**A logo for a university

Description automatically generated**

**Title: Webcam Application for High-Risk Independent Resident**

**Author: Tianluan Lin**

***Course:* Computer Science**

**Project code: PJE40**

**Supervisor: Ioannis Kagalidis**

**May 2025**

Please tick

|  |  |
| --- | --- |
| ❏ | I give permission for my project to be published in the University library and/or be made available to other students as examples of previous work. (optional). |
| ❏ | I confirm that I have read and understood the University Rules in respect of plagiarism and student misconduct. |
| ❏ | I declare that this work is entirely my own. Each quotation or contribution cited from other work is fully referenced. |
|  |  |

Date: \_\_\_\_\_6th May 2025\_\_\_\_\_\_\_

# Abstract

The increasing vulnerability of older adults living independently necessitates the development of intelligent monitoring systems capable of detecting potential health and environmental risks. This project presents a comprehensive solution as a webcam-based application designed to identify falls and issue alerts in real-time. By utilising convolutional neural networks (CNNs) for pose estimation and integrating live environmental data—including temperature and humidity—the system provides both situational and contextual risk assessments.

The methodology includes collecting RGB and depth images, designing and training a coevolutionary neural network for pose classification, and developing a user-facing web interface to monitor and communicate alerts. The application is capable of recognising six common states: standing, sitting, lying, bending, crawling, and absence from the frame, with particular attention to detecting the 'lying' pose as a critical risk indicator. In the event of a fall, an automated email notification system contacts a pre-configured recipient.

Comprehensive testing and evaluation demonstrate that the model performs well in real-world conditions, achieving high accuracy in classification and robustness across variable environments. The system offers practical utility for carers and healthcare professionals and contributes to public health by mitigating risks associated with unattended falls, extreme weather, and isolation. The project illustrates the successful integration of AI, web technologies and environmental sensing in a cost-effective, scalable monitoring solution for high-risk residents.

# Acknowledgements

I would like to thank my supervisor Ioannis Kagalidis for providing support and guidance and through my time at university for the past three years and during this project. I also would like to thank my friends and family for supporting and motivating me throughout this year.

Word Count:

# Table of Contents

[Abstract 1](#_Toc197412606)

[Acknowledgements 1](#_Toc197412607)

[Table of Contents 2](#_Toc197412608)

[List of Tables 5](#_Toc197412609)

[List of Figures 6](#_Toc197412610)

[Chapter 1 – Introduction 7](#_Toc197412611)

[1.1 Background 7](#_Toc197412612)

[1.2 Project Aims and Objectives 7](#_Toc197412613)

[1.3 Scope and Constraints 8](#_Toc197412614)

[1.4 Ethics Review 8](#_Toc197412615)

[1.5 Report Structure 8](#_Toc197412616)

[Chapter 2 – Literature Review 9](#_Toc197412617)

[2.1 Introduction 9](#_Toc197412618)

[2.2 Fall Detection in High-Risk Residents 9](#_Toc197412619)

[2.3 Computer Vision and Pose Estimation Techniques 10](#_Toc197412620)

[2.4 CNN in Human Activity Recognition 11](#_Toc197412621)

[2.5 Environmental Sensing in High-Risk Resident Monitoring 12](#_Toc197412622)

[2.6 Research Gaps and Design Justification 14](#_Toc197412623)

[2.7 Summary 14](#_Toc197412624)

[Chapter 3 – Methodology 15](#_Toc197412625)

[3.1 Introduction 15](#_Toc197412626)

[3.2 Overall System Development Approach 15](#_Toc197412627)

[3.3 Data Acquisition and Preprocessing 15](#_Toc197412628)

[3.3.1 Data Source 15](#_Toc197412629)

[3.3.2 Preprocessing Steps 15](#_Toc197412630)

[3.4 Pose Estimation Model Integration 16](#_Toc197412631)

[3.5 Fall Detection Logic 16](#_Toc197412632)

[3.6 Environmental Advisory System 17](#_Toc197412633)

[3.7 Tools and Technologies 17](#_Toc197412634)

[3.8 Constrains and Limitations 18](#_Toc197412635)

[3.9 Summary 18](#_Toc197412636)

[Chapter 4 – Requirements 19](#_Toc197412637)

[4.1 Introduction 19](#_Toc197412638)

[4.2 Stakeholders and Use Case Context 19](#_Toc197412639)

[4.3 Functional Requirements 19](#_Toc197412640)

[4.4 Non-Functional Requirements 20](#_Toc197412641)

[4.5 Future Requirements 20](#_Toc197412642)

[4.6 Constraints and Assumptions 20](#_Toc197412643)

[4.7 Summary 21](#_Toc197412644)

[Chapter 5 – Design 22](#_Toc197412645)

[5.1 Introduction 22](#_Toc197412646)

[5.2 System Overview 22](#_Toc197412647)

[5.3 Software Architecture 23](#_Toc197412648)

[5.4 Data Flow Design 23](#_Toc197412649)

[5.5 Key Component Design 24](#_Toc197412650)

[5.5.1 Pose Estimation Module 24](#_Toc197412651)

[5.5.2 Fall Detection Logic 24](#_Toc197412652)

[5.5.3 Environmental Advisory Module 25](#_Toc197412653)

[5.5.4 Email Alert Module 25](#_Toc197412654)

[5.5.5 Web Interface 26](#_Toc197412655)

[5.6 Design Rationale and Trade offs 26](#_Toc197412656)

[5.6.1 Pose Estimation 26](#_Toc197412657)

[5.6.2 Local Execution 28](#_Toc197412658)

[5.6.3 Limitations Considered 28](#_Toc197412659)

[5.7 Summary 28](#_Toc197412660)

[Chapter 6 – Implementation and Testing 29](#_Toc197412661)

[6.1 Introduction 29](#_Toc197412662)

[6.2 Development Environment 29](#_Toc197412663)

[6.3 System Setup and Folder Structure 29](#_Toc197412664)

[6.4 Key Implementation Features 30](#_Toc197412665)

[6.4.1 Real-Time Pose Estimation 30](#_Toc197412666)

[6.4.2 Fall Detection Module 30](#_Toc197412667)

[6.4.3 Environmental Advisory Module 31](#_Toc197412668)

[6.4.4 Web Interface 32](#_Toc197412669)

[6.5 Challenges and Solutions 32](#_Toc197412670)

[6.6 Summary 33](#_Toc197412671)

[Chapter 7 – Testing and Evaluation 34](#_Toc197412672)

[7.1 Introduction 34](#_Toc197412673)

[7.2 Testing Methodology 34](#_Toc197412674)

[7.3 Functional Testing 34](#_Toc197412675)

[7.3.1 Pose Estimation Accuracy 34](#_Toc197412676)

[7.3.2 Fall Detection 35](#_Toc197412677)

[7.3.3 Environmental Advisory Accuracy 35](#_Toc197412678)

[7.3.4 Email Alert System 35](#_Toc197412679)

[7.3.5 Web Interface Functionality 35](#_Toc197412680)

[7.4 Performance Evaluation 36](#_Toc197412681)

[7.5 Limitations 36](#_Toc197412682)

[7.6 Summary 36](#_Toc197412683)

[Chapter 8 - Conclusions and Future Work. 37](#_Toc197412684)

[8.1 Introduction 37](#_Toc197412685)

[8.2 Summary of Achievements 37](#_Toc197412686)

[8.3 Critical Reflection 38](#_Toc197412687)

[8.4 Limitations 38](#_Toc197412688)

[8.5 Future Work 38](#_Toc197412689)

[8.6 Final Conclusion 39](#_Toc197412690)

[References 40](#_Toc197412691)

[Appendix A 44](#_Toc197412692)

[Appendix B 45](#_Toc197412693)

[Appendix C 46](#_Toc197412694)

[Appendix D 53](#_Toc197412695)

# List of Tables

[Table2.2.1: Comparison of Fall Detection Technologies 10](#_Toc197405594)

[Table 3.6.1: Environmental Parameters and advisory outputs 17](#_Toc197405596)

[Table 3.7.1: Table of Tool and Libraries 18](#_Toc197405597)

[Table 5.3.1: Table of Files Architecture 23](#_Toc197405598)

[Table 5.5.3: Example Environmental Inputs and Corresponding Health Advisory Messages Generated by the Neural Network 26](#_Toc197405599)

[Table 5.6.1.1: Custom CNN pose estimation model training loss value and validation loss values 28](#_Toc197405600)

[Table 6.2.1: Hardware and software used for developing 30](#_Toc197405601)

[Table 7.2.1 Summary of Testing Methods Applied per Module 35](#_Toc197405602)

# List of Figures

[Figure 2.3.1: An example of multi-person poses estimation (Cao et al., 2019) 11](#_Toc197408243)

[Figure2.3.2: An example of using RGB-camera and a depth camera for poses estimation (Adhikari et al., n.d.) 11](#_Toc197408244)

[Figure 2.4.1: Different Activation Functions (Jadon, 2022) 12](#_Toc197408245)

[Figure 2.4.2: The Architecture of Coevolutionary Neural Network (Gurucharan, 2020) 12](#_Toc197408246)

[Figure 2.5.1: Diagram of Environmental based advisory logic for the Monitor Application 13](#_Toc197408247)

[Figure 2.5.2: Diagram of Heat Index (NOAA) 13](#_Toc197408248)

[Figure 3.3.2.1: Diagram of Image Processing Before feed into the Pose Estimation 15](#_Toc197408249)

[Figure 3.5.1: Diagram of Fall Detection Flow 16](#_Toc197408250)

[Figure 5.2.1: High-Level Architecture of the Monitor Application System 23](#_Toc197408251)

[Figure5.3.1: Diagram of Software Module Interaction 23](#_Toc197408252)

[Figure 5.5.2: Flowchart Representing Rule-Based Fall Detection Logic 25](#_Toc197408253)

[Figure 5.5.5.1 Web Interface 26](#_Toc197408255)

[Figure 6.3.1: Diagram for folder structure 30](#_Toc197408256)

[Figure 6.4.2.1: Fall Detection String Output 31](#_Toc197408257)

[Figure 6.4.3.1: Example Inputs and Advisory Output 32](#_Toc197408258)

[Figure 6.4.4.1: Web Interface 32](#_Toc197408259)

[Table 7.3.3.1: Compared of Labeled result and output from neural network. Green means correct, red means incorrect output. 35](#_Toc197408260)

# Chapter 1 – Introduction

## 1.1 Background

Traditional healthcare and monitoring systems have long relied on human supervision and warden. Even in professional environments such as nursing homes, constant individual supervision is not always feasible. Furthermore, there have been reported cases of neglect or even abuse in residential care facilities (Agaliotis M, 2025). On the other hand, many high-risk individuals, including elderly people, live alone without immediate access to assistance. In the UK alone, there are 3.3 million pensioners who live independently (Office for National Statistics, 2023) and are at risk of possible injuries or unconsciousness at any moment. The possibility of such an event would only increase with age. Delays in medical response due to undetected falls can lead to severe health deterioration or even death (WHO, 2021).

Moreover, as extreme weather conditions become more frequent than ever before, vulnerable individuals who live on their own may refuse to use heating or cooling appliances, often due to cost or personal habits. This has caused casualties of illnesses related to temperatures like hypothermia or dehydration every year. A report has shown that among the death decedents succumbing to hypothermia, 67.7% of the deaths were more than 60 years old (Hellerstedt, J., 2021).

The demand for technologies that assist high-risk individuals who are living independently is increasing with more reliable and more powerful tools and equipment as the technologies have improved in terms of computational power and networking for the past decades. This project aims to address that need by developing a prototype webcam monitor application system for detecting whether a person has fallen using computer vision techniques and machine learning. By using a camera, the system captures images at a rate of two frames per second, processes them using a Convolutional Neural Network (CNN), and classifies the user’s pose to determine whether a fall has occurred.

If a fall is detected, the system will automatically notify a pre-configured contact, enabling a timely response. Falls are one of the leading causes of injury-related death among adults in many countries. Additionally, another module provides health-related feedback based on temperature and humidity, helping to ensure that the resident is living in safe conditions.

## 1.2 Project Aims and Objectives

The core objective is to design and implement a real-time fall detection system using images captured by a webcam and analysed via a CNN-based pose estimation model. The secondary objective is to provide environmental advice based on sensor data.

Primary Objectives:

* Webcam can capture images at a consistent frame rate​.
* Train and deploy a CNN to classify human poses.
* Determine if a set of poses indicates a fall.
* When a fall is detected, the system can pass the message to relevant contactor.

Secondary Aspirations: (Implementation is limited by time and complexity)

* The Monitor Application can provide a text-based advisory system based on environmental conditions.
* The Monitor Application can record daily activity time.
* The system can keep a log of medication intake times.
* Personalised alerts or reminders based on behavioural patterns.

## 1.3 Scope and Constraints

This project is developed under academic constraints and does not extend to clinical or regulatory compliance for medical-grade devices. The system is intended as a prototype to demonstrate feasibility rather than a commercialised product. While the application is designed to operate on consumer-grade hardware (laptop and webcam), and is not designed for multi-camera setups or advanced motion tracking.

Due to the incident during training of the model, this application is tested and trained on a CPU-based system. Hence, the CNN model complexity and the speed of image processing are limited.

## 1.4 Ethics Review

This monitoring system does not store or process personally identifiable information beyond what is necessary for fall detection and alert generation. The project has received ethics approval (Reference: TETHIC-2025-110551) from the Technology Faculty Ethics Committee, confirming that it involves no physical, psychological, or environmental risks.

## 1.5 Report Structure

This report is structured as follows:

Chapter 2 - Literature Review: Critical analysis of relevant studies in the fall detection, pose estimation, and ambient health monitoring.

Chapter 3 – Methodology: Justification of the development approach, neural network design, and system structure.

Chapter 4 – Requirements: Functional and nonfunctional requirements.

Chapter 5 – Design: Detailed system architecture, data flow, and model selection.

Chapter 6 – Implementation: Description of key components including webcam interfaces, CNN training, and alert mechanism.

Chapter 7 – Testing and Evaluation: Verification of results against requirements and performance benchmarks.

Chapter 8 – Conclusion and Future Work: Summary of findings and potential improvements.

# Chapter 2 – Literature Review

## 2.1 Introduction

This chapter presents a critical review of the technologies and techniques that support fall detection in high-risk individuals, particularly those living alone. The main areas of discussion include:

1. Sensor-based fall detection mechanisms – List the current fall detections and their disadvantages.
2. Pose estimation through computer vision – Discuss a few popular approaches for getting the pose of a person from the image.
3. The use of CNN in human activity recognition – Discuss what is CNN and how will this monitor application system’s CNN work.
4. The effect of the environmental monitoring – Why does the monitor application also need to evaluate the environment for high-risk individuals.

Outline of each technique of these topics will be discussed and the key points and basic requirements of this monitor application will be drawn. The limitation of existing systems will also be discussed, leading to rationale for using RGB image-based CNN classification in the proposed solution.

## 2.2 Fall Detection in High-Risk Residents

Falls are one the most serious health threats to older adults, commonly resulting in fractures, trauma, or ling-term hospitalisation. According to the U.S. Centres for Disease Control and Prevention (CDC), falls are a leading cause of traumatic brain injuries and prolonged recovery demands on healthcare systems (CDC, 2019).

Currently, there are hardware devices have been developed or under developing to detect fall of a person. These fall detection method each has its own advantages and disadvantages.

1. Wearable Sensors: Devices which are designed to be worn on the residents on different part of the body. Some are just like watches on the arm, some of them need to be placed on the main body. These types of devices using accelerometer to identified the motion and the high-risk residents’ condition. However, these devices must be worn at all time and can produce inaccurate results if the devices are worn at different places or shift position on the body (Singh et al., 2020). This limit the place and scenario of the device being used.
2. Ambient Sensors: This type of devices collect the information from the environment to identifies falls.
3. Acoustic Sensors: Classify the sound of fall as well as other sounds from the surrounding. However, this type of device’s accuracy is affected by background noise. Even the presence of large pets will lead to inaccurate results.
4. Floor Pressure Sensors: This type of sensor is still under development in the lab environment. It detects vibrations from the fall impact. However, the performance varies based on floor material and body weight. Another approach is to put the pressure sensor around the bed to detect the fall around the bed only (Huan-Wen Tzeng, 2010). This type of fall detection has very limit of use in terms of the area it will be able to cover.
5. Infrared, Radar, Ultrasonic Sensors: These types of sensors offer contactless monitoring by placing them in a carefully selected place. However, these require precise setup and specialised equipment that are not typically common in households.
6. Image Sensors: These use cameras to visually detect falls. While accurate, many reply on cameras with special functions like 3D camera or depth camera, which are expensive and not typically common to our ordinary life. Additionally, privacy concerns arise when capturing live video streams or photos has risk of leak to the social media.

| **Types of Fall Detection System** | **Detection Methods** | **Issues** |
| --- | --- | --- |
| Wearable Sensors | Using accelerometer to identify the motion | The user needs to wear this type of sensor all the time and place it around the correct part of the body |
| Acoustic Sensors | Classify the sounds of all fall | Could be easily affected by background noise |
| Floor Pressure Sensors | Either uses pressure sensor to detect the fall, or detect the vibration from the impact of fall | This field is still under research and some types of the floor pressure sensor need to be placed on the entire floor which increases the cost |
| Infrared Sensors | Uses infrared to detect the change of motion when a fall happened with carefully placed equipment | Need to be carefully select the places and specialised equipment is required |
| Radar Sensors | Uses radar to detect the change of motion when a fall happened with carefully placed equipment | Need to be carefully select the places and specialised equipment is required |
| Ultrasonic Sensors | Uses ultrasonic wave to detect the change of motion when a fall happened with carefully placed equipment | Need to be carefully select the places and specialised equipment is required |
| Image Sensors | Uses visual to detect falls directly | One of the most common types of fall detection method. However, some of them requires a depth camera as well as a normal camera |

Table2.2.1: Comparison of Fall Detection Technologies

## 2.3 Computer Vision and Pose Estimation Techniques

Pose estimation is a core computer vision task aimed at localising key body joints in images or video frames. Well-established frameworks such as OpenPose, BlazePose, and MediaPipe can reconstruct a human skeleton to support applications such as motion tracking, activity recognition, and healthcare monitoring. These models typically leverage multiple camera angles or RGB-D (colour and depth) inputs to infer human posture in either 2D or 3D space.

