



Helmet Saftey System **(AllSafe)**

Graduation Project

For Faculty of Computers and Information Technology

The Egyptian E-learning University-Alexandria University
(Joint program)

By

Ramy Zaky Abd El Aty	(2000596)
Maram Amr Mohamad	(2000473)
Hussein Farag Fathy	(2000604)
Ahmed Nasser Fekry Abdullah	(2000283)
Mohamad El Sebaei Mahmoud	(2001263)
Yousef Mohamed Ramadan	(2000279)
Osama Elsayed Ibrahim	(2001208)

Supervised By

Dr. Rodaina Abdelsalam

TA. Rana

Assistant Professor at The Egyptian

Teaching Assistant at The Egyptian

E-learning University

E-learning University

2024

Contents Of Documentation

CHAPTER ONE: INTRODUCTION.....	4
1.1 Overview	4
1.2 Problem Definition.....	4
1.3 Proposed Solution	6
1.4 Challenges	8
CHAPTER TWO: RELATET WORK	9
2.1 Frist paper.....	9
2.2 Second Paper.....	10
2.3 Third Paper.....	10
2.4 Fourth paper	11
2.5 Fifth paper	11
CHAPTER THREE: LITERATURE REVIEW, ANALYSIS AND DESIGN.....	12
3.1 Overview:	12
3.2 System Analysis :.....	12
2.3 Use Case:.....	13
2.4 Sequence Diagram:	14
3.5 Flow Chart.....	15
CHAPTER FOUR: IMPLEMENTATIAN.....	16
4.1 Dataset.....	16
4.2 Introduction Yolo	16
What Makes YOLO Popular for Object Detection?	17
1. Speed:	18

2. High detection accuracy	19
3. Better generalization	19
4. Open source.....	19
4.3 Compare Between YOLOv7 And YOLOv8	21
4.4 What is YOLOv8?	22
4.5 Code of Project.....	26
4.6 Code API:.....	31
4.7 Testing And Result:	39
CHAPTER FIVE: WEBSITE IMPLEMENTATION	40
5.1 Overview	40
5.2 System Integration	40
5.3 Tools, Technologies and Techniques	40
5.4 Website Implementation	47
CHAPTER SIX: LASER SECURITY SYSYEM.....	51
6.1 Introduction:.....	51
6.2 Architecture Diagram:	51
6.3 Components List:	52
6.4 Arduino Code :.....	55
6.5 Testing and Troubleshooting :	55
6.6 Sample Arduino Code :.....	56
6.7 Explain The Previous Code:	57
7.8 Conclusion :	64
References	65

CHAPTER ONE: INTRODUCTION

1.1 Overview

Our project aims to protect workers by using artificial intelligence and a simple, user-friendly web interface to facilitate interaction. It detects whether workers are following standard safety procedures, such as wearing helmets and vests, because some workers find adhering to safety protocols tedious and believe it slows down their work, leading them to neglect these practices. The project can also be further developed in the future.

Users can utilize many features in our system:

- Create accounts on the website and log in
- Upload images for detection through the AI system
- Perform detection on videos
- Perform detection in real time

This enables us to use it in factories and construction sites to create a safer environment for workers.

1.2 Problem Definition

The primary goal of our safety system is to address the issue of non-compliance with safety protocols in industrial and construction environments. Despite the availability of safety guidelines, many workers tend to overlook or ignore these protocols, such as wearing helmets and vests, due to the perception that these measures are cumbersome and delay their tasks. This negligence can lead to serious accidents, injuries, and even fatalities, compromising the overall safety and productivity of the workplace.

Our system aims to solve the following problems:

1. Non-Compliance with Safety Protocols:

- Workers often neglect to follow safety procedures, believing them to be time-consuming and inconvenient.
- There is a need for a reliable method to ensure that safety guidelines are adhered to consistently.

2. Lack of Real-Time Monitoring:

- Traditional methods of safety compliance rely on periodic inspections and manual monitoring, which are not sufficient for ensuring continuous adherence to safety protocols.
- There is a requirement for a system that can provide real-time monitoring and immediate feedback.

