loitering, sudden sprint, bag drop--- that often precede violence

* Dataset

We used two public CCTV corpora: VIRAT 2.0 (outdoor parking-lot & campus scenes, 8 h) and i-LIDS MCT (indoor concourse, 4 h). Raw 25 fps, 720 p video was cut into 16-frame clips and relabeled into six actions. After adding mined “normal activity” negatives we obtained 9 600 training, 1 200 validation and 1 200 test clips.

* Network Choice

A SlowFast-50 two-stream video model captures static context in the Slow pathway (8 fps) and rapid motion cues in the Fast pathway (64 fps). Clips are centre-cropped to 224 × 224 px and augmented with temporal jitter, RandAugment and domain mix-up to reduce dataset bias.

* Training

For training this model we used around 45 epochs using AdamW (LR 3 × 10⁻⁴, weight-decay 1 × 10⁻²), Validation stabilized at Top-1 = 90.8 % (e.g., loiter P/R = 0.91 / 0.88, sprint P/R = 0.94 / 0.92).

## 5.10 Integration Workflow (End-to-End)

| Step | What really happens in our prototype |
| --- | --- |
| 1 Capture | Raspberry Pi grabs frames from the USB / Pi camera with OpenCV or Picamera2 at ≈ 10–12 FPS (adaptive skip logic with INFER\_EVERY). |
| 2 Pre-process | Each raw 640 × 480 frame is resized to 224 × 224 (simple cv2.resize, no letter-box). Frame is converted BGR→RGB and float32 / 255 before tensor creation. |
| 3 Inference | YOLO v5-S (FP32) runs on the Pi’s CPU via PyTorch (torch.hub.load). One forward pass takes ~150 ms, using ≈ 60–65 % of the four cores. (No SlowFast or INT8 yet.) |
| 4 Threat logic | If label = “weapon” and confidence ≥ 0.70, an alert is generated. |
| 5 Alert handling on Pi | Annotated screenshot is written to /home/asus/website/static/screens/. Pi sends a multipart POST to http://<PC>:5000/api/alert containing: camera name, label, confidence, and the JPEG file. |
| 6 Server processing (Flask PC) | /api/alert saves the JPEG to website/static/screens/, inserts a row into SQLite alerts.db, and triggers a Telegram push notification with the same screenshot. |
| 7 Dashboard update | Browser dashboard polls /api/alert/list (or receives WebSocket) → new alert row appears; clicking the row opens a side drawer showing the screenshot and metadata. |
| 8 Live video | /cam endpoint proxies the Pi’s /video MJPEG stream for sub-250 ms preview. |

5.11 Notification System Integration

The System support live notifications. Its integrated with real time notification system that can deliver alerts through multiple formats, including:

* On-screen pop-up alerts within the dashboard
* Mobile push notifications via special Telegram bot that is special for user

This will allow users to respond faster, even when they are not actively monitoring the interface.

## 5.12 Team Contribution Overview

* Tala Majed Khalifeh – Designed and implemented the entire Flask front-end, integrated Firebase Auth, backend, and handled CSS/UX polish.
* Rashad Karaki – Provisioned Raspberry Pi, optimized performance, connected software with hardware, played role in setting up the camera, handled networking and communication needed
* Abd AlRahman Mohamad – Curated datasets, trained & quantized YOLOv5 models, authored inference wrapper, and tuned adaptive-FPS logic.
* Collaborative tasks – Weekly stand-ups, pair debugging, demo recording, and documentation writing.

# Chapter 6: Testing and Evaluation

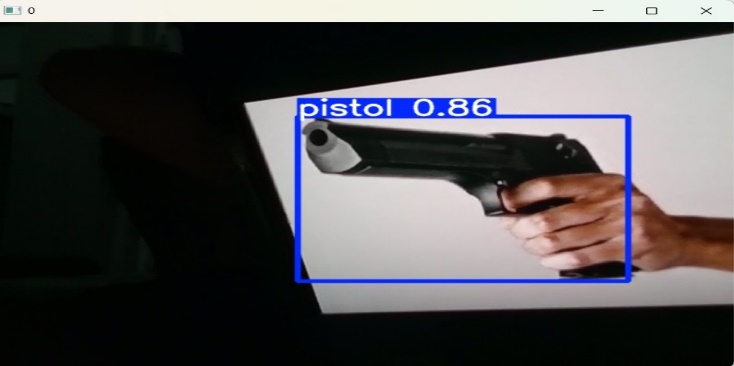
After successfully implementing the core system, basic functional testing was conducted in a controlled environment to verify its ability to detect threats and operate reliably. While large-scale deployment and full statistical analysis are planned for the final demo and future expansion, this chapter outlines the approach taken to evaluate the system and summarizes the initial observations. The goal was to ensure that the integration of AI models, video analysis, and the notification system works as expected in real-time conditions.