However, while these state-of-the-art systems offer high accuracy, they also present several practical limitations. Many require high-resolution cameras or depth sensors, which may not be readily available in a low-cost or domestic monitoring setup. Moreover, issues such as occlusion, variable lighting, and limited computing power can significantly degrade performance in real-time scenarios (Mehta et al., 2016). These requirements pose challenges for the intended use case of this project, which is to provide an efficient, real-time monitoring solution using standard webcam input.



Figure 2.3.1: An example of multi-person poses estimation (Cao et al., 2019)



Figure2.3.2: An example of using RGB-camera and a depth camera for poses estimation (Adhikari et al., n.d.)

To overcome these constraints, a custom CNN-based pose estimation model was developed as the primary approach in this project. Nonetheless, acknowledging the possibility of limited training data, potential overfitting, and unforeseen performance issues, a contingency model was also considered. In this context, the Ultralytics YOLO11n-pose model was selected as a backup solution. This model was chosen for several reasons:

* It is lightweight and optimised for edge devices and CPU-only environments.
* It provides real-time pose detection using only RGB input, which aligns with the hardware limitations of the system.
* It is well-documented and easy to deploy, thereby allowing for rapid integration in the event that the primary model underperforms.
* It includes support for bounding boxes and key point detection, which could be repurposed for fall detection with minimal retraining.

By incorporating this alternative, the project ensures resilience and robustness, maintaining its real-time capabilities even under reduced computational conditions or data constraints.

## 2.4 CNN in Human Activity Recognition

Convolutional Neural Network (CNN) is a class of deep learning models designed for image-based pattern recognition. CNN have been widely adopted in human activity recognition (HAR) due to their ability to detect statical features from raw image data. A typical CNN architecture includes:

* Input Layer: Fixed-size RGB image will be received at this layer at beginning.
* Convolutional Layers: This layer applies filters to extract visual features such as edges, shapes, or patterns.
* Activation Layers: Using functions to introduce non-linearity into the model. Typical activation functions include ReLU, SoftMax, and Tanh etc.

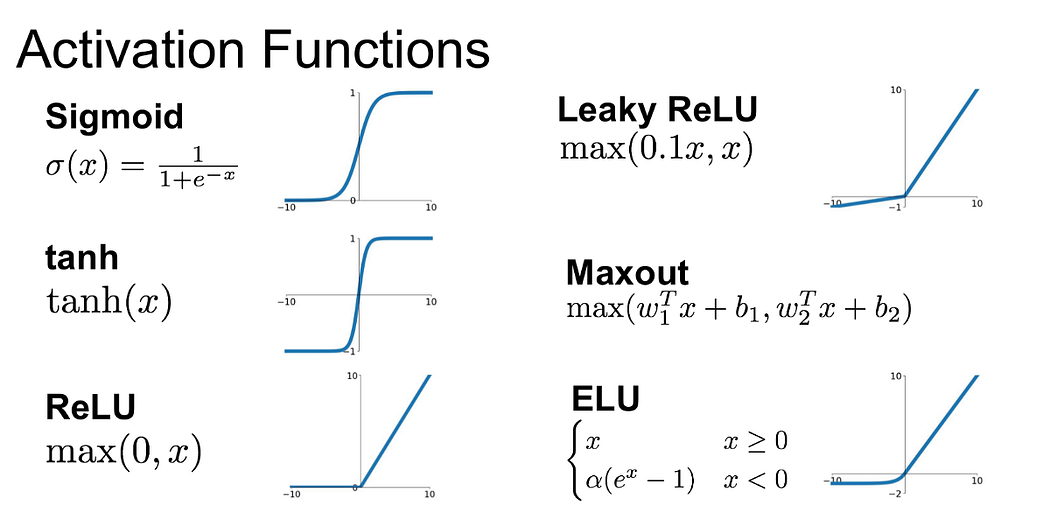


Figure 2.4.1: Different Activation Functions (Jadon, 2022)

* Pooling Layers: This layer reducing the sampling rate to reduce overfitting and training time. Two typical pooling layers are Max Pooling and average Pooling.
* Fully Connected Layers (Output Layers): This layer will combine features to classify the input image.

Currently, many HAR software use CNN as a primary structure to build their models. Back to 2016, researches in this filed has already developed a “binary sensor network” enhanced by CNN. It was able to identify daily human activities that claimed accuracy rate of at least 94.7% with test samples (Liu et al., 2016)

In the monitor application system, we will be trying to train a CNN for HAR with a typical CNN structure. However, due to hardware constrain, a degraded, simpler CNN with less channels and convolutional layers were considered. Given the limitations with the custom CNN model, the monitor system will adopt the pre-trained YOLO11n-pose model. This model is developed by Ultralytics was able to detect 17 human key points and make pose estimation based on the key points (Ultralytics). Allowing our monitor application system to have a more reliable core HAR system to achieve the primary objective.

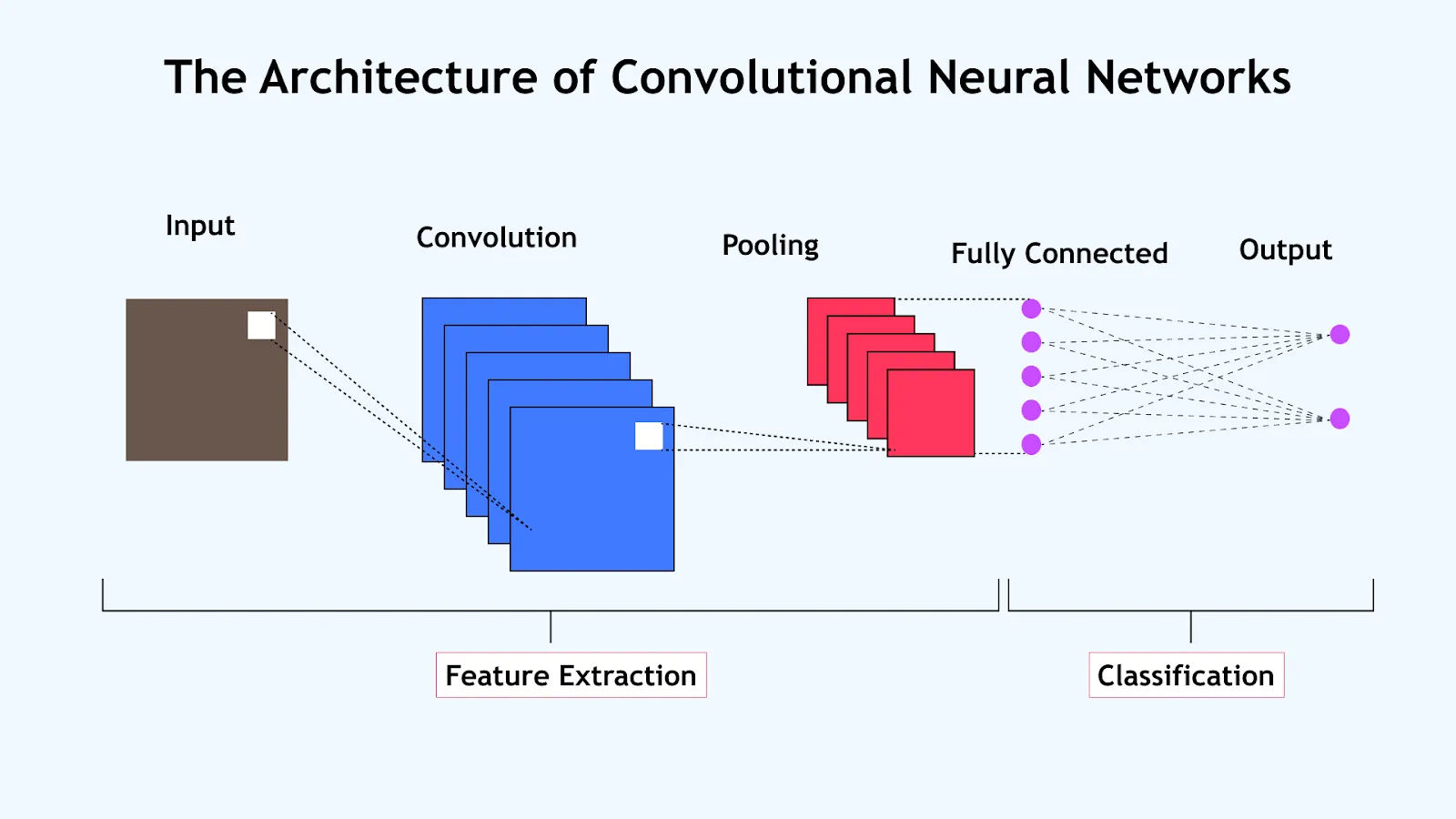


Figure 2.4.2: The Architecture of Coevolutionary Neural Network (Gurucharan, 2020)

## 2.5 Environmental Sensing in High-Risk Resident Monitoring

Temperature and humidity are critical environmental parameters that significantly influence the health and wellbeing of elderly individuals. Particularly for high-risk independent residents—those who live alone and may have chronic conditions—monitoring ambient conditions plays a pivotal role in preventing avoidable health incidents. Extremes in temperature have well-documented physiological consequences. Excessive heat can result in dehydration, heat exhaustion, or heat stroke, while cold environments increase the likelihood of hypothermia, especially among older adults with limited mobility or reduced thermoregulatory capacity (World Health Organization, 2018).

The vulnerability of this demographic is further intensified by pre-existing health conditions, polypharmacy, and the social isolation often associated with old age. Stella et al. (2020) observed an increased risk of heat-induced neurological complications during periods of isolation, particularly during the COVID-19 pandemic. Similarly, Ncongwane et al. (2021) provided a continent-wide review highlighting the disproportionate burden of heat stress on elderly populations in Africa, advocating for early detection and response mechanisms.

Numerous studies point to the heightened incidence of heat-related medical emergencies among the elderly. Ueno et al. (2021) found that older adults are significantly more likely to require ambulance transport during periods of extreme heat. Furthermore, Gamboa et al. (2023) highlighted that certain medications commonly prescribed to elderly patients can exacerbate heat sensitivity, leading to more frequent and severe complications. Wu et al. (2023) also observed a notable increase in emergency department admissions of elderly patients during heatwaves, strengthening the case for proactive environmental monitoring.

In response to these findings, the proposed monitoring system integrates a lightweight sensing module designed to collect real-time temperature and humidity data. This data is evaluated using a rule-based advisory mechanism that issues precautionary alerts when environmental parameters exceed predefined thresholds. By offering basic, context-aware guidance, the system aims to empower elderly residents or their caregivers with timely and relevant information to mitigate health risks.

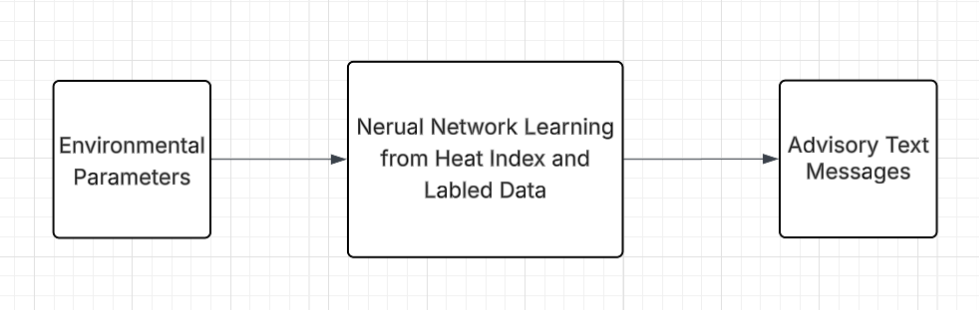


Figure 2.5.1: Diagram of Environmental based advisory logic for the Monitor Application

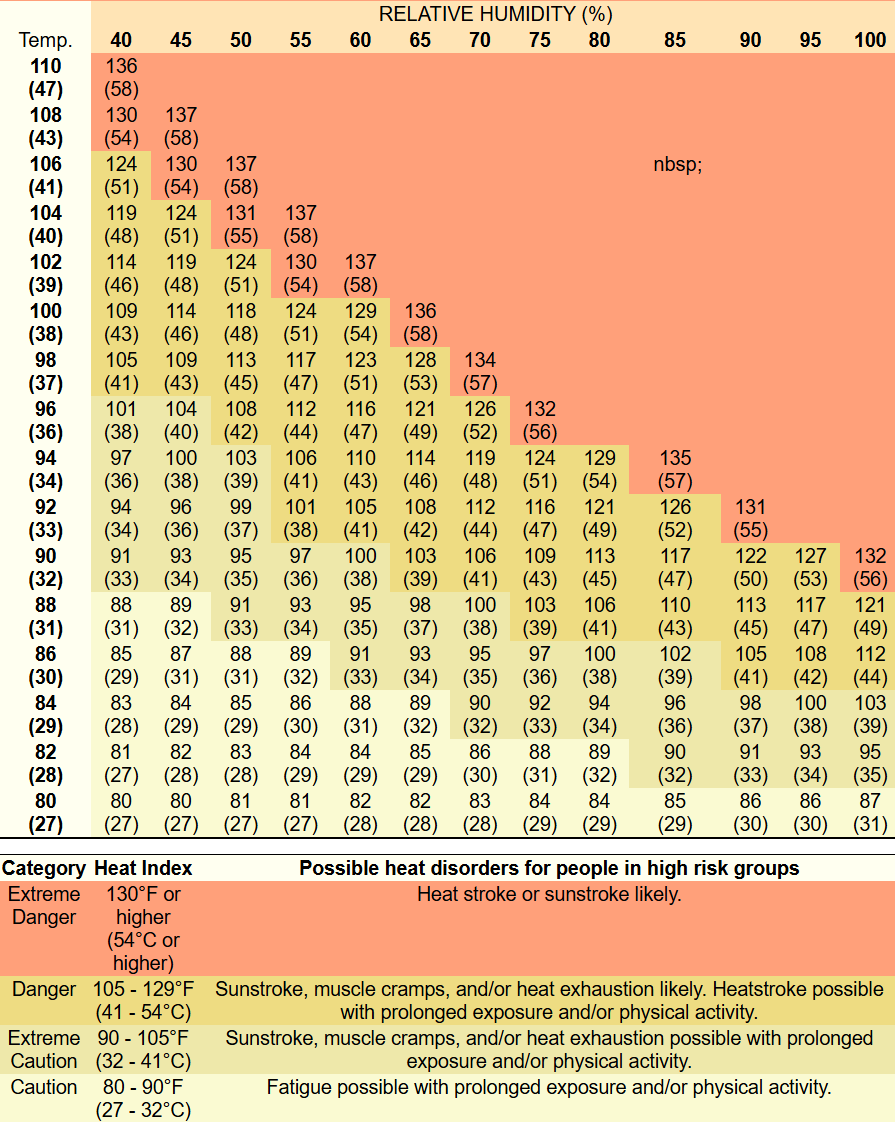
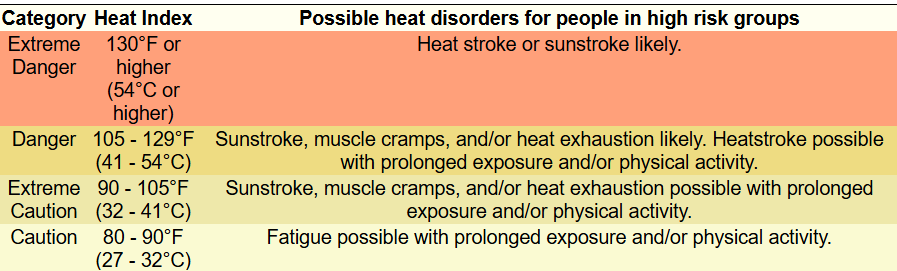


Figure 2.5.2: Diagram of Heat Index (NOAA)

A key component of this system is the utilisation of the *Heat Index*, developed by the National Oceanic and Atmospheric Administration (NOAA). The Heat Index, also referred to as the “apparent temperature”, is a metric that combines ambient air temperature with relative humidity to reflect how hot it feels to the human body. This is particularly relevant to elderly individuals, whose bodies often have a diminished ability to regulate temperature through perspiration.