3. Inefficient Incident Prevention:

- Without timely detection of non-compliance, the risk of accidents and injuries increases.
- An effective solution is needed to proactively prevent incidents rather than just responding to them after they occur.

4. User-Friendly Interface:

- Many safety systems are complex and difficult to use, leading to poor adoption and compliance.
- There is a need for an intuitive and easy-to-use interface that facilitates quick and efficient interaction for users of all technical levels.

1.3 Proposed Solution

To address the challenges outlined in the problem definition, we propose the implementation of an integrated safety system that leverages artificial intelligence (AI) and user-friendly web interfaces. Our solution encompasses the following components:

1. AI-Powered Compliance Detection:

- Develop AI algorithms capable of analyzing images and videos to detect whether workers are adhering to safety protocols, such as wearing helmets and vests.
- Utilize machine learning techniques to continuously improve the accuracy and efficiency of compliance detection.

2. Real-Time Monitoring System:

- Implement a real-time monitoring system that processes live video feeds from surveillance cameras installed in the workplace.
- Utilize computer vision algorithms to analyze video streams in real-time and identify instances of non-compliance with safety procedures.

3. User-Friendly Web Interface:

- Design and develop a user-friendly web interface that allows administrators and workers to interact with the safety system effortlessly.
- Provide intuitive features for uploading images, viewing real-time video feeds, and accessing compliance reports and alerts.

4. Alerting Mechanism:

- Integrate an alerting mechanism that notifies administrators and supervisors immediately when instances of non-compliance are detected.

- Enable customizable alert settings to ensure timely response to safety violations.

5. Scalability and Flexibility:

- Design the system architecture to be scalable and adaptable to different industrial and construction environments.
- Allow for easy integration with existing surveillance systems and safety management platforms.

6. Continuous Improvement and Updates:

- Establish a feedback loop for collecting data on safety incidents and user interactions with the system.
- Use this data to iteratively improve the AI algorithms, enhance user experience, and address emerging safety challenges.

By implementing this comprehensive solution, we aim to create a proactive safety management system that effectively addresses the issues of non-compliance with safety protocols, enhances workplace safety, and fosters a culture of safety awareness among workers.

1.4 Challenges

Implementing an effective safety system in industrial and construction environments presents several challenges, including:

1. Complexity of Work Environments:

- Industrial and construction sites often have complex layouts and dynamic work environments, making it challenging to deploy and manage safety monitoring systems effectively.

2. Variability in Safety Protocols:

- Safety protocols can vary widely across industries and regions, requiring the safety system to be adaptable and customizable to accommodate different standards and regulations.

3. Accuracy of AI Detection:

- Achieving high accuracy in AI-powered compliance detection poses a significant challenge, as the system must accurately differentiate between various safety equipment and detect subtle instances of non-compliance.

4. Real-Time Monitoring and Response:

- Ensuring real-time monitoring and immediate response to safety violations is essential but challenging, as any delays in detection or alerting could result in serious accidents or injuries.

CHAPTER TWO: RELATET WORK

2.1 Frist paper

Authors: Nipun D. Nath, Amir H. Behzadan, Stephanie G. Paal

Publication Date: April 2020

Algorithm: YOLOv3

In this section, we present the findings from our analysis on worker safety compliance, specifically focusing on the use of safety hats and vests. The data collected includes a total of 3,109 instances of workers, categorized based on their compliance with safety gear requirements.