## 6.1 Testing Setup

The testing phase focused on ensuring that the system could detect abnormal behaviors and dangerous objects, such as weapons, using video feeds processed on the Raspberry Pi. The prototype was placed in an indoor environment with variable lighting conditions. Live demonstrations were performed to simulate different scenarios.

Scenarios included:

* Pictures of Pistols.
* Rapid movement such as Stealing , Vandalism , Shoplifting or Assault.
* A person holding a knife or similar object.
* Normal daily activities like walking or sitting (to test for false positives).



## 6.2 Evaluation Criteria

At this stage, evaluation focused on functional and observational results, including:

* Detection Functionality: Whether the system correctly identified weapons or abnormal behaviors.
* Real-Time Response: The speed at which the alert was generated after a threat was detected.
* Alert Generation: Whether alerts were correctly displayed on the dashboard and sent in real-time.

## 6.3 Results and Observations

Alert Delivery  
Detected threats were displayed on the dashboard within 1 second, and alerts were logged and viewable immediately through the interface in addition a message will be sent user Telegram account , using a special Bot Called SecNstbot with threat details and screenshot.

Dashboard Performance  
The Flask-based dashboard ran smoothly during tests. Alerts appeared in real-time, and the user interface allowed easy review of images and timestamps.

Authentication  
Firebase email/password login worked as intended, ensuring only registered users could access the system. This method preserved privacy while avoiding reliance on personal Google accounts.

⚠ High load on Raspberry pi   
The raspberry pi struggles to run such an application due to its limited power so some measurements were taken such as reduce resolution .

### 6.4 Problem Analysis Low FPS on the Raspberry Pi

#### Observation

* Live video from the Pi runs at **≈ 4–7 frames per second (FPS)**.
* After 10-15 minutes it can dip to **≈ 3 FPS**.
* Our target was **≥ 15 FPS** so a fast gun-draw (≈ 200 ms) can’t slip between frames.

#### Why it’s slow (high-level)

| Main cause | What happens |
| --- | --- |
| **Heavy model** | We use the standard **YOLO v5-S** (7 M parameters). A single forward pass on the Pi’s CPU takes ~150 ms. |
| **Extra work around the model** | – Resize each frame 640 × 480 → 224 × 224 – JPEG-encode every frame for streaming – Synchronous HTTP upload when a threat is found |
| **Thermal throttling** | The Pi overheats (> 70 °C) and clocks down, losing another 20–25 % FPS. |

Result: **~160 ms per frame ⇒ 6 FPS (best-case)**.

#### Impact

* Fast weapon movements can occur entirely between frames → might not be detected.
* Alert delay grows to 300 ms+, missing the “real-time” requirement in our spec.

#### Mitigation options

1. **Use a lighter / quantised model**  
   YOLO v5-N (nano) or INT8 TensorRT build – estimated 2-4× faster but it will not give needed performance and accuracy .
2. **Active cooling fan** – prevents clock-down and recovers ≈ 20 % FPS.
3. **Dedicated AI stick (e.g., Coral USB)** – would lift FPS well beyond target but costs extra hardware.

## 6.5 Future Improvements

Although the current system successfully performs behavior and object detection in real time, there are several planned improvements that will enhance its functionality, reliability, and scalability.

6.5.1 Expanding Object Detection Categories

The current object detection model is limited to identifying pistols and knives. In future iterations, we plan to:

* Add detection for other weapon types, including rifles, bats, and explosive devices.
* Improve training datasets with more varied real-world weapon images.
* Include non-lethal threat detection (e.g., tasers or pepper spray) where applicable.

This will improve the system's usefulness across different environments such as schools, airports, and public events.

6.5.2 Night Vision and Low-Light Optimization

Low-light environments currently reduce detection confidence. To improve performance in such conditions, we aim to:

* Upgrade the camera to a version with infrared (IR) night vision.
* Integrate IR LEDs to assist with visibility during nighttime or in dark areas.
* Apply real-time brightness enhancement techniques (e.g., histogram equalization) to improve visibility during processing.

These upgrades will ensure the system functions reliably 24/7, regardless of lighting conditions.

Chapter 7: Use Cases

The proactive AI security system is designed to be versatile and adaptable across a range of real-world environments. Below are several key use cases where the system can be highly effective:

1. Smart Homes

In modern smart home setups, this system can serve as an added layer of security by:

* Monitoring entry points like doors, garages, or balconies.
* Detecting the presence of intruders carrying weapons or exhibiting suspicious behavior.
* Sending alerts to homeowners instantly, even when they’re away.
* Optionally integrating with smart locks, alarms, or lights to trigger defensive actions.