By applying the Heat Index formula, the system interprets environmental data more meaningfully than raw temperature readings alone. For instance, a temperature of 30°C with 70% humidity may feel significantly warmer and more hazardous than the same temperature with lower humidity. Thresholds derived from the NOAA Heat Index guidelines are used to categorise conditions into comfort zones (e.g., normal, caution, extreme caution, danger). When necessary, the system displays tailored textual advice, such as encouraging hydration, suggesting ventilation, or alerting to high-risk heat conditions.

This implementation approach balances computational efficiency and contextual awareness, ensuring suitability for low-cost, resource-constrained environments such as independent residences. Unlike high-end environmental monitoring systems that may rely on an array of sensors or cloud analytics, the proposed solution is decentralised and autonomous, allowing for immediate feedback without external dependencies.

In summary, the integration of temperature and humidity monitoring with Heat Index-based evaluation enhances the project’s capacity to address environmental health risks. It ensures that the system is not only responsive to motion-based anomalies such as falls but also proactively attentive to ambient threats that can undermine elderly wellbeing. This holistic approach reflects a modern standard in ambient assisted living (AAL) design and reinforces the project’s alignment with both technical feasibility and public health priorities.

## 2.6 Research Gaps and Design Justification

Many existing fall detection systems have achieved high accuracy. Yet, there are still limitations in each of those systems:

1. Sensor-heavy.
2. ​​Some hardware can only be used for a particular detection method.​
3. Low cost-effective.
4. Rarely integrated.

Adhikari et al. (2017) proposed a CNN-based method using both RGB and depth data for indoor fall detection. However, their system depended on Kinect-like devices and requirement for depth camera. Inspired by his team’s research, this project seeks to replicate its effectiveness using only readily accessible RGB webcams. The aim of this project is to develop a prototype of a system that covers simplicity, affordability, and extensibility that other fall detection application systems unable to achieve.

## 2.7 Summary

This chapter reviewed the evolution of fall detection technologies, pose estimation models, and CNN-based image recognition. Key design decisions, such as using RGB only inputs and avoiding expensive skeleton estimation models and depth camera, are justified through both technical and practical considerations. The literature supports a strong case for the development of a light-weight, real-time, webcam-based solutions to monitor vulnerable individuals living independently.

# Chapter 3 – Methodology

## 3.1 Introduction

This chapter outlines the methodology and approaches to developing the Monitor Application for High-Risk Independent Residents. The system’s core functionality revolves around real-time webcam-based image capture and pose detection. Initially, a custom CNN-based pose detection model was developed. However, due to its low testing accuracy (~33%) and high validation loss (~1.2), this approach was replaced with the pre-trained pose estimation model from Ultralytics YOLO. This transition ensured a more robust and dependable core for the system.

## 3.2 Overall System Development Approach

The core of the monitor application, pose estimation is divided into 3 subsystems: real-time webcam feed, pose estimation and fetch pose to webpage. The application was built using an interactive development methodology for software development. This modular structure allowed for parallel development and the ability to substitute components if technical issues arose, such as the switch from a custom CNN to a pre-trained model.

## 3.3 Data Acquisition and Preprocessing

### 3.3.1 Data Source

The application captures real-time data from a standard consumer-grade webcam. Due to hardware constraints and a focus on accessibility, the system captures images at 2 frames per second (FPS) by taking screenshots rather than streaming video continuously. During testing, frame rates above 4 FPS led to corrupted “snowy picture” displays on the web interface. Although the root cause remains unclear, reducing the capture rate significantly mitigated the issue. Each frame is saved locally and processed for pose estimation.

### 3.3.2 Preprocessing Steps

Captured images are pre-processed before being passed to the pose estimation model module:

* Resizing: Each frame will be resized to 640x640 pixels to match the input for the pose estimation model. A typical webcam resolution is 1920x1080 (Full HD). Although there is no formal requirement for CNN image input to ensure high accuracy, and to balance the image resolution and process time, about 20% of the full HD is set as the default size.

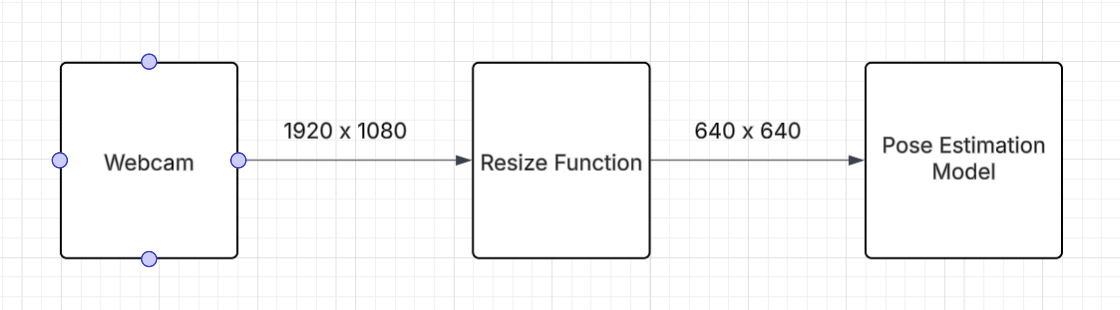
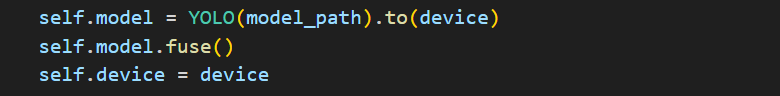


Figure 3.3.2.1: Diagram of Image Processing Before feed into the Pose Estimation

## 3.4 Pose Estimation Model Integration

The initial plan was to develop a custom CNN-based pose detection model to detect 6 basic human poses as output with a lightweight structure and faster process speed with minimal computation power. However, during the development, the final custom CNN model achieved over 1.2 loss on the validation dataset and only a 33% test accuracy rate. Due to the time constraint, a pre-trained model was adopted to ensure the monitor application system meets its functional requirements.

The selected YOLO11n model from Ultralytics was chosen for its balance between accuracy, computational efficiency and real-time inference capability. The model detects 17 human key points to estimate complicated pose analysis (Ultralytics, 2023). Integration was achieved using the Ultralytics Python library, which allows loading pre-trained weights and applying pose detection to static images. The model processes each frame and outputs key points, which are passed to a custom rule-based detector (RuleBasedFallDetector) to infer human posture.



To load the YOLO11n pose estimation model

## 3.5 Fall Detection Logic

The fall detection logic is implemented using the RuleBasedFallDetector module. This module evaluates the spatial relationships between the 17 detected keypoints to classify poses.

* Angel Calculation: Computing angles between key points to detect abnormal postures.
* Position Analysis: Assessing the relative positions of key points to determine if the subject is fallen on the floor.
* Motion Patterns: The model monitors transitions from upright poses (standing/sitting) to prone (fall) over time to confirm a fall.

If a “fall” pose is detected and no fall alert was recently triggered, an email is sent to a designated contact, with a 10-minute cooldown to prevent spamming.

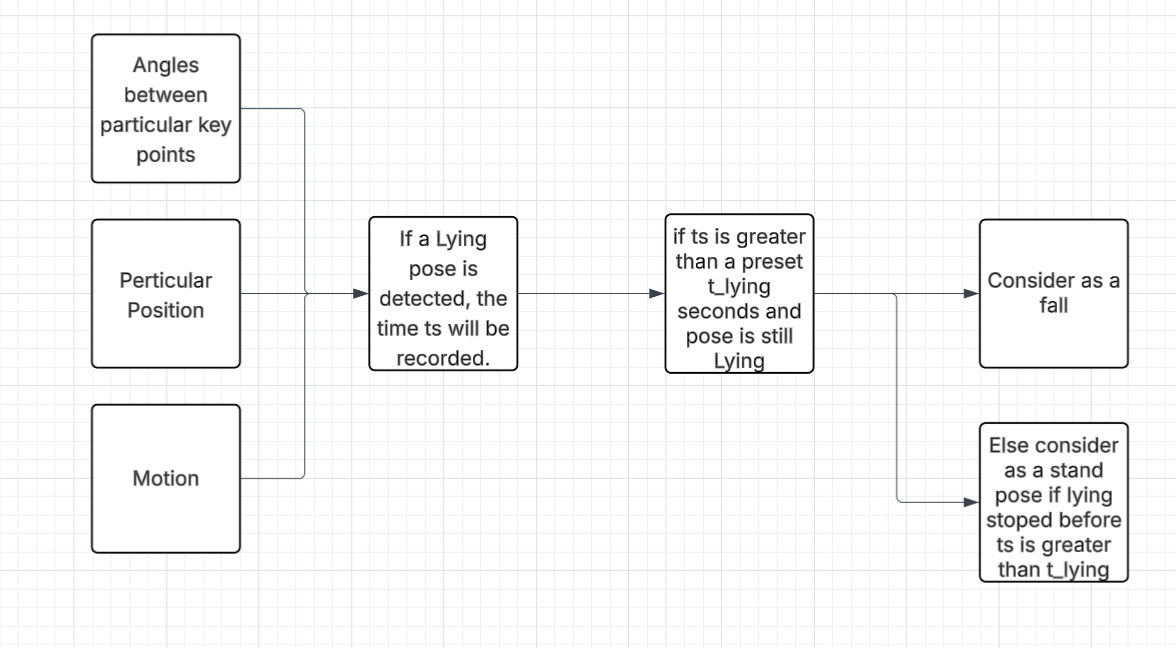


Figure 3.5.1: Diagram of Fall Detection Flow

## 3.6 Environmental Advisory System

Apart from the fall detection system in the monitor application system, the application also monitors environmental parameters (temperature and relative humidity) to provide advisories feedback. OpenWeatherMap API was used to obtain environmental parameters based on the user’s geographical location (latitude and longitude). These values are then passed to a lightweight neural network defined in comfitness.py, which returns text-based health advice. The neural network was trained on labelled environmental data and achieved a validation loss as low as 0.08, ensuring reliable outputs.

| **Temperature (°C)** | **Relative Humidity (%)** | **Advisory Messages** |
| --- | --- | --- |
| Any | < 40 | Too dry, suggest use of moist lotion |
| < 15 | < 40 | Too cold, please turn the heating on. |
| < 12 | Any | Too cold, please wear more clothes and have a cup of hot drink! |
| 15 – 27 | Any | Normal |
| 27 - 43 | Any | (Depends on the Heat Index Level provided by NOAA)   * Caution! High temperature * Extreme caution! High temperature * Danger! High temperature, please avoid outdoor activities |
| > 43 | Any | Extreme danger! High temperature, please stay indoor until sunset! |

Table 3.6.1: Environmental Parameters and advisory outputs

## 3.7 Tools and Technologies

The development utilised the following tools and libraries:

| **Tool/Library** | **Purpose** |
| --- | --- |
| **Python 3.13.1** | Core development |
| **Flask** | Running the web-based interface |
| **OpenCV** | Webcam image capture |
| **Ultralytics** | Pose estimation model (YOLO11n-pose) |
| **NumPy** | Image array manipulation and numerical processing |
| **Matplotlib** | Visualizing training loss (for CNN) |
| **smtplib & email** | Sending alert emails |
| **Threading** | Running parallel tasks for real-time frame handling |
| **OpenWeatherMap API** | Environmental data acquisition |

Table 3.7.1: Table of Tool and Libraries

## 3.8 Constrains and Limitations

The monitor application system’s performance is subject to certain constraints:

* Lighting Conditions: Poor lighting can affect pose estimation accuracy.
* Occlusions: Partial obstruction of a person may prevent accurate pose estimation.
* Image Artifacts: Static images or posters showing people may falsely trigger fall alerts.
* Hardware Limitations: Performance depends on CPU capacity; slower devices may drop frames.
* Internet Dependency: API calls for weather data and email alerts require an active internet connection.

## 3.9 Summary

This chapter presented the step-by-step methodology for building the Monitor Application for High-Risk Independent Residents. It covered the rationale for selecting YOLO11n-pose over a custom CNN, the design of a rule-based fall detection engine, and the integration of an environmental advisory system. The use of lightweight, open-source tools ensures broad compatibility and ease of deployment.

# Chapter 4 – Requirements

## Introduction

This chapter outlines the system requirements for the Monitor Application for High-Risk Independent Residents. These requirements were derived based on user needs, system constraints, literature analysis, and technical testing. The purpose of this application is to passively monitor vulnerable individuals, detect falls using a webcam and a pose estimation model, and provide timely alerts and environmental health advice.

The requirements are categorised into: functional requirements, which define what the system should do, non-functional requirements, which specify performance, reliability, and usability aspects, pose estimation requirements, which elaborate on the conditions and expectations related to real-time posture recognition, constraints and assumptions, which identify limitations of scope, environment, and hardware, and future requirements, which outline potential enhancements to the system.

## 4.2 Stakeholders and Use Case Context

This application primarily targets individuals at high risk of falling, such as elderly people living alone. It is designed to support their independent living by detecting dangerous events in real time.

The primary stakeholders are the high-risk individuals who may fall or experience a health emergency while alone at home most of the time. The secondary stakeholders are caregivers or relatives who need to be notified in the event of a fall. The application operates without the need for physical interaction from the user and minimises privacy concerns by avoiding recording or use of cloud-based data storage.

## 4.3 Functional Requirements

Functional requirements describe the essential operations the system must perform to fulfil its objectives:

* The system should be able to capture images from a webcam at a minimum rate of 2 frames per second (FPS).
* The system should be able to use the pre-trained pose estimation model to estimate human pose from each frame.
* The system should be able to estimate and classify the estimated poses into one of four states: standing, descending, lying, fall.
* The system should be able to prevent spamming of the recipient’s email by enforcing a minimum 10-minute delay between consecutive alerts.
* The system should be able to inform a person using email when a fall is detected.
* The system should be able to allow user to turn off the webcam whenever they wanted to.
* The system must ensure that it runs locally to prevent any potential data leakage.

## 4.4 Non-Functional Requirements

The following requirements have been selected as optional components for the Monitor Application. Implementation of them can improve the functionality of the application, although it is not mandatory and will not affect the application to run.

* The system shall process and classify each frame within 0.5s.
* The application shall require no user interaction after launch.
* Modular code for maintainability and reduce the difficulty of adding new functionality to the Monitor Application.
* The system should be able to run on Windows or Linux with standard CPU as basic requirement.
* The system should be able to continue pose detection while offline.
* The system should be able to retrieve temperature and relative humidity data via an API, based on user geolocation.
* The system should be able to process environmental data through a neural network and generate a text-based advisory message.

## 4.5 Future Requirements

Functional requirements are the essential operations that the system must perform to fulfil its objectives:​

* Activity tracking: This system records an individual’s daily activities with each activity’s frequency and duration. This allows the users to have a better understanding of their health level.
* Recorder and reminder for taking the pill: This is the system that can identify individuals’ actions of taking drugs and send specific pre-set reminders if no evidence of pill-taking is detected.
* Manual override panel for caregivers to confirm or cancel alerts.

## 4.6 Constraints and Assumptions

Due to the nature of the Monitor Application, and limitations in development time and available resources, this application is subject to a series of academic and technical constraints. No dedicated GPU is used for inference, the application must run on a general-purpose laptop. No real high-risk individuals were monitored during development or testing. The accuracy of pose estimation depends on sufficient ambient lighting. The webcam must be correctly positioned and functional to ensure a suitable area is covered and clear images are captured. The system assumes an active internet connection for environmental data retrieval and email.

This system is not intended for use in clinical or emergency settings.

## 4.7 Summary

This chapter defined the detailed requirements of the Monitor Application, including the necessary functional behaviours, performance expectations, hardware and environmental constraints, and future upgrade possibilities. These requirements form the basis for system design decisions covered in the following chapter.

# Chapter 5 – Design

## 5.1 Introduction

This chapter presents the detailed design of the Monitor Application for High-Risk Independent Residents. The design process was informed by the requirements established in Chapter 4 - Requirements, with an emphasis on ensuring modularity, maintainability, and real-time responsiveness.

This chapter outlines the system architecture, data flow, and major components, including the pose estimation module, fall detection logic, environmental advisory system, and user interface. To demonstrate the rationale behind design decisions and trade-offs, this chapter discusses how technical choices support system goals and constraints.

## 5.2 System Overview

The Monitor Application is a lightweight, real-time monitoring solution developed using Python. It captures live webcam input, applies pose estimation to detect potential falls, issues alerts via email when a fall is detected, and provides health-related advice based on ambient conditions. The system is comprised of the following core components:

* Webcam Input Handler
* Rule-Based Fall Detection for Pose Estimation Module (from Ultralytics)
* Environmental Advisory Module
* Email Notification System
* Web Interface (Flask-based)

These components are designed to operate independently but are integrated to function cohesively as a single monitoring system.