Categories of Workers:

1. Unsafe Workers (UN W): Workers not wearing a hat or vest.

- Instances: 778

2. Workers Wearing Hat Only (WH):

- Instances: 1,436

3. Workers Wearing Vest Only (WV):

- Instances: 248

4. Workers Wearing Hat and Vest (WHV):

- Instances: 647

Accuracy of YOLOv3 in Detecting Safety Gear:

- Hat: 79.33%
- Vest: 83.01%
- Hat + Vest: 67.93%

2.2 Second Paper

Authors: Yange Li, Han Wei, Zheng Han, Jianling, Huang, Weidong Wang

Publication Date: Sept 2020

Framework : CNN(TensorFlow)

In this section, we present the findings from our analysis on worker safety compliance, specifically focusing on the use of safety Helmet. The data collected includes a total of 3261 instances of workers,

Categories of Workers:

1. Helmet

Accuracy : 80%

2.3 Third Paper

Authors: Federico Divina, Javier Alonso Ruiz, Jeroen Ploeg

Publication Date: April 2021

Algorithm: YOLOv3, YOLOv4

In this section, we present the findings from our analysis on worker safety compliance, specifically focusing on the use of safety Helmet and no Helmet. The data collected includes a total of 6063 instances of workers,

Categories of Workers:

1. Helmet, no Helmet

Accuracy:

- Yolov3: 75% , Yolov4: 79%

2.4 Fourth paper

Authors: Shilei Tan, Gonglin Lu,Ziqiang Jiang, Li Huang

Publication Date: March 2021

Algorithm: YOLOv5

In this section, we present the findings from our analysis on worker safety compliance, specifically focusing on the use of safety Helmet and no Helmet. The data collected includes a total of 6000 instances of workers,

Categories of Workers:

1. Helmet, no Helmet

Accuracy: 86%

2.5 Fifth paper

Authors: Desu Fu, Lin Gao, Tao Hu,Shukun Wang,Wei Liu

Publication Date: Feb 2022

Algorithm: YOLOv5

In this section, we present the findings from our analysis on worker safety compliance, specifically focusing on the use of safety Helmet and no Helmet. The data collected includes a total of 18,500 instances of workers,

Categories of Workers:

1. Helmet, no Helmet

Accuracy: 92%

By reviewing this paper, we found that we should focus on achieving the highest number of classes like the first paper while aiming for the highest accuracy. Additionally, through careful examination and analysis of all related work, we discovered that the YOLO algorithm yields the best results and the lowest error rate.

Therefore, we concentrated on YOLO and attempted to apply these findings in our project.

CHAPTER THREE: LITERATURE REVIEW, ANALYSIS AND DESIGN

3.1 Overview:

This chapter introduces an Overview of the Safety Detection System , It's a system that Analyzes, Detects, and Returns an accurate results Using the Yolo Algorithm. The system ensures the safety and security of work environments. It helps Workers and Employees Work safely.

The System provides a user-interface to interact with the machine learning algorithm and more details and information about the model.

Yolo Algorithm uses a variety of techniques to detect on image, videos and real time frame by frame

3.2 System Analysis :

1-Fuction requirements:

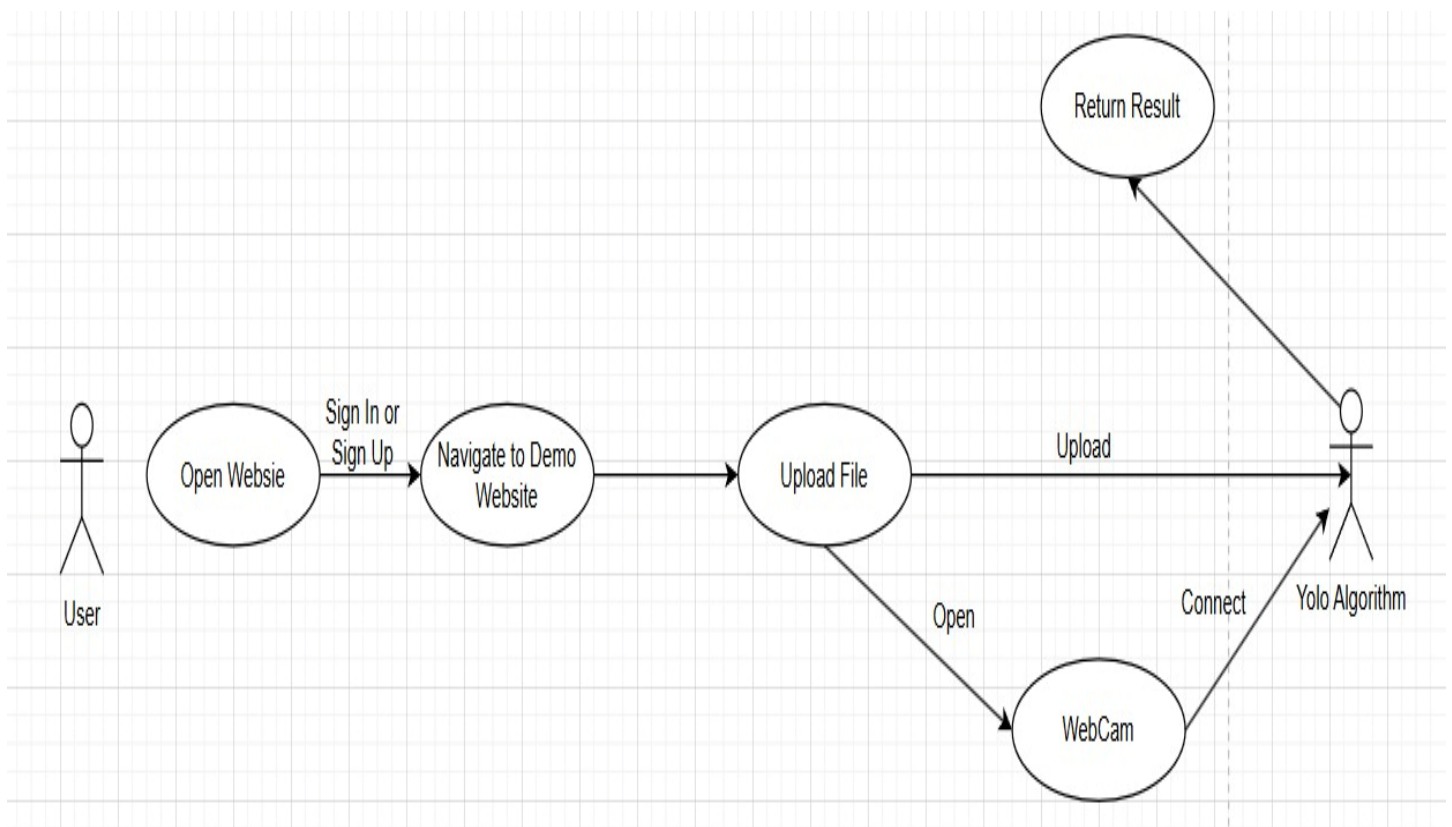
- a. User can Upload Images, Videos and open a real-life camera.
- b. the Algorithm detect the safety measures on the the uploaded image or video.
- c. Model returns an image or video with highlighted safety measurement.
- d. User can purchase different pricing plans.

2-Non-Fuction requirements:

- a. The system should have a Simple UI.
- b. The system should be easy to use & user-friendly.
- c. The system should be provide Security (Authentication & Authorization).
- d. The system should be compatible widely Accessible.