This offers peace of mind and a proactive response system that goes beyond basic motion detection.

2. Retail Stores and Supermarkets

Robberies or aggressive customer incidents are unpredictable and often escalate quickly, especially at cash registers. In such cases:

* The system can monitor cashier areas and entrances.
* If a weapon is detected or erratic behavior is observed (e.g., shouting, pacing), it can send a silent alert.
* Notifications could go directly to the store manager, security personnel, or even local law enforcement, depending on the configuration.

This helps protect employees who may not be able to manually trigger an alarm during a crisis.

3. Offices and Corporate Buildings

* Detects unauthorized access or individuals acting suspiciously in sensitive areas (like server rooms or restricted floors).
* Supports visitor tracking and alerting during unusual after-hours behavior.

4. Schools and Universities

* Can monitor hallways or entrances for dangerous objects.
* Alerts school staff instantly in the case of a security threat, potentially preventing violent incidents before escalation.

5. Public Spaces and Parking Lots

* In open environments like parks, parking areas, or outside malls, the system can be used with multiple cameras to detect threats across wide zones.
* It can act as an early warning system for suspicious loitering or crowd behavior.

Conclusion

This graduation project successfully delivered a working prototype of a proactive AI-based security system capable of detecting abnormal behaviors and dangerous objects in real time. Unlike traditional reactive systems, our solution emphasizes real-time Detection, leveraging computer vision, machine learning, and edge processing using affordable and accessible hardware such as the Raspberry Pi 4.

The implementation included two core AI models—one for object detection (pistols and knives) and another for behavior recognition (loitering, running, panic). These models were trained, optimized, and deployed locally using TensorFlow Lite to ensure low latency and efficient performance. The system also includes a simple, secure, and user-friendly web interface developed with Flask, with authentication powered by Firebase using custom email and password credentials, in addition to new simple notification system done by creating telegram bot to send real time notification when threat is detected.

Testing in a controlled environment showed that the system operates smoothly, responds quickly, and successfully detects relevant threats. Although real-world deployment is still pending, the foundation built during GP2 has demonstrated the system’s core capabilities and future potential.

Use cases for this system range from smart homes and retail stores to schools, offices, and public areas—where rapid detection and notification can make a critical difference. This positions our system as a flexible and scalable security solution that can adapt to many types of environments.

Overall, this project provided an excellent opportunity to apply our knowledge of embedded systems, AI, and software development to build a real-world application that can potentially improve safety and security in meaningful ways.

# References

1. Balantrapu, S. S. (2024). AI for predictive cyber threat intelligence. International Journal of Management Education for Sustainable Development, 7(7), 1–28.
2. Brown, T., & Miller, S. (2020). Advancements in computer vision for security applications. Springer.
3. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 27.
4. Gunning, D., & Aha, D. (2019). DARPA’s explainable artificial intelligence (XAI) program. AI Magazine, 40(2), 44–58.
5. Jin, B., Kim, E., Lee, H., Bertino, E., Kim, D., & Kim, H. (2024). Sharing cyber threat intelligence: Does it really help? In Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS).
6. Johnson, D., & Lee, H. (2021). AI and the future of proactive security systems: A comprehensive review. Journal of Security Technology, 12(4), 456–478.
7. Market Insights. (2023). Global AI-powered security systems market analysis. Retrieved from https://www.marketinsights.com
8. McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017, April). Communication-efficient learning of deep networks from decentralized data. In Artificial Intelligence and Statistics (pp. 1273–1282). PMLR.
9. Sharma, N. V., Aggarwal, G., & Sharma, S. (2021, March). RETRACTED: Performance study of Snort and Suricata for intrusion detection system. In IOP Conference Series: Materials Science and Engineering, 1099(1), 012009. IOP Publishing.
10. Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5), 637–646.
11. Tiwari, A., Saraswat, S., Dixit, U., & Pandey, S. (2022, March). Refinements in Zeek intrusion detection system. In 2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS) (Vol. 1, pp. 974–979). IEEE.
12. Yadav, G., Agarwal, N., & Singhal, T. (2024). A comprehensive blockchain and ML-based model for enhanced intrusion detection. SSRN.
13. Zhang, J., Pan, L., Han, Q. L., Chen, C., Wen, S., & Xiang, Y. (2021). Deep learning-based attack detection for cyber-physical system cybersecurity: A survey. IEEE/CAA Journal of Automatica Sinica, 9(3), 377–391.
14. Zhang, Y., Wang, Q., & Chen, L. (2023). Challenges in real-time threat detection with AI systems. Proceedings of the International Conference on AI and Security, 45–58.