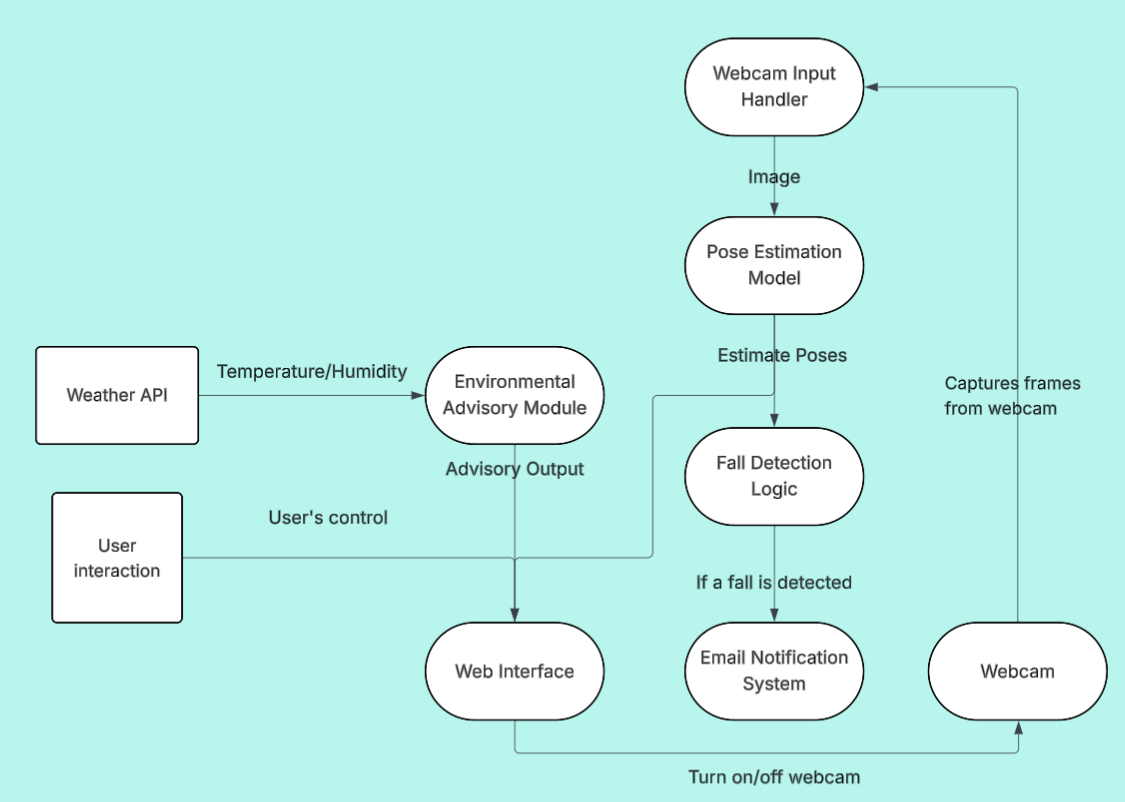


Figure 5.2.1: High-Level Architecture of the Monitor Application System

## 5.3 Software Architecture

The application follows a modular architecture, with responsibilities clearly separated into functional scripts. This enhances maintainability and allows individual components to be updated or replaced without affecting the entire system.

| **Module** | **Description** |
| --- | --- |
| app.py | Main script that runs the system, captures frames, and coordinates modules |
| RuleBasedFallDetector.py | Implements the rule-based fall classification logic and load YOLO11n-pose model |
| comfitness.py | Neural network model for generating environmental health advice |
| Index.html | Contains HTML and frontend logic for displaying results via Flask |

Table 5.3.1: Table of Files Architecture

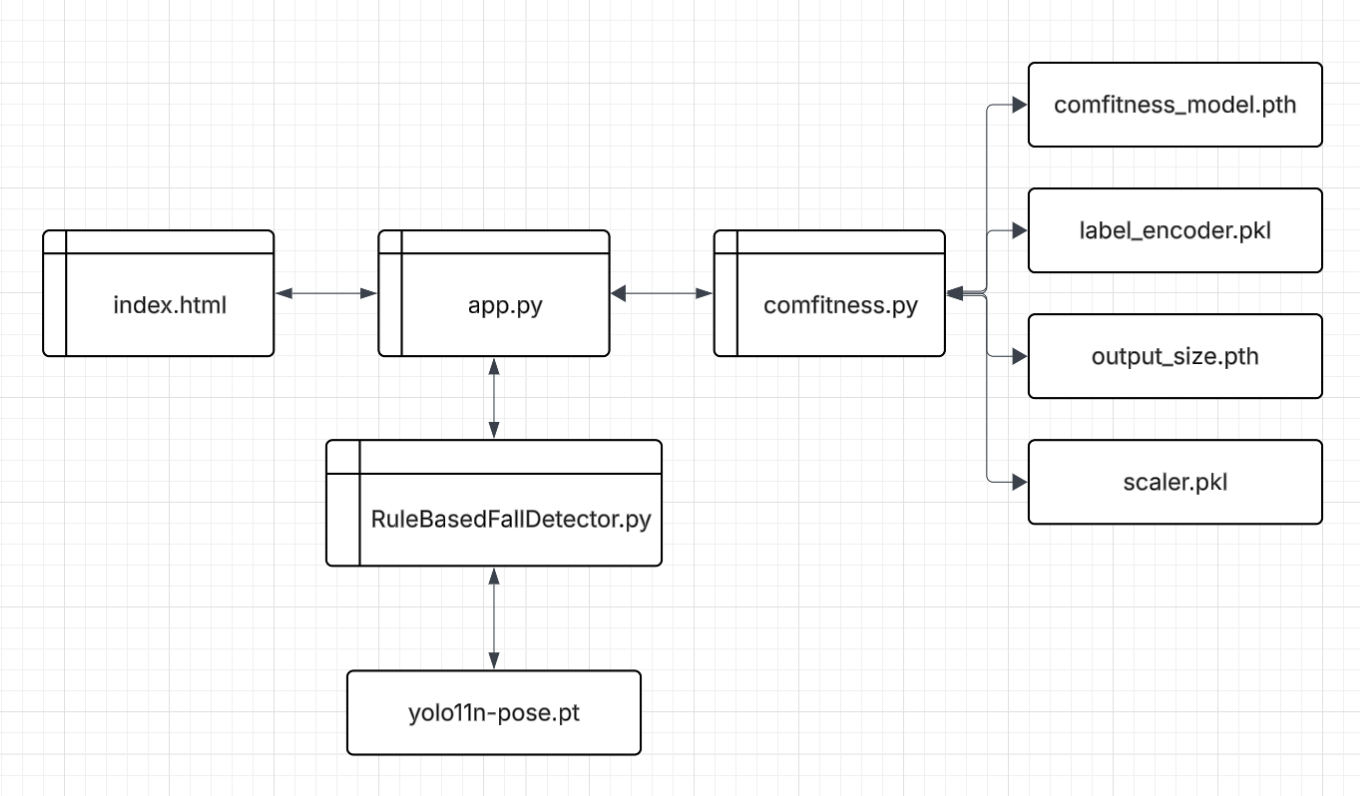


Figure5.3.1: Diagram of Software Module Interaction

## 5.4 Data Flow Design

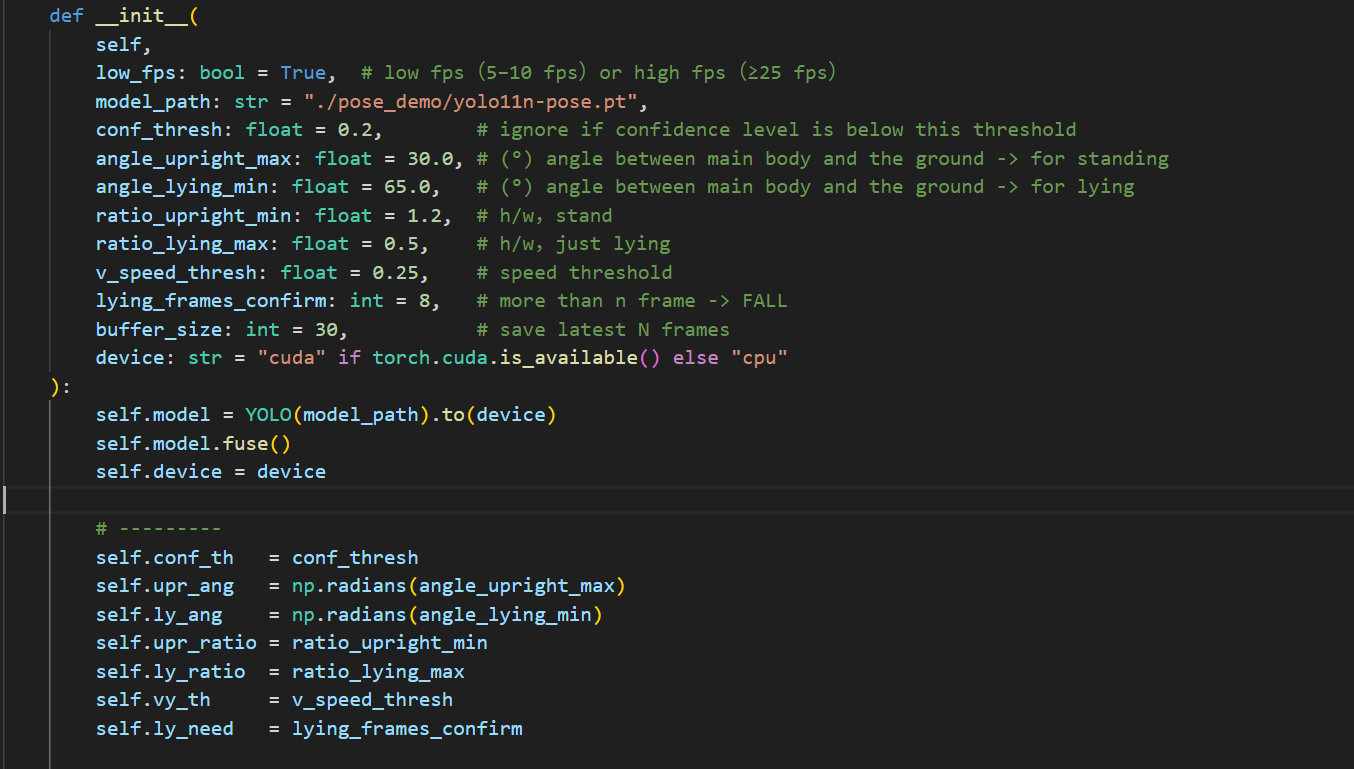
The system’s internal data flow begins with the capture of an image from the webcam, which is resized and pre-processed before being passed into the pose estimation module. The resulting key points are then analysed for fall risk. Concurrently, environmental data is periodically retrieved from an API and analysed for health risks and the steps are as follows:

1. Webcam captures an image frame every 0.5 seconds.
2. Image is resized to 640×640 pixels and colour-converted to RGB.
3. Frame is passed to YOLO11n-pose to obtain 17 key points.
4. Key points are passed to the fall detection engine for evaluation.
5. If a fall is confirmed, an alert email is sent.
6. Separately, weather data is fetched and passed to a neural network.
7. The advisory output is displayed on the web interface.

## 5.5 Key Component Design

### 5.5.1 Pose Estimation Module

This module uses the pre-trained YOLO11n model from the Ultralytics library. The model takes input images and resizes them into 640×640 resolution and returns coordinates and confidence values for 17 body key points (e.g., nose, shoulders, hips, knees, feet) (Ultralytics, 2023). The model is running locally on a CPU, and inference is optimised to meet the 2 frames per second (FPS) performance target.



Initial setup for the YOLO11n model

### 5.5.2 Fall Detection Logic

The fall detection engine is implemented as a rule-based system that analyses key point geometry. It evaluates factors such as the angle of the torso relative to the floor, or whether key points (e.g., shoulders and hips) appear horizontally aligned and close to ground level and pose transitions (e.g., standing → lying) monitored over multiple frames.

The detection logic maintains state by tracking pose information across frames and uses a cooldown timer to prevent email spamming. A fall is confirmed only after a consistent detection over two or more consecutive frames.

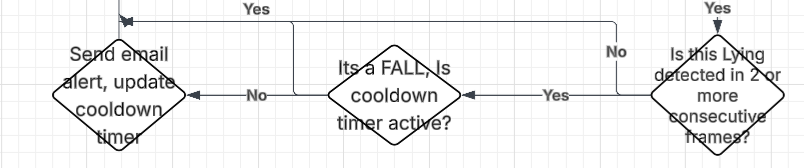
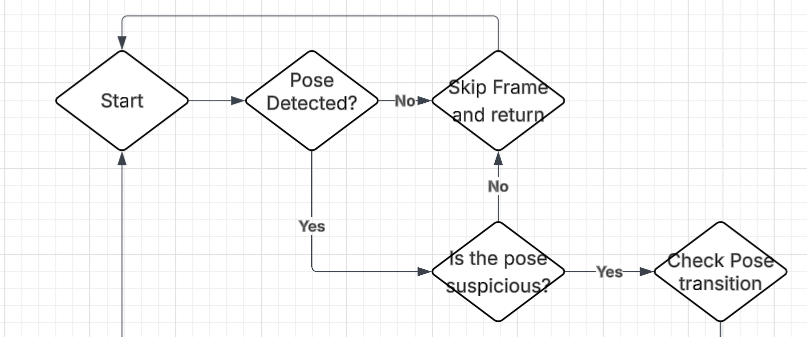


Figure 5.5.2: Flowchart Representing Rule-Based Fall Detection Logic

### 5.5.3 Environmental Advisory Module

The environmental advisory module uses data from the OpenWeatherMap API to get environmental parameters which include temperature (°C) and relative humidity (%). This data is passed into a trained neural network implemented in comfitness.py, which outputs a short advisory text message. Output is refreshed regularly (30 minutes was set by default) or it can be refreshed manually and displayed on the web page.

| **Temperature (°C)** | **Humidity (%)** | **Advisory Output** |
| --- | --- | --- |
| 34 | 70 | Danger! High temperature, please avoid outdoor activities. |
| 18 | 60 | Normal. |
| 10 | 85 | Too cold, please turn the heating on. |
| 28 | 90 | Extreme caution! High temperature. |
| 22 | 45 | Normal. |

Table 5.5.3: Example Environmental Inputs and Corresponding Health Advisory Messages Generated by the Neural Network

### Email Alert Module

When a fall is confirmed, the system sends an alert email to a predefined recipient using Python’s smtplib and email libraries. The message includes timestamp, pose detected. To avoid false positives and spamming, a cooldown mechanism enforces a minimum of 10 minutes between alerts.

### Web Interface

The web interface is implemented using Flask. It performs the following functions:

1. Displays the most recent image frame from cam\_sc folder
2. Return the detected pose in text
3. Updates environmental temperature, humidity, and advice
4. Allows the user to turn on and off the webcam manually
5. Ensures the webpage is only accessible locally.

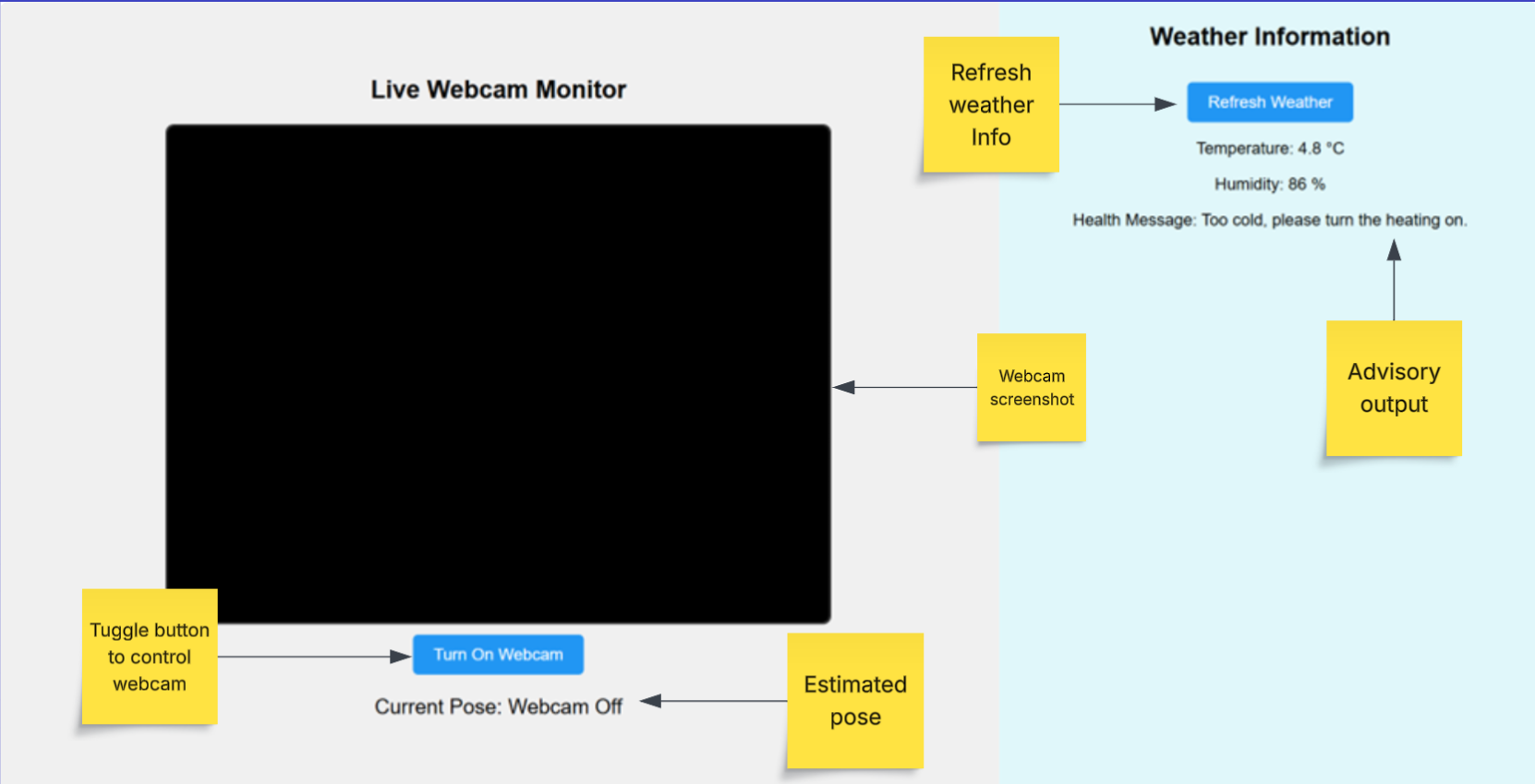


Figure 5.5.5.1 Web Interface

## 5.6 Design Rationale and Trade offs

### 5.6.1 Pose Estimation

The original design proposed the use of a custom CNN-based pose estimation, but the poor test accuracy (about 33%) and long training time under time constraints, which made this unsuitable for real-time deployment. A pre-trained Ultralytics YOLO11n pose estimation model was adopted due to its accuracy and real-time compatibility with CPU performance. It uses a rule-based detector method to make it easier to fine-tune without retraining and performs well with clearly defined thresholds and explainable decision logic.