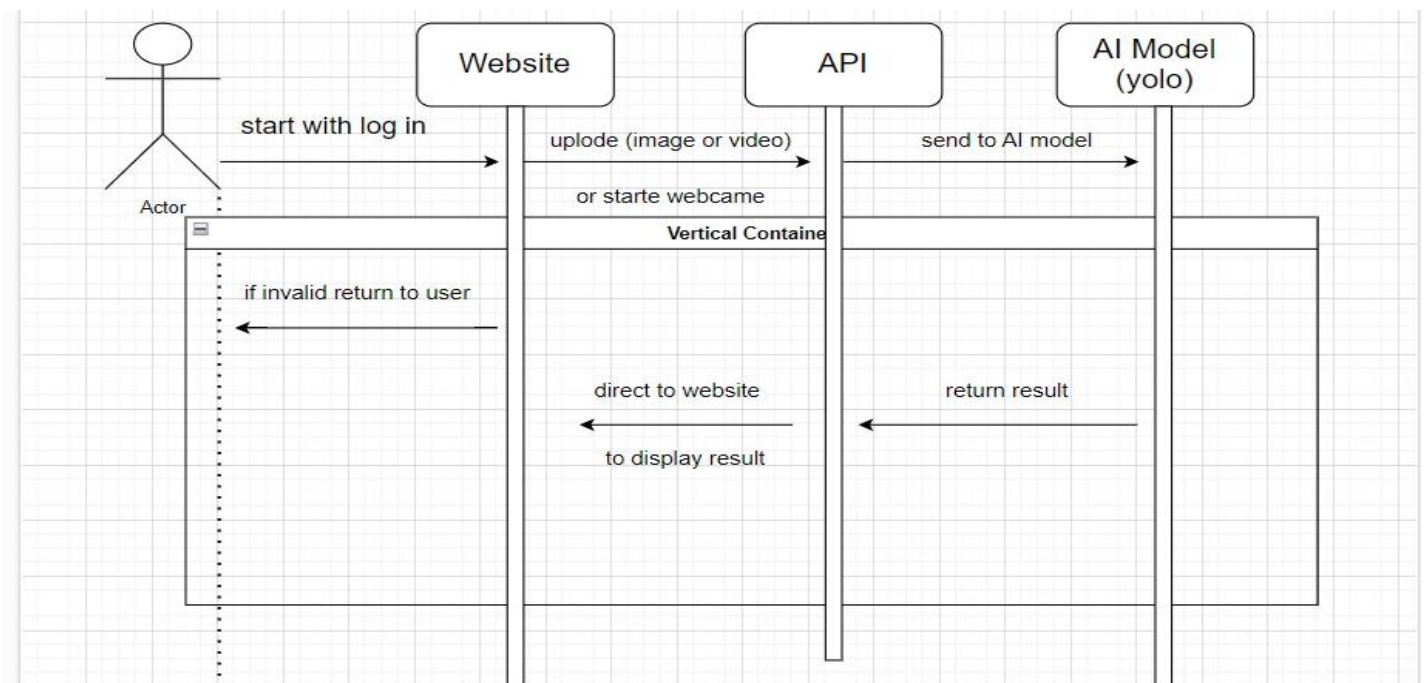
e. The system should be maintainable

2.3 Use Case:



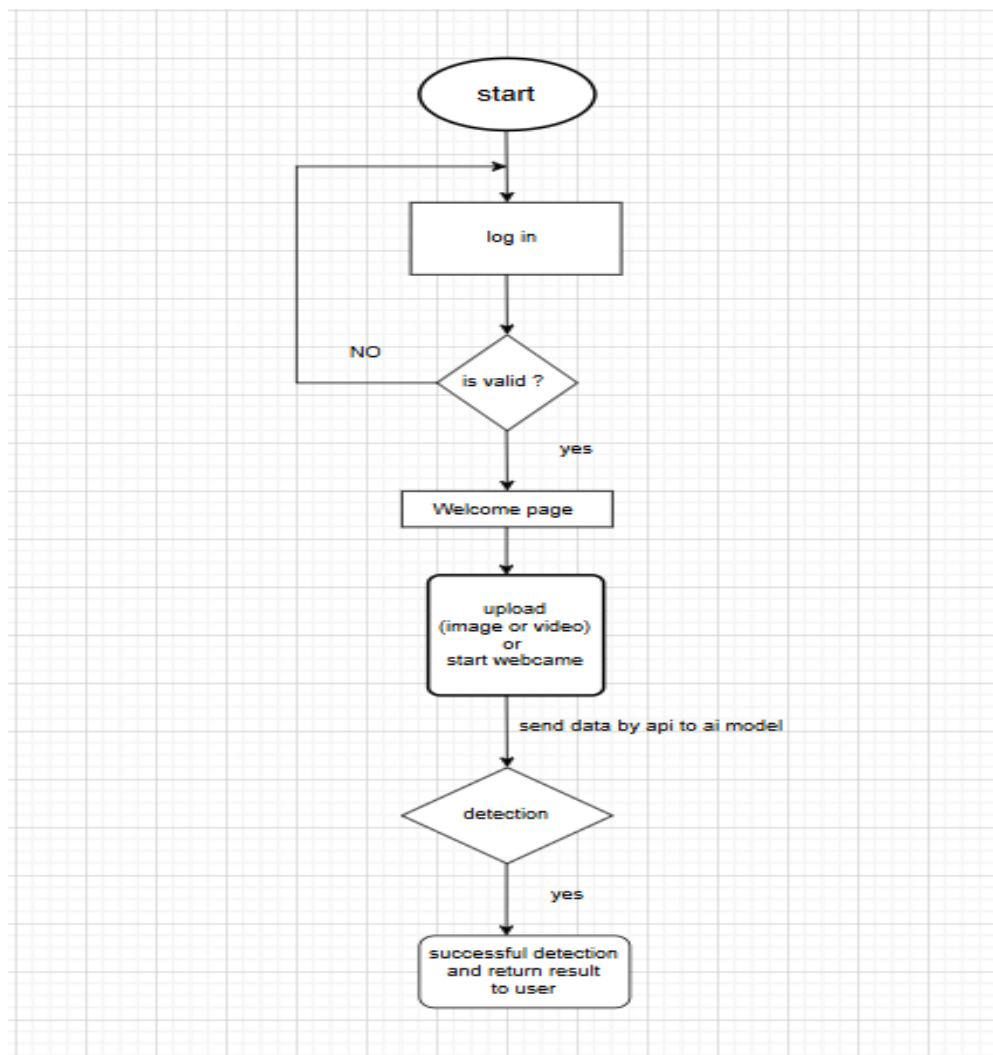
- User open the website and log in to be able to interact with the system. Then upload the required file (image or video) or open the camera . Ai model (YOLO) would Analyze the uploaded file frame by frame and return the result

2.4 Sequence Diagram:



The user interacts with the website after signed in then by uploading (images or videos) or start webcam and send data with api to ai model and detect and return result.

3.5 Flow Chart



This Flow Chart shows the path how the user interact with the website. Starting with the welcome page after signed in then uploading (images or videos) or start webcam
Then send data with api to ai model and detect it with ai model and response to the user

CHAPTER FOUR: IMPLEMENTATION

4.1 Dataset

From roboflow

<https://universe.roboflow.com/projet-dinitiation-s3gke/ppe-eulup/dataset/5>

Dataset : 5978 image

Class Names = ["Hardhat", "Safe worker", "Unsafe worker", "Vest", "Worker-only hat", "Worker-only vest"]

Accuracy of model : 95%

4.2 Introduction Yolo

Object detection is a technique used in computer vision for the identification and localization of objects within an image or a video.

Image Localization is the process of identifying the correct location of one or multiple objects using bounding boxes, which correspond to rectangular shapes around the objects.

This process is sometimes confused with image classification or image recognition, which aims to predict the class of an image or an object within an image into one of the categories or classes.

The illustration below corresponds to the visual representation of the previous explanation. The object detected within the image is “Person.”



In this conceptual blog, you will first understand the benefits of object detection, before introducing YOLO, the state-of-the-art object detection algorithm.

In the second part, we will focus more on the YOLO algorithm and how it works. After that, we will provide some real-life applications using YOLO.

The last section will explain how YOLO evolved from 2015 to 2020 before concluding on the next steps.

What Makes YOLO Popular for Object Detection?