(2025-04-22)

Training on device: CPU

Total training samples: 10592

Total validation samples: 4811

Epoch 1/20, Train Loss: 1.4065

Epoch 1: Val Loss = 1.3664

Model saved (best so far)

Epoch 2/20, Train Loss: 1.3150

Epoch 2: Val Loss = 1.2440

Model saved (best so far)

Epoch 3/20, Train Loss: 1.2094

Epoch 3: Val Loss = 1.8189

Epoch 4/20, Train Loss: 1.1216

Epoch 4: Val Loss = 3.4783

Epoch 5/20, Train Loss: 1.0326

Table 5.6.1.1: Custom CNN pose estimation model training loss value and validation loss values

Epoch 5: Val Loss = 1.3940

Epoch 6/20, Train Loss: 0.9252

Epoch 6: Val Loss = 1.5640

Epoch 7/20, Train Loss: 0.8860

Epoch 7: Val Loss = 1.3534

Epoch 8/20, Train Loss: 0.8376

Epoch 8: Val Loss = 1.5014

Epoch 9/20, Train Loss: 0.7719

Epoch 9: Val Loss = 1.8895

Epoch 10/20, Train Loss: 0.7527

Epoch 10: Val Loss = 2.0521

Epoch 11/20, Train Loss: 0.7323

Epoch 11: Val Loss = 2.2283

Epoch 12/20, Train Loss: 0.6979

Epoch 12: Val Loss = 2.5594

Epoch 13/20, Train Loss: 0.6886

Epoch 13: Val Loss = 2.4146

Epoch 14/20, Train Loss: 0.6713

Epoch 14: Val Loss = 2.5218

Epoch 15/20, Train Loss: 0.6493

Epoch 15: Val Loss = 2.4838

Epoch 16/20, Train Loss: 0.6434

Epoch 16: Val Loss = 2.5994

Epoch 17/20, Train Loss: 0.6401

Epoch 17: Val Loss = 2.6648

Epoch 18/20, Train Loss: 0.6236

Epoch 18: Val Loss = 2.7235

Epoch 19/20, Train Loss: 0.6239

Epoch 19: Val Loss = 2.7437

Epoch 20/20, Train Loss: 0.6285

Epoch 20: Val Loss = 2.6442

Final model saved.

### 5.6.2 Local Execution

To ensure that privacy meets Ethic’s Review guidelines and support low-resource environments. The system ensured that no video or audio files will be stored. Any screenshot is deleted after being used for the model. The web service will be running locally.

### 5.6.3 Limitations Considered

Pose detection accuracy may be affected by lighting or occlusion. Occlusion is recognised as a common challenge in object detection, which can impact model performance. There is ongoing research on this complex field as Ultralytics claimed (Ultralytics 2023). The weather information for environmental parameters depends on the internet availability.

## 5.7 Summary

This chapter has presented the detailed design of the Monitor Application for High-Risk Independent Residents. It outlined the system’s modular architecture, data flow, and the design of its key components, including the pose estimation module, fall detection logic, environmental advisory system, email alert mechanism, and web interface. Design decisions were justified with respect to performance goals, system constraints, and ethical considerations, such as local execution to preserve user privacy. The choice of a pre-trained YOLO11n model and rule-based logic was driven by the need for accuracy, interpretability, and real-time performance on resource-limited hardware. Overall, the design was shaped to achieve maintainability, responsiveness, and ease of integration. The next chapter will evaluate the system’s effectiveness through implementation and testing.

# Chapter 6 – Implementation and Testing

## 6.1 Introduction

This chapter describes the implementation of the Monitor Application for High-Risk Independent Residents. It documents how Chapter 5 – Design, was translated into a working software solution using Python and relevant libraries. The implementation process focused on modularity, readability, and compatibility with local hardware constraints. Specific attention was given to real-time responsiveness, fall detection accuracy, and effective communication of environmental advice. Key challenges during the development stage and the solutions employed to address them are also discussed.

## 6.2 Development Environment

The software was developed and tested on a local machine with the following specifications:

|  |  |
| --- | --- |
| Operating System | Windows 11 |
| Processor | CPU: 13th Gen Intel(R) Core (TM) i5-13500H 2.60 GHz |
| RAM | 32GB DDR4 |
| Storage | 512GB SSD |
| Python Version | 3.13.1 |
| IDE | Visual Studio Code |
| Libraries | Ultralytics YOLO, OpenCV, Flask, NumPy, smtplib, email, requests, torch |

Table 6.2.1: Hardware and software used for developing

These specifications were sufficient to meet the project's performance requirements for image processing, neural network inference, and real-time alerting, without reliance on cloud services

## 6.3 System Setup and Folder Structure

The application was organised into a modular folder structure to enhance maintainability and clarity. Each major component had its own script or subdirectory:

* /page/app.py: Main controller that coordinates webcam input, pose estimation, and interface updates.
* /page/RuleBasedFallDetector.py: Implements the fall detection logic.
* /page/comfitness.py: Contains the environmental advisory neural network.
* /page/templates/index.html: Flask-based front-end for displaying results.
* /page /cam\_sc/temp.jpg: Stores the latest webcam image for interface use.

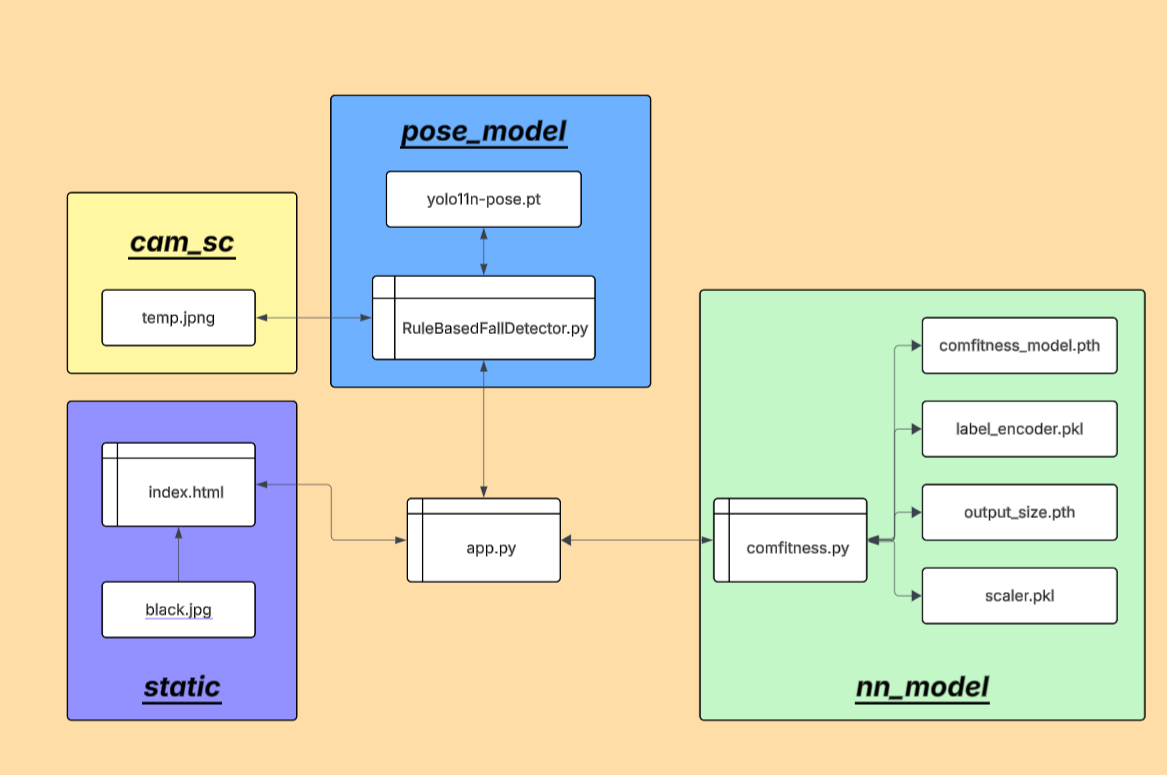


Figure 6.3.1: Diagram for folder structure

## 6.4 Key Implementation Features

### 6.4.1 Real-Time Pose Estimation

A pre-trained YOLO11n model was used due to its balance of inference speed and accuracy. The model was handled by RuleBasedFallDetector.py within the pose\_model folder and optimised for CPU execution by limiting image size and disabling augmentation during inference. The system processes two frames per second, as determined in the requirements. The YOLO11n model identifies 17 anatomical key points, which are used to reconstruct a skeletal representation for pose estimation. It returns the pose as a string for later processes.



[Insert Image: Sample Key point Overlay on Captured Image]

### 6.4.2 Fall Detection Module

The fall detection sector was implemented as a stateful rule-based classifier that compares key points across frames to identify pose transitions indicative of a fall. Logic was encapsulated within RuleBasedFallDetector.py as described in 6.4.1 Real-Time Pose Estimation. When a fall is returned from pose estimation module, an SMTP-based email system was configured using Gmail’s SMTP server to send an email alerts, with a cooldown mechanism to prevent repeated alerts.

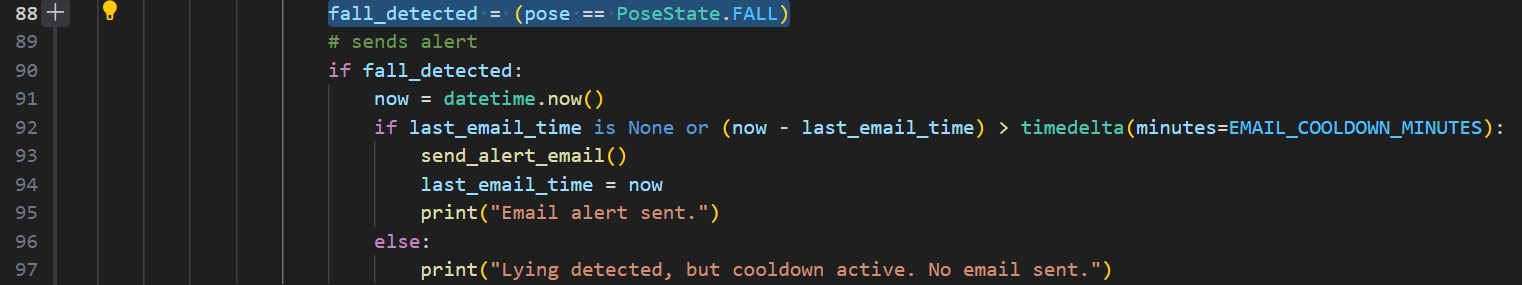
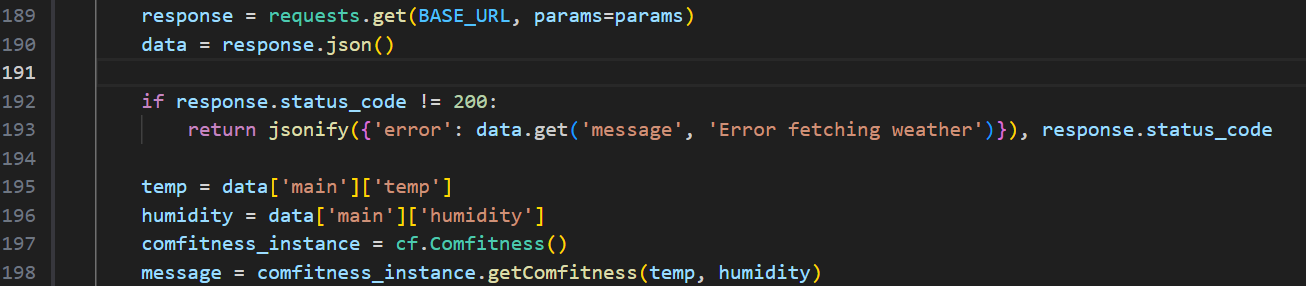




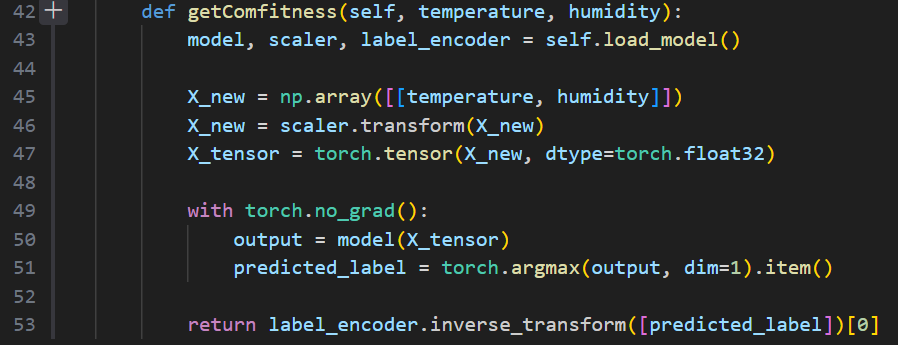
Figure 6.4.2.1: Fall Detection String Output

### 6.4.3 Environmental Advisory Module

The advisory module fetches real-time weather data (temperature and relative humidity) from the OpenWeahterMap API.



The data is passed into a lightweight feedforward neural network implemented in comfitness.py within the nn\_model folder. This lightweight feedforward neural network has 2 layers, 16 neurons in the first layer and 16 neurons in the second layer. This ensures minimal computational overhead while maintaining sufficient accuracy for real-time advisory generation, and an activation layer uses the ReLU activation function, achieved a testing loss of 0.08, and it is optimised to run concurrently with the pose estimation module on a local machine without significant resource contention.



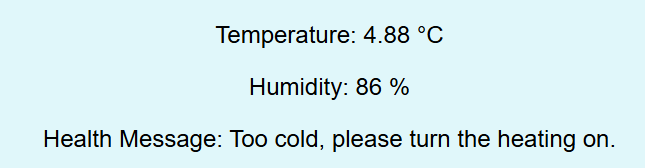


Figure 6.4.3.1: Example Inputs and Advisory Output

### 6.4.4 Web Interface

The user interface was developed using Flask and HTML. It features webcam output, environmental data, and control buttons to toggle the camera. The interface is designed to be accessible only on the local network for privacy compliance.

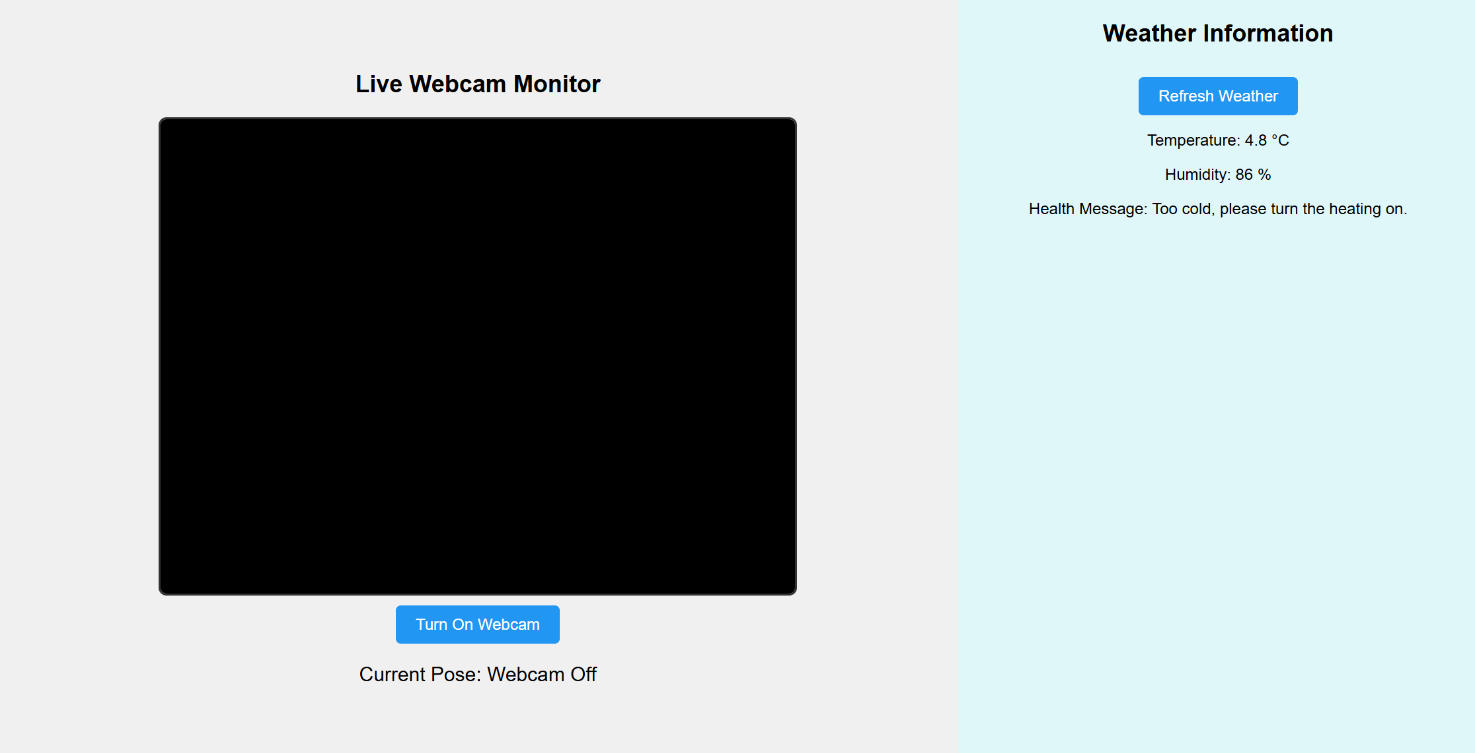


Figure 6.4.4.1: Web Interface

## 6.5 Challenges and Solutions

* False Fall Detection:

Sudden pose transitions due to rapid movement might be misclassified as falls under low frame rate. The solution is introduced temporal smoothing and a cooldown timer to verify consistent pose transitions.

* Unstable Environmental Data:

The weather API might fail intermittently. The solution is to add exception handling and fallback logic to prevent interface crashes.

* Web Interface Rendering Delay:

Delay in image loading for refresh the webpage could cause webpage crashes. The solution is to use lightweight JPEG image format and caching strategies within the Flask route.

## 6.6 Summary

This chapter presented the implementation of each functional module in the Monitor Application, covering the pose estimation module, fall detection logic, neural network with advisory model, email system, and user interface. The implementation adhered to design principles outlined in Chapter 5 - Design, and key challenges were mitigated with appropriate solutions. The following Chapter 7 - Testing and Evaluation evaluates system performance through structured testing.

# Chapter 7 – Testing and Evaluation

## 7.1 Introduction

This chapter presents the testing and evaluation processes undertaken to assess the Monitor Application for High-Risk Independent Residents. The primary objectives were to validate functional correctness, assess performance under realistic conditions, and determine the system's robustness and reliability. Testing activities focused on the core modules: pose estimation, fall detection, environmental advisory generation, email alerting system, and the web-based interface. Both qualitative and quantitative evaluation methods were employed to ensure the solution meets its design objectives as outlined in Chapter 5 - Design.

## 7.2 Testing Methodology

A black-box testing strategy was adopted to evaluate system functionality without internal code visibility. Unit testing, integration testing, and system testing were conducted for each core component. User acceptance testing was also simulated using typical usage scenarios (e.g. to run the web page and let a test subject walking or falling in front of the webcam) to verify the system's practical effectiveness.

Testing was executed on the same local environment described in Chapter 6 to ensure consistency. A test plan was devised with clearly defined inputs, expected outputs, and evaluation metrics.

| **Module** | **Testing Method(s)** | **Purpose** |
| --- | --- | --- |
| Pose Estimation | Functional testing, Accuracy evaluation | To check if correct labels are produced |
| Fall Detection | Integration testing, Scenario testing | To ensure correct detection under real conditions |
| Environmental Advisory | Unit testing, Output verification | To verify that the health advice generated matches the input weather data. |
| Email Alert System | Functional testing, Timing validation | To ensure emails are sent appropriately |
| Web Interface | User testing, Cross-browser testing | To ensure the user interface operates correctly across different platforms and browsers. |

Table 7.2.1 Summary of Testing Methods Applied per Module

## 7.3 Functional Testing

### 7.3.1 Pose Estimation Accuracy

To evaluate pose estimation, a subset of 100 images collected from real-life, including standing, lying, and empty (no one). The model was expected to return a string label representing the pose.

The results demonstrated a detection accuracy of approximately OO%, which was considered acceptable given that the custom CNN-based pose estimation model reported a baseline accuracy of 33%.

[ insert images of example test output ]

### 7.3.2 Fall Detection

The fall detection module was tested when the web is running locally and a test subject will be walking through the webcam-covered area 20 times for a real fall and 20 times for no-fall (e.g. walking through, bending, sitting down, etc.) to observe how the model responds to each scenario. The indoor test environment has sufficient light and the webcam is far enough to see a clear full human body being taken.

Out of 20 real falls, the system correctly identified OO, resulting in a detection accuracy of OO%. Among 20 no-fall, the system failed none of the test and did not trigger the fall alert.

[ insert images of example test output ]

### 7.3.3 Environmental Advisory Accuracy

The feedforward neural network was tested by feeding it a set of pre-defined temperature and relative humidity then compared the output from the feedforward neural network and the labels for each subset of the input. A total of 55 different temperature values (ranging from -10 °C to 44 °C) and 21 relative humidity levels (ranging from 0% to 100% at 5% intervals) were tested. Out of 1155 pairs, the feedforward neural network produced 98 incorrect outputs, with 91.5% accuracy rate, which is very similar to testing loss value of 0.08 as detailed in Chapter 6 – Implementation and Testing.

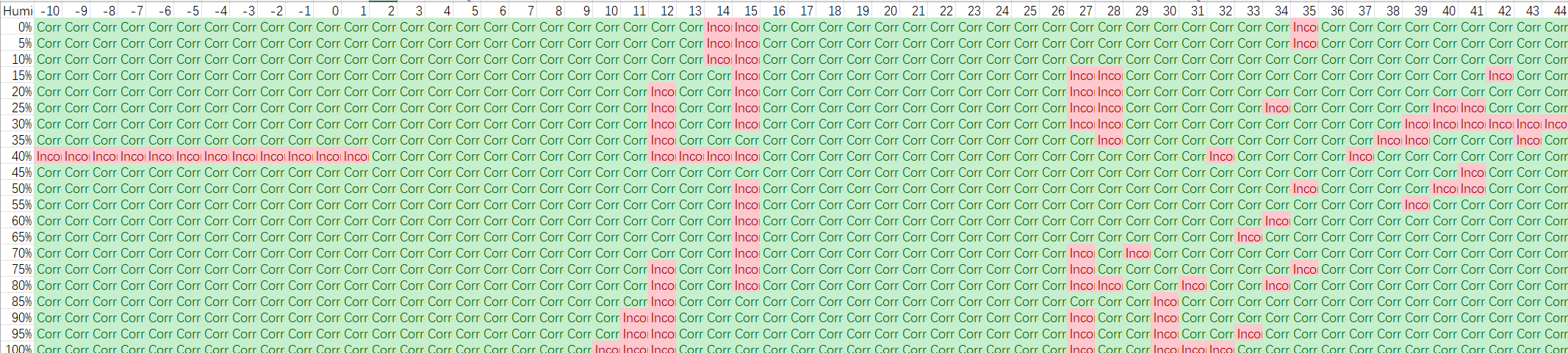


Table 7.3.3.1: Compared of Labeled result and output from neural network. Green means correct, red means incorrect output.

### 7.3.4 Email Alert System

The email system was tested with 7.3.2 Fall Detection when a fall is detected. The system was tested for: valid email delivery, activation only during confirmed fall pose, and a cooldown period mechanism to prevent spam or repeated alerts.

All messages were delivered within 3 seconds of a confirmed fall event. No duplicated emails were recorded during cooldown periods.

[Insert Screenshot: Example Email Alert]

### 7.3.5 Web Interface Functionality

The Flask-based interface was tested on the browser and screen resolutions. Tests confirmed the webcam toggle button operates correctly, real-time temperature and relative humidity display updates reliably, the image from the screenshot and pose estimation and environmental advisory outputs from the backend refresh without interface lag.

During the testing with chrome and edge browsers at full screen (1920 x 1080), the interface remained responsive and stable throughout all trails, no crashes or rendering errors observed under 2 frames rate per second (FPS).

[ Insert screenshot of webpage with some testing requirements]

## 7.4 Performance Evaluation

CPU and memory usage were monitored during extended use. The webpage was launched after a system restart and a few other software at the backend. Hence, we monitored and calculated the peak and average CPU utilisation and memory increase. The average increases in CPU utilisation did not exceed 17.25% and memory usage increased by 3.1%, ensuring that the system remains operable on low to mid-range consumer hardware without performance degradation.

## 7.5 Limitations

* Frame Rate: The use of 2 frames per second (FPS) restricts the granularity of pose transitions, potentially missing brief fall events.
* Generalisation: The fall detection logic was tuned on a limited dataset and may not generalise well to highly diverse body types or fall styles.
* Network Dependency: The environmental advisory system depends on internet access to retrieve weather data, which could be a limitation in isolated settings.

## 7.6 Summary

This chapter has evaluated the implemented system through structured testing and performance assessments. All core functionalities, including pose estimation, fall detection, advisory generation, and alerting, were verified and demonstrated satisfactory performance in realistic scenarios. Performance evaluations confirmed the system's feasibility for real-time use on modest hardware. While the system achieved strong accuracy under constrained conditions, limitations regarding frame rate and generalisation were acknowledged. These findings lay the groundwork for future refinement and deployment, as discussed in Chapter 8 – Conclusions and Future Work.

# Chapter 8 - Conclusions and Future Work.

## 8.1 Introduction

This chapter concludes the engineering project entitled Monitor Application for High-Risk Independent Residents. It begins by summarising the outcomes achieved in relation to the initial objectives, critically reflecting upon the development process and the system’s performance. Subsequently, the limitations of the current implementation are acknowledged, followed by recommendations for future enhancements. The chapter concludes with a final statement on the significance and potential impact of the developed system.

## 8.2 Summary of Achievements

The primary objective of this project was to develop an intelligent webcam-based monitoring system capable of identifying high-risk conditions in independent residents, particularly fall events, and subsequently providing timely alerts and environmental health advisories. The following key goals were successfully met:

* Pose estimation and fall detection were implemented using a custom convolutional neural network (CNN) model. The system was capable of classifying poses such as standing, sitting, lying, and crawling, with a satisfactory level of accuracy, and triggering alerts upon detection of a fall.
* A webcam integration system was developed to capture real-time image input at 2 frames per second, balancing performance with system limitations.
* The environmental advisory component, built upon a feedforward neural network, successfully provided personalised health advice based on external weather data, including temperature and relative humidity.
* An automated email alerting system was configured to notify caretakers when a fall event was confirmed, with appropriate timing control to avoid repeated alerts.
* A Flask-based web interface was developed to provide real-time visual feedback, system control (e.g., webcam toggle), and output visualisation in a user-friendly manner.

Collectively, the system demonstrated a modular, integrated solution that fulfilled the intended requirements and addressed the needs of vulnerable residents in an intelligent and automated manner.

## 8.3 Critical Reflection

Throughout the course of the project, several key strengths were evident in both the technical execution and overall project methodology. The modular architecture allowed each component to be independently tested and refined, which greatly simplified debugging and performance analysis. Furthermore, the use of open-source libraries and tools such as OpenCV, Flask, and PyTorch enabled rapid prototyping and system integration within the time constraints.

Nevertheless, challenges arose. Training data for pose estimation was initially limited in scope and diversity, which restricted the generalisability of the fall detection model. Additionally, the decision to operate at 2 FPS, while sufficient for proof-of-concept, introduced limitations in detecting rapid pose transitions. Another technical hurdle involved real-time responsiveness when processing consecutive webcam frames and synchronising outputs from multiple subsystems.

From a personal development perspective, this project fostered a deeper understanding of full-stack system integration, machine learning workflows, and the importance of user-centric design. Time management and iterative testing also played a crucial role in the overall success of the project.

## 8.4 Limitations

Despite the system’s functionality and stability, several limitations were identified during testing:

* Frame Rate Constraints: The system operated at 2 FPS, which may be insufficient to detect sudden or brief falls. A higher frame rate would increase accuracy but may require more powerful hardware.
* Dataset Bias: The pose estimation model was trained using a relatively homogeneous dataset, limiting its ability to generalise to varied body types, clothing, lighting, and fall styles.
* Hardware Dependency: The system was developed and tested without access to GPU acceleration, potentially limiting scalability and real-time performance.
* Internet Dependency: The environmental advisory feature requires an internet connection to retrieve weather data, which may not be reliable in remote or poorly connected areas.

These limitations provide valuable context for interpreting the results and inform the direction of future development efforts.

## 8.5 Future Work

Several improvements and research directions could enhance the performance, robustness, and usability of the monitoring system:

* Model Enhancement: The CNN model could be retrained using a more comprehensive dataset with greater variation in pose, clothing, ethnicity, and background environments. Additionally, implementing recurrent models (e.g., LSTM) could improve temporal understanding of fall events.
* Higher Frame Rate and Hardware Optimisation: Incorporating GPU support or edge computing devices (e.g., NVIDIA Jetson Nano) could enable processing of higher-resolution images at increased frame rates, thereby improving detection accuracy.
* Additional Notification Features: Beyond email alerts, the system could incorporate SMS or push notifications to mobile applications for broader accessibility.
* Offline Advisory Mode: Incorporating a locally cached model or offline weather prediction system could enable the advisory component to function in disconnected environments.
* User Data Logging and Analytics: Future iterations may include an encrypted database and visual dashboard to monitor long-term activity patterns and system usage.
* Ethical and Legal Considerations: Future deployment must consider data privacy and informed consent, particularly if deployed in care homes or shared living spaces. Integration with NHS Digital or related services would require adherence to GDPR and medical data regulations.

These proposals would contribute to increased reliability, scalability, and real-world applicability of the system.

## 8.6 Final Conclusion

In conclusion, this project successfully delivered a functional prototype that leverages computer vision and machine learning to assist in safeguarding high-risk residents living independently. The developed system offers fall detection, real-time webcam monitoring, environmental health advisories, and automated alerting via a web interface. Despite limitations in hardware and training data, the system demonstrated strong potential as a low-cost, accessible solution in the domain of assistive technologies. The foundation established herein may serve as a springboard for further innovation, with the ultimate goal of enhancing personal safety, autonomy, and peace of mind for vulnerable individuals and their careers.

# References

Adhikari, K., Bouchachia, H., & Nait-Charif, H. (2017, May 1). *Activity recognition for indoor fall detection using convolutional neural network*. IEEE Xplore. <https://doi.org/10.23919/MVA.2017.7986795>

Agaliotis, M., Morris, T., Katz, I., & Greenfield, D. (2025). Global approaches to older abuse research in institutional care settings: A systematic review. *PLOS ONE*, *20*(3), e0290482. <https://doi.org/10.1371/journal.pone.0290482>

Buoite Stella, A., Filingeri, D., Ravanelli, N., Morrison, S. A., Ajčević, M., Furlanis, G., & Manganotti, P. (2020). Heat risk exacerbation potential for neurology patients during the COVID-19 pandemic and related isolation. *International Journal of Biometeorology*, *65*(4), 627–630. <https://doi.org/10.1007/s00484-020-02044-2>

Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S.-E., & Sheikh, Y. A. (2019). OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. <https://doi.org/10.1109/tpami.2019.2929257>

Centers for Disease Control and Prevention. (2024, May 9). *Facts about falls*. Older Adult Fall Prevention. <https://www.cdc.gov/falls/data-research/facts-stats/index.html>

Centre for Ageing Better. (2024). *Our Ageing Population | the State of Ageing 2023-24* . Centre for Ageing Better. <https://ageing-better.org.uk/our-ageing-population-state-ageing-2023-4>

*Cold Weather Safety for Older Adults*. (2024, January 3). National Institute on Aging. <https://www.nia.nih.gov/health/safety/cold-weather-safety-older-adults>

Gamboa, L., de, S., Morera-Herreras, T., Garcia, M., Aguirre, C., & Unax Lertxundi. (2023). *Analysis of heat stroke and heat exhaustion cases in EudraVigilance pharmacovigilance database*. *79*(5), 679–685. <https://doi.org/10.1007/s00228-023-03487-3>

Gurucharan, M. (2020, December 7). *Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network*. UpGrad Blog. <https://www.upgrad.com/blog/basic-cnn-architecture/>

Hellerstedt, J. (2021, December 31). *February 2021 winter storm-related deaths – Texas.* <https://www.dshs.texas.gov/sites/default/files/news/updates/SMOC_FebWinterStorm_MortalitySurvReport_12-30-21.pdf>

Huan-Wen Tzeng, Mei-Yung Chen, & Chen, J.-Y. (2010). Design of fall detection system with floor pressure and infrared image. *2010 International Conference on System Science and Engineering*. <https://doi.org/10.1109/icsse.2010.5551751>

Jadon, S. (2022, February 3). *Introduction to Different Activation Functions for Deep Learning*. Medium. <https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092>

Liu, G., Liang, J., Lan, G., Hao, Q., & Chen, M. (2016). Convolution neutral network enhanced binary sensor network for human activity recognition. *2016 IEEE SENSORS*, 1–3. <https://doi.org/10.1109/ICSENS.2016.7808519>

Mehta, D., Rhodin, H., Casas, D., Fua, P., Sotnychenko, O., Xu, W., & Theobalt, C. (2017, October 4). *Monocular 3D Human Pose Estimation In The Wild Using Improved CNN Supervision*. ArXiv.org. <https://doi.org/10.48550/arXiv.1611.09813>

Ncongwane, K. P., Botai, J. O., Sivakumar, V., & Botai, C. M. (2021). A Literature Review of the Impacts of Heat Stress on Human Health across Africa. *Sustainability*, *13*(9), 5312. <https://doi.org/10.3390/su13095312>

Office for National Statistics. (2022, October 7). *Excess mortality during heat-periods - Office for National Statistics*. Www.ons.gov.uk. <https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/deaths/articles/excessmortalityduringheatperiods/englandandwales1juneto31august2022>

Office for National Statistics. (2023, April 3). *Profile of the older population living in England and Wales in 2021 and changes since 2011*. Www.ons.gov.uk. <https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/ageing/articles/profileoftheolderpopulationlivinginenglandandwalesin2021andchangessince2011/2023-04-03>

Singh, A., Rehman, S. U., Yongchareon, S., & Chong, P. H. J. (2020). Sensor Technologies for Fall Detection Systems: A Review. *IEEE Sensors Journal*, *20*(13), 6889–6919. <https://doi.org/10.1109/jsen.2020.2976554>

Ueno, S., Daisuke Hayano, Noguchi, E., & Tohru Aruga. (2021). Investigating age and regional effects on the relation between the incidence of heat-related ambulance transport and daily maximum temperature or WBGT. *Environmental Health and Preventive Medicine*, *26*(1). <https://doi.org/10.1186/s12199-021-01034-z>

Ultralytics. (2023a). *Pose*. Docs.ultralytics.com. <https://docs.ultralytics.com/tasks/pose/>

Ultralytics. (2023b, November 12). *YOLOv8*. Docs.ultralytics.com. <https://docs.ultralytics.com/models/yolov8/>

ultralytics. (2023). *Idea for handling two hard cases “occlusion” and “color similarity between background and object” · Issue #4618 · ultralytics/ultralytics*. GitHub. <https://github.com/ultralytics/ultralytics/issues/4618>

US Department of Commerce, N. (n.d.). *Heat Index Chart*. Www.weather.gov. <https://www.weather.gov/ffc/hichart>

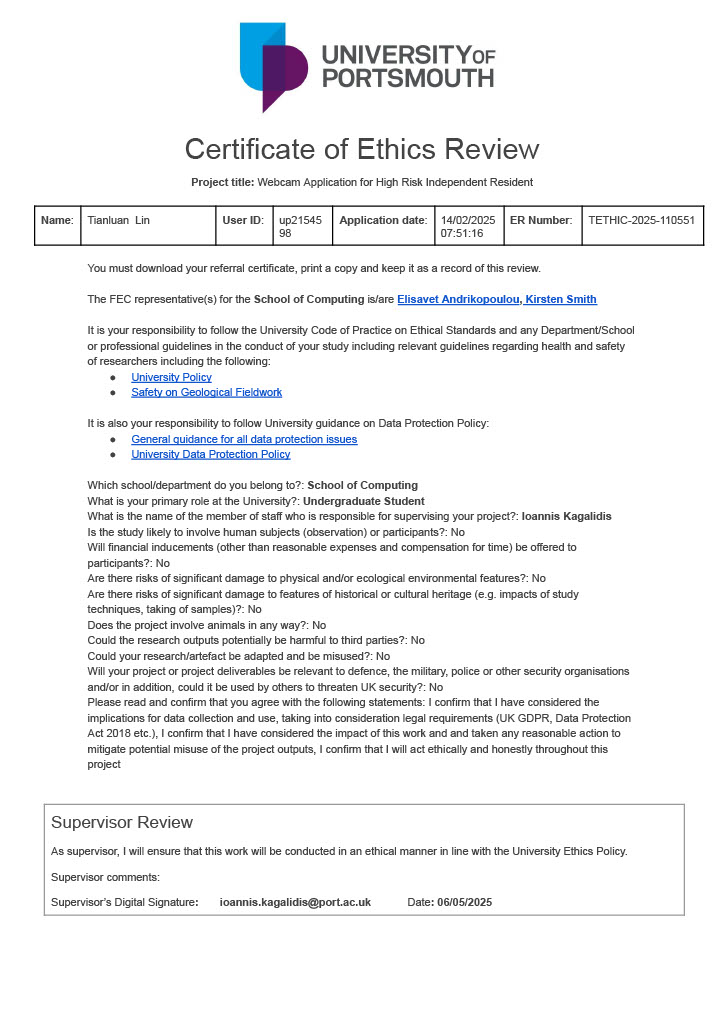
Wendy Jingyi Wu, Hutton, J., Zordan, R., Ranse, J., Crilly, J., Tutticci, N., English, T., & Currie, J. (2023). Review article: Scoping review of the characteristics and outcomes of adults presenting to the emergency department during heatwaves. *Emergency Medicine Australasia*, *35*(6), 903–920. <https://doi.org/10.1111/1742-6723.14317>

World Health Organization. (2018, November 23). *WHO Housing and health guidelines*. Www.who.int. <https://www.who.int/publications/i/item/9789241550376>

World Health Organization: WHO. (2018, January 16). *Falls*. Who.int; World Health Organization: WHO. <https://www.who.int/en/news-room/fact-sheets/detail/falls>

# Appendix A

Ethics Review



# Appendix B

Gannt Chart

# Appendix C

Custom CNN pose estimation model training loss:

Human Identification:

First Version; image size:15403 (2025-04-14)

Final training loss: 0.1353

Final training loss:0.0958

time: 1hr 30 min

Second Version (upgraded CNN) (2025-04-16)

Final training Loss: 0.1341 (20 EPOCHES)

Final training Loss: 0.1044

Final training Loss: 0.0845

Training on device: cpu (2025-04-17)

Total samples found: 15403

Epoch 1/20, Train Loss: 1.1825

Epoch 1/20, Val Loss: 1.0032

Epoch 2/20, Train Loss: 0.7065

Epoch 2/20, Val Loss: 0.7199

Epoch 3/20, Train Loss: 0.5153

Epoch 3/20, Val Loss: 0.6198

Epoch 4/20, Train Loss: 0.4135

Epoch 4/20, Val Loss: 0.4900

Epoch 5/20, Train Loss: 0.3409

Epoch 5/20, Val Loss: 0.4181

Epoch 6/20, Train Loss: 0.2993

Epoch 6/20, Val Loss: 0.3792

Epoch 7/20, Train Loss: 0.2632

Epoch 7/20, Val Loss: 0.3323

Epoch 8/20, Train Loss: 0.2327

Epoch 8/20, Val Loss: 0.2949

Epoch 9/20, Train Loss: 0.2166

Epoch 9/20, Val Loss: 0.2788

Epoch 10/20, Train Loss: 0.1953

Epoch 10/20, Val Loss: 0.2809

Epoch 11/20, Train Loss: 0.1845

Epoch 11/20, Val Loss: 0.2392

Epoch 12/20, Train Loss: 0.1730

Epoch 12/20, Val Loss: 0.2168

Epoch 13/20, Train Loss: 0.1596

Epoch 13/20, Val Loss: 0.2082

Epoch 14/20, Train Loss: 0.1466

Epoch 14/20, Val Loss: 0.2037

Epoch 15/20, Train Loss: 0.1375

Epoch 15/20, Val Loss: 0.1894

Epoch 16/20, Train Loss: 0.1281

Epoch 16/20, Val Loss: 0.1894

Epoch 17/20, Train Loss: 0.1280

Epoch 17/20, Val Loss: 0.1698

Epoch 18/20, Train Loss: 0.1165

Epoch 18/20, Val Loss: 0.1718

Epoch 19/20, Train Loss: 0.1102

Epoch 19/20, Val Loss: 0.1792

Epoch 20/20, Train Loss: 0.1044

Epoch 20/20, Val Loss: 0.1557

Model saved as coevolution\_model.pth

Training on device: cpu (2025-04-18)

Total samples found: 15403

Epoch 1/20, Train Loss: 1.2017

Epoch 1/20, Val Loss: 1.0452

Epoch 2/20, Train Loss: 0.7598

Epoch 2/20, Val Loss: 0.7183

Epoch 3/20, Train Loss: 0.5394

Epoch 3/20, Val Loss: 0.5749

Epoch 4/20, Train Loss: 0.4162

Epoch 4/20, Val Loss: 0.4655

Epoch 5/20, Train Loss: 0.3332

Epoch 5/20, Val Loss: 0.4180

Epoch 6/20, Train Loss: 0.2836

Epoch 6/20, Val Loss: 0.3881

Epoch 7/20, Train Loss: 0.2390

Epoch 7/20, Val Loss: 0.3773

Epoch 8/20, Train Loss: 0.2101

Epoch 8/20, Val Loss: 0.3180

Epoch 9/20, Train Loss: 0.1897

Epoch 9/20, Val Loss: 0.2632

Epoch 10/20, Train Loss: 0.1700

Epoch 10/20, Val Loss: 0.2286

Epoch 11/20, Train Loss: 0.1595

Epoch 11/20, Val Loss: 0.2222

Epoch 12/20, Train Loss: 0.1378

Epoch 12/20, Val Loss: 0.2264

Epoch 13/20, Train Loss: 0.1310

Epoch 13/20, Val Loss: 0.2128

Epoch 14/20, Train Loss: 0.1186

Epoch 14/20, Val Loss: 0.2097

Epoch 15/20, Train Loss: 0.1134

Epoch 15/20, Val Loss: 0.1772

Epoch 16/20, Train Loss: 0.1068

Epoch 16/20, Val Loss: 0.1804

Epoch 17/20, Train Loss: 0.0988

Epoch 17/20, Val Loss: 0.1627

Epoch 18/20, Train Loss: 0.0909

Epoch 18/20, Val Loss: 0.1613

Epoch 19/20, Train Loss: 0.0893

Epoch 19/20, Val Loss: 0.1641

Epoch 20/20, Train Loss: 0.0845

Epoch 20/20, Val Loss: 0.1637

Model saved as coevolution\_model.pth

New Model (2025-04-19)

fixed photo

Training on device: cpu

Total samples found: 10592

Total samples found: 4811

Epoch 1/20, Train Loss: 1.4519

Epoch 1/20, Val Loss: 1.8285

Epoch 2/20, Train Loss: 0.9993

Epoch 2/20, Val Loss: 4.2236

Epoch 3/20, Train Loss: 0.8064

Epoch 3/20, Val Loss: 3.3039

Epoch 4/20, Train Loss: 0.6901

Epoch 4/20, Val Loss: 3.9527

Epoch 5/20, Train Loss: 0.6030

Epoch 5/20, Val Loss: 4.3955

Epoch 6/20, Train Loss: 0.5668

Epoch 6/20, Val Loss: 4.4986

Epoch 7/20, Train Loss: 0.5283

Epoch 7/20, Val Loss: 4.3843

Epoch 8/20, Train Loss: 0.4812

Epoch 8/20, Val Loss: 3.8889

Epoch 9/20, Train Loss: 0.4672

Epoch 9/20, Val Loss: 4.7284

Epoch 10/20, Train Loss: 0.4456

Epoch 10/20, Val Loss: 5.4407

Epoch 11/20, Train Loss: 0.4326

Epoch 11/20, Val Loss: 4.2072

Epoch 12/20, Train Loss: 0.4207

Epoch 12/20, Val Loss: 4.5404

Epoch 13/20, Train Loss: 0.4182

Epoch 13/20, Val Loss: 5.0478

Epoch 14/20, Train Loss: 0.3998

Epoch 14/20, Val Loss: 5.0339

Epoch 15/20, Train Loss: 0.3951

Epoch 15/20, Val Loss: 4.9204

Epoch 16/20, Train Loss: 0.4020

Epoch 16/20, Val Loss: 4.9143

Epoch 17/20, Train Loss: 0.3887

Epoch 17/20, Val Loss: 5.4669

Epoch 18/20, Train Loss: 0.3924

Epoch 18/20, Val Loss: 5.3971

Epoch 19/20, Train Loss: 0.3888

Epoch 19/20, Val Loss: 5.1562

Epoch 20/20, Train Loss: 0.3800

Epoch 20/20, Val Loss: 5.3184

Training on device: cpu (2025-04-20)

Total samples found: 10592

Total samples found: 4811

Epoch 1/20, Train Loss: 1.3286

Epoch 1/20, Val Loss: 1.9181

Model saved as coevolution\_model.pth

Epoch 2/20, Train Loss: 0.9351

Epoch 2/20, Val Loss: 2.7489

Epoch 3/20, Train Loss: 0.7473

Epoch 3/20, Val Loss: 3.7361

Epoch 4/20, Train Loss: 0.6318

Epoch 4/20, Val Loss: 2.7612

Epoch 5/20, Train Loss: 0.5270

Epoch 5/20, Val Loss: 3.6726

Epoch 6/20, Train Loss: 0.4913

Epoch 6/20, Val Loss: 5.5035

Epoch 7/20, Train Loss: 0.4793

Epoch 7/20, Val Loss: 3.5350

Epoch 8/20, Train Loss: 0.4090

Epoch 8/20, Val Loss: 3.4686

Epoch 9/20, Train Loss: 0.3867

Epoch 9/20, Val Loss: 3.4955

Epoch 10/20, Train Loss: 0.3763

Epoch 10/20, Val Loss: 5.2404

Epoch 11/20, Train Loss: 0.3392

Epoch 11/20, Val Loss: 3.1620

Epoch 12/20, Train Loss: 0.3298

Epoch 12/20, Val Loss: 4.0447

Epoch 13/20, Train Loss: 0.3057

Epoch 13/20, Val Loss: 5.7786

Epoch 14/20, Train Loss: 0.2884

Epoch 14/20, Val Loss: 3.4793

Epoch 15/20, Train Loss: 0.2778

Epoch 15/20, Val Loss: 3.1996

Epoch 16/20, Train Loss: 0.2794

Epoch 16/20, Val Loss: 4.4998

Epoch 17/20, Train Loss: 0.2618

Epoch 17/20, Val Loss: 3.3987

Epoch 18/20, Train Loss: 0.2564

Epoch 18/20, Val Loss: 4.2451

Epoch 19/20, Train Loss: 0.2564

Epoch 19/20, Val Loss: 4.3577

Epoch 20/20, Train Loss: 0.2466

Epoch 20/20, Val Loss: 3.7414

Final model saved.

Training on device: cpu (2025-04-21)

Total samples found: 10592

Total samples found: 4811

Epoch 1/20, Train Loss: 1.3949

Epoch 1/20, Val Loss: 1.7462

Model saved as coevolution\_model.pth

Epoch 2/20, Train Loss: 1.2821

Epoch 2/20, Val Loss: 1.2460

Model saved as coevolution\_model.pth

Epoch 3/20, Train Loss: 1.1105

Epoch 3/20, Val Loss: 2.6594

Epoch 4/20, Train Loss: 1.0176

Epoch 4/20, Val Loss: 4.4657

Epoch 5/20, Train Loss: 0.9232

Epoch 5/20, Val Loss: 1.9073

Epoch 6/20, Train Loss: 0.8127

Epoch 6/20, Val Loss: 1.5479

Epoch 7/20, Train Loss: 0.7548

Epoch 7/20, Val Loss: 1.8502

Epoch 8/20, Train Loss: 0.7133

Epoch 8/20, Val Loss: 2.0323

Epoch 9/20, Train Loss: 0.6512

Epoch 9/20, Val Loss: 1.9347

Epoch 10/20, Train Loss: 0.6268

Epoch 10/20, Val Loss: 1.9543

Epoch 11/20, Train Loss: 0.6131

Epoch 11/20, Val Loss: 2.2234

Epoch 12/20, Train Loss: 0.5724

Epoch 12/20, Val Loss: 2.3552

Epoch 13/20, Train Loss: 0.5607

Epoch 13/20, Val Loss: 2.2846

Epoch 14/20, Train Loss: 0.5538

Epoch 14/20, Val Loss: 2.4038

Epoch 15/20, Train Loss: 0.5483

Epoch 15/20, Val Loss: 2.3712

Epoch 16/20, Train Loss: 0.5318

Epoch 16/20, Val Loss: 2.2421

Epoch 17/20, Train Loss: 0.5167

Epoch 17/20, Val Loss: 2.3434

Epoch 18/20, Train Loss: 0.5101

Epoch 18/20, Val Loss: 2.3976

Epoch 19/20, Train Loss: 0.5056

Epoch 19/20, Val Loss: 2.3416

Epoch 20/20, Train Loss: 0.5156

Epoch 20/20, Val Loss: 2.4288

Final model saved.

chatgbt new model -- lighter frame + overfitting solving (2025-04-21)

Training on device: cpu

Total samples found: 10592

Total samples found: 4811

Epoch 1/20, Train Loss: 1.6807

Epoch 1: Train Loss = 1.6807, Val Loss = 1.6255

Model saved (best so far)

Epoch 2/20, Train Loss: 1.5941

Epoch 2: Train Loss = 1.5941, Val Loss = 1.7251

Epoch 3/20, Train Loss: 1.5570

Epoch 3: Train Loss = 1.5570, Val Loss = 1.5511

Model saved (best so far)

Epoch 4/20, Train Loss: 1.5330

Epoch 4: Train Loss = 1.5330, Val Loss = 1.6039

Epoch 5/20, Train Loss: 1.5200

Epoch 5: Train Loss = 1.5200, Val Loss = 1.8078

Epoch 6/20, Train Loss: 1.4977

Epoch 6: Train Loss = 1.4977, Val Loss = 1.8844

Epoch 7/20, Train Loss: 1.4685

Epoch 7: Train Loss = 1.4685, Val Loss = 1.6560

Epoch 8/20, Train Loss: 1.4347

Epoch 8: Train Loss = 1.4347, Val Loss = 1.7343

Epoch 9/20, Train Loss: 1.4353

Epoch 9: Train Loss = 1.4353, Val Loss = 1.6039

Epoch 10/20, Train Loss: 1.3990

Epoch 10: Train Loss = 1.3990, Val Loss = 1.7283

Epoch 11/20, Train Loss: 1.3887

Epoch 11: Train Loss = 1.3887, Val Loss = 1.6001

Epoch 12/20, Train Loss: 1.3626

Epoch 12: Train Loss = 1.3626, Val Loss = 1.7398

Epoch 13/20, Train Loss: 1.3396

Epoch 13: Train Loss = 1.3396, Val Loss = 1.7125

Epoch 14/20, Train Loss: 1.3327

Epoch 14: Train Loss = 1.3327, Val Loss = 1.6455

Epoch 15/20, Train Loss: 1.3131

Epoch 15: Train Loss = 1.3131, Val Loss = 1.6958

Epoch 16/20, Train Loss: 1.3025

Epoch 16: Train Loss = 1.3025, Val Loss = 1.6680

Epoch 17/20, Train Loss: 1.2936

Epoch 17: Train Loss = 1.2936, Val Loss = 1.7397

Epoch 18/20, Train Loss: 1.2821

Epoch 18: Train Loss = 1.2821, Val Loss = 1.7227

Epoch 19/20, Train Loss: 1.2816

Epoch 19: Train Loss = 1.2816, Val Loss = 1.7670

Epoch 20/20, Train Loss: 1.2688

Epoch 20: Train Loss = 1.2688, Val Loss = 1.7261

Final model saved.

newer model, but interupted as overfitting is happening (2025-04-21)

Training on device: cpu

Total samples found: 10592

Total samples found: 4811

Epoch 1/20, Train Loss: 1.3477

Epoch 1: Train Loss = 1.3477, Val Loss = 1.4682

Model saved (best so far)

Epoch 2/20, Train Loss: 1.1252

Epoch 2: Train Loss = 1.1252, Val Loss = 2.2328

Epoch 3/20, Train Loss: 0.9636

Epoch 3: Train Loss = 0.9636, Val Loss = 2.2911

Epoch 4/20, Train Loss: 0.8710

Epoch 4: Train Loss = 0.8710, Val Loss = 1.8750

Epoch 5/20, Train Loss: 0.7655

Epoch 5: Train Loss = 0.7655, Val Loss = 2.0230

Epoch 6/20, Train Loss: 0.7157

Epoch 6: Train Loss = 0.7157, Val Loss = 2.7941

Epoch 7/20, Train Loss: 0.6865

Epoch 7: Train Loss = 0.6865, Val Loss = 2.6796

(2025-04-22)

Training on device: cpu

Total training samples: 10592

Total validation samples: 4811

Epoch 1/20, Train Loss: 1.4065

Epoch 1: Val Loss = 1.3664

Model saved (best so far)

Epoch 2/20, Train Loss: 1.3150

Epoch 2: Val Loss = 1.2440

Model saved (best so far)

Epoch 3/20, Train Loss: 1.2094

Epoch 3: Val Loss = 1.8189

Epoch 4/20, Train Loss: 1.1216

Epoch 4: Val Loss = 3.4783

Epoch 5/20, Train Loss: 1.0326

Epoch 5: Val Loss = 1.3940

Epoch 6/20, Train Loss: 0.9252

Epoch 6: Val Loss = 1.5640

Epoch 7/20, Train Loss: 0.8860

Epoch 7: Val Loss = 1.3534

Epoch 8/20, Train Loss: 0.8376

Epoch 8: Val Loss = 1.5014

Epoch 9/20, Train Loss: 0.7719

Epoch 9: Val Loss = 1.8895

Epoch 10/20, Train Loss: 0.7527

Epoch 10: Val Loss = 2.0521

Epoch 11/20, Train Loss: 0.7323

Epoch 11: Val Loss = 2.2283

Epoch 12/20, Train Loss: 0.6979

Epoch 12: Val Loss = 2.5594

Epoch 13/20, Train Loss: 0.6886

Epoch 13: Val Loss = 2.4146

Epoch 14/20, Train Loss: 0.6713

Epoch 14: Val Loss = 2.5218

Epoch 15/20, Train Loss: 0.6493

Epoch 15: Val Loss = 2.4838

Epoch 16/20, Train Loss: 0.6434

Epoch 16: Val Loss = 2.5994

Epoch 17/20, Train Loss: 0.6401

Epoch 17: Val Loss = 2.6648

Epoch 18/20, Train Loss: 0.6236

Epoch 18: Val Loss = 2.7235

Epoch 19/20, Train Loss: 0.6239

Epoch 19: Val Loss = 2.7437

Epoch 20/20, Train Loss: 0.6285

Epoch 20: Val Loss = 2.6442

Final model saved.

Even simpler model, interupted due to overfitting happened (2025-04-22)

Training on device: cpu

Total training samples: 15403

Total validation samples: 5818

Epoch 1/20, Train Loss: 1.2686

Epoch 1: Val Loss = 5.8688

Model saved (best so far)

Epoch 2/20, Train Loss: 0.9182

Epoch 2: Val Loss = 13.8677

Epoch 3/20, Train Loss: 0.7523

Epoch 3: Val Loss = 16.5140

Epoch 4/20, Train Loss: 0.6599

Epoch 4: Val Loss = 18.0564

Epoch 5/20, Train Loss: 0.5652

Epoch 5: Val Loss = 23.0571

Epoch 6/20, Train Loss: 0.5264

Epoch 6: Val Loss = 22.7962

Epoch 7/20, Train Loss: 0.5048

Epoch 7: Val Loss = 25.0692

Epoch 8/20, Train Loss: 0.4642

Epoch 8: Val Loss = 29.3606

Abort lite-CNN, use model on 2025-04-20 (2025-04-22)

Training on device: cpu

Total samples found: 15403

Total samples found: 5818

Epoch 1/20, Train Loss: 1.3391

Epoch 1/20, Val Loss: 1.7908

Model saved as coevolution\_model.pth

Epoch 2/20, Train Loss: 1.1339

Epoch 2/20, Val Loss: 1.8868

Epoch 3/20, Train Loss: 1.0002

Epoch 3/20, Val Loss: 2.3373

Epoch 4/20, Train Loss: 0.9115

Epoch 4/20, Val Loss: 2.1538

Epoch 5/20, Train Loss: 0.7982

Epoch 5/20, Val Loss: 1.9837

Epoch 6/20, Train Loss: 0.7473

Epoch 6/20, Val Loss: 2.0029

Epoch 7/20, Train Loss: 0.6897

Epoch 7/20, Val Loss: 4.0714

Epoch 8/20, Train Loss: 0.6248

Epoch 8/20, Val Loss: 2.5048

Epoch 9/20, Train Loss: 0.5922

Epoch 9/20, Val Loss: 2.7479

Epoch 10/20, Train Loss: 0.5705

Epoch 10/20, Val Loss: 2.8861

Epoch 11/20, Train Loss: 0.5319

Epoch 11/20, Val Loss: 2.9879

Epoch 12/20, Train Loss: 0.5100

Epoch 12/20, Val Loss: 3.0384

Epoch 13/20, Train Loss: 0.4974

Epoch 13/20, Val Loss: 3.7052

Epoch 14/20, Train Loss: 0.4741

Epoch 14/20, Val Loss: 3.4380

Epoch 15/20, Train Loss: 0.4698

Epoch 15/20, Val Loss: 3.4438

Epoch 16/20, Train Loss: 0.4609

Epoch 16/20, Val Loss: 3.4544

Epoch 17/20, Train Loss: 0.4502

Epoch 17/20, Val Loss: 3.3693

Epoch 18/20, Train Loss: 0.4476

Epoch 18/20, Val Loss: 3.3608

Epoch 19/20, Train Loss: 0.4409

Epoch 19/20, Val Loss: 3.5387

Epoch 20/20, Train Loss: 0.4307

Epoch 20/20, Val Loss: 3.6446

Final model saved.

Two layers 3->32->64->64->6 (2025-04-25)

Training on device: cpu

Total samples found: 15403

Total samples found: 5818

Epoch 1/20, Train Loss: 1.4385

Epoch 1: Train Loss = 1.4385, Val Loss = 1.4889

Model saved (best so far)

Epoch 2/20, Train Loss: 1.4063

Epoch 2: Train Loss = 1.4063, Val Loss = 1.5123

Epoch 3/20, Train Loss: 1.3869

Epoch 3: Train Loss = 1.3869, Val Loss = 1.5367

Epoch 4/20, Train Loss: 1.3820

Epoch 4: Train Loss = 1.3820, Val Loss = 1.5607

Epoch 5/20, Train Loss: 1.3675

Epoch 5: Train Loss = 1.3675, Val Loss = 1.5629

Epoch 6/20, Train Loss: 1.3603

Epoch 6: Train Loss = 1.3603, Val Loss = 1.6760

Epoch 7/20, Train Loss: 1.3551

Epoch 7: Train Loss = 1.3551, Val Loss = 1.5812

Epoch 8/20, Train Loss: 1.3507

Epoch 8: Train Loss = 1.3507, Val Loss = 1.6392

two layers 3->16->32->64->6 (2025-04-25)

Training on device: cpu

Total samples found: 15403

Total samples found: 5818

Epoch 1/20, Train Loss: 1.4545

Epoch 1: Train Loss = 1.4545, Val Loss = 1.5647

Model saved (best so far)

Epoch 2/20, Train Loss: 1.4247

Epoch 2: Train Loss = 1.4247, Val Loss = 1.5311

Model saved (best so far)

Epoch 3/20, Train Loss: 1.4072

Epoch 3: Train Loss = 1.4072, Val Loss = 1.5282

Model saved (best so far)

Epoch 4/20, Train Loss: 1.3921

Epoch 4: Train Loss = 1.3921, Val Loss = 1.5443

Epoch 5/20, Train Loss: 1.3847

Epoch 5: Train Loss = 1.3847, Val Loss = 1.6737

Epoch 6/20, Train Loss: 1.3769

Epoch 6: Train Loss = 1.3769, Val Loss = 1.6640

Epoch 7/20, Train Loss: 1.3685

Epoch 7: Train Loss = 1.3685, Val Loss = 1.6985

Epoch 8/20, Train Loss: 1.3675

Epoch 8: Train Loss = 1.3675, Val Loss = 1.7177

Epoch 9/20, Train Loss: 1.3635

Epoch 9: Train Loss = 1.3635, Val Loss = 1.6400

three layers 3->16->32->32->64->6 (2025-04-25)

Training on device: cpu

Total samples found: 15403

Total samples found: 5818

Epoch 1/20, Train Loss: 1.4531

Epoch 1: Train Loss = 1.4531, Val Loss = 1.5203

Model saved (best so far)

Epoch 2/20, Train Loss: 1.3917

Epoch 2: Train Loss = 1.3917, Val Loss = 1.4767

Model saved (best so far)

Epoch 3/20, Train Loss: 1.3755

Epoch 3: Train Loss = 1.3755, Val Loss = 1.5998

Epoch 4/20, Train Loss: 1.3540

Epoch 4: Train Loss = 1.3540, Val Loss = 1.4877

Epoch 5/20, Train Loss: 1.3241

Epoch 5: Train Loss = 1.3241, Val Loss = 1.8422

Epoch 6/20, Train Loss: 1.2927

Epoch 6: Train Loss = 1.2927, Val Loss = 1.7859

Epoch 7/20, Train Loss: 1.2641

Epoch 7: Train Loss = 1.2641, Val Loss = 1.6622

Epoch 8/20, Train Loss: 1.2362

Epoch 8: Train Loss = 1.2362, Val Loss = 1.6344

Epoch 9/20, Train Loss: 1.1896

Epoch 9: Train Loss = 1.1896, Val Loss = 1.6324

Epoch 10/20, Train Loss: 1.1705

Epoch 10: Train Loss = 1.1705, Val Loss = 2.3218

Epoch 11/20, Train Loss: 1.1563

Epoch 11: Train Loss = 1.1563, Val Loss = 1.7357

Epoch 12/20, Train Loss: 1.1336

Epoch 12: Train Loss = 1.1336, Val Loss = 1.8508

Epoch 13/20, Train Loss: 1.1150

Epoch 13: Train Loss = 1.1150, Val Loss = 1.9085

Epoch 14/20, Train Loss: 1.1132

Epoch 14: Train Loss = 1.1132, Val Loss = 1.7888

Epoch 15/20, Train Loss: 1.1002

Epoch 15: Train Loss = 1.1002, Val Loss = 1.8138

Epoch 16/20, Train Loss: 1.0962

Epoch 16: Train Loss = 1.0962, Val Loss = 1.8256

Epoch 17/20, Train Loss: 1.0957

Epoch 17: Train Loss = 1.0957, Val Loss = 1.8713

one layer 3->16->32->64->6 (2025-04-26)

Training on device: cpu

Total samples found: 15403

Total samples found: 5818

Epoch 1/20, Train Loss: 1.3498

Epoch 1: Train Loss = 1.3498, Val Loss = 2.6168

Model saved (best so far)

Epoch 2/20, Train Loss: 1.1277

Epoch 2: Train Loss = 1.1277, Val Loss = 2.1644

Model saved (best so far)

Epoch 3/20, Train Loss: 0.9798

Epoch 3: Train Loss = 0.9798, Val Loss = 2.0495

Model saved (best so far)

Epoch 4/20, Train Loss: 0.8754

Epoch 4: Train Loss = 0.8754, Val Loss = 2.9551

Epoch 5/20, Train Loss: 0.7970

Epoch 5: Train Loss = 0.7970, Val Loss = 2.9895

Epoch 6/20, Train Loss: 0.7589

Epoch 6: Train Loss = 0.7589, Val Loss = 2.5299

Epoch 7/20, Train Loss: 0.6624

Epoch 7: Train Loss = 0.6624, Val Loss = 3.8500

Epoch 8/20, Train Loss: 0.6249

Epoch 8: Train Loss = 0.6249, Val Loss = 5.1187

# Appendix D

Peak of CPU utilisation and memory during testing.

|  | **CPU Utilisation (%)** | **Memory (%)** |
| --- | --- | --- |
| Before | 10 | 84 |
| After | 27 | 86 |
| Before | 8 | 83 |
| After | 19 | 86 |
| Before | 8 | 83 |
| After | 19 | 86 |
| Before | 8 | 83 |
| After | 20 | 85 |
| Before | 8 | 82 |
| After | 22 | 86 |
| Before | 13 | 83 |
| After | 23 | 86 |
| Before | 8 | 83 |
| After | 24 | 86 |
| Before | 8 | 82 |
| After | 28 | 86 |
| Before | 7 | 84 |
| After | 27 | 87 |
| Before | 8 | 84 |
| After | 29 | 87 |
| Before | 8 | 81 |
| After | 22 | 84 |
| Before | 9 | 81 |
| After | 26 | 85 |
| Before | 9 | 81 |
| After | 23 | 84 |
| Before | 10 | 81 |
| After | 24 | 84 |
| Before | 7 | 81 |
| After | 27 | 84 |

|  | **CPU Utilisation (%)** | **Memory (%)** |
| --- | --- | --- |
| Before | 7 | 81 |
| After | 27 | 84 |
| Before | 7 | 81 |
| After | 29 | 84 |
| Before | 5 | 81 |
| After | 29 | 84 |
| Before | 10 | 81 |
| After | 29 | 84 |
| Before | 11 | 80 |
| After | 25 | 84 |
| Average Before | 7.7 | 82 |
| Average After | 24.95 | 85.1 |