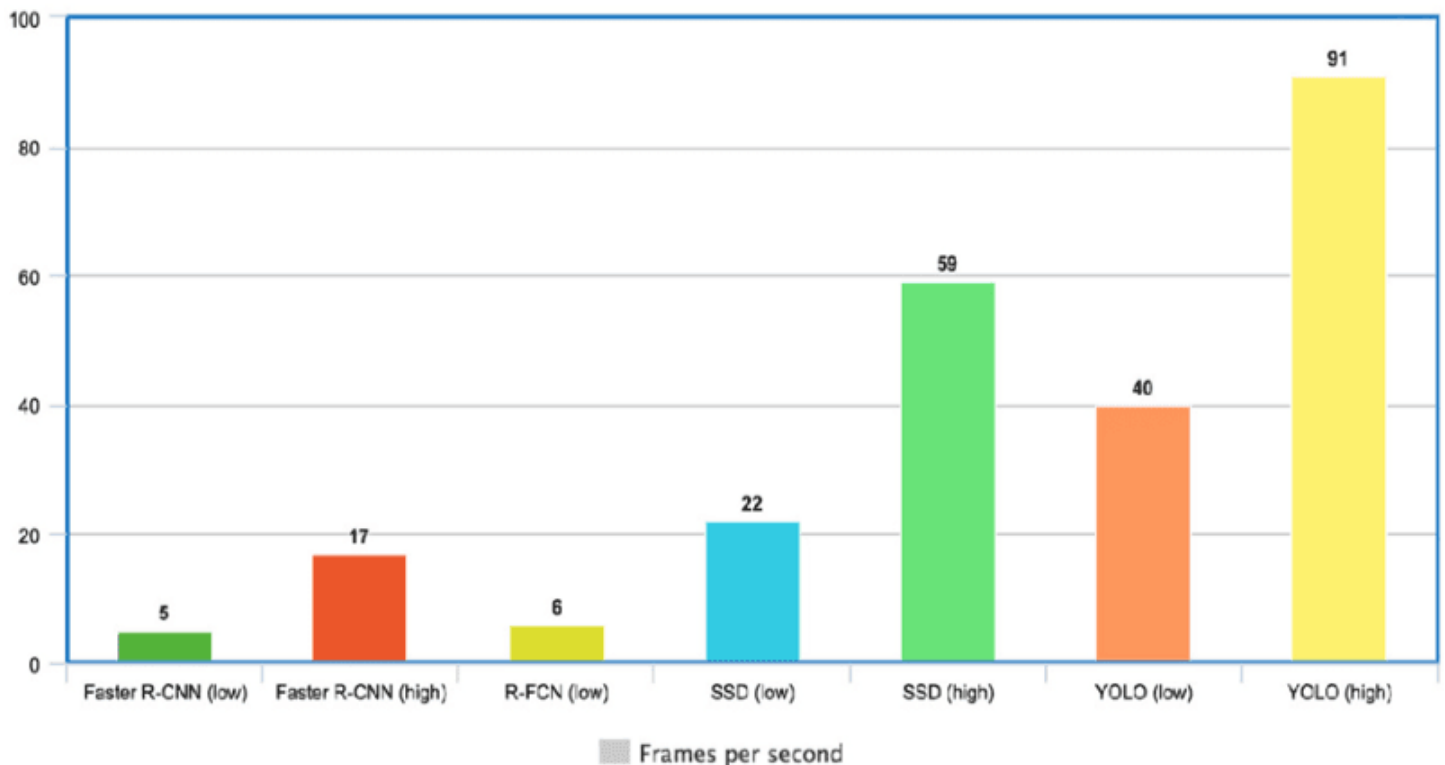
Some of the reasons why YOLO is leading the competition include its:

- Speed
- Detection accuracy
- Good generalization
- Open source

1. Speed:

YOLO is extremely fast because it does not deal with complex pipelines. It can process images at 45 Frames Per Second (FPS). In addition, YOLO reaches more than twice the mean Average Precision (mAP) compared to other real-time systems, which makes it a great candidate for real-time processing.

From the graphic below, we observe that YOLO is far beyond the other object detectors with 91 FPS.



YOLO Speed compared to other state-of-the-art object detectors

2. High detection accuracy

YOLO is far beyond other state-of-the-art models in accuracy with very few background errors.

3. Better generalization

This is especially true for the new versions of YOLO, which will be discussed later in the article. With those advancements, YOLO pushed a little further by providing a better generalization for new domains, which makes it great for applications relying on fast and robust object detection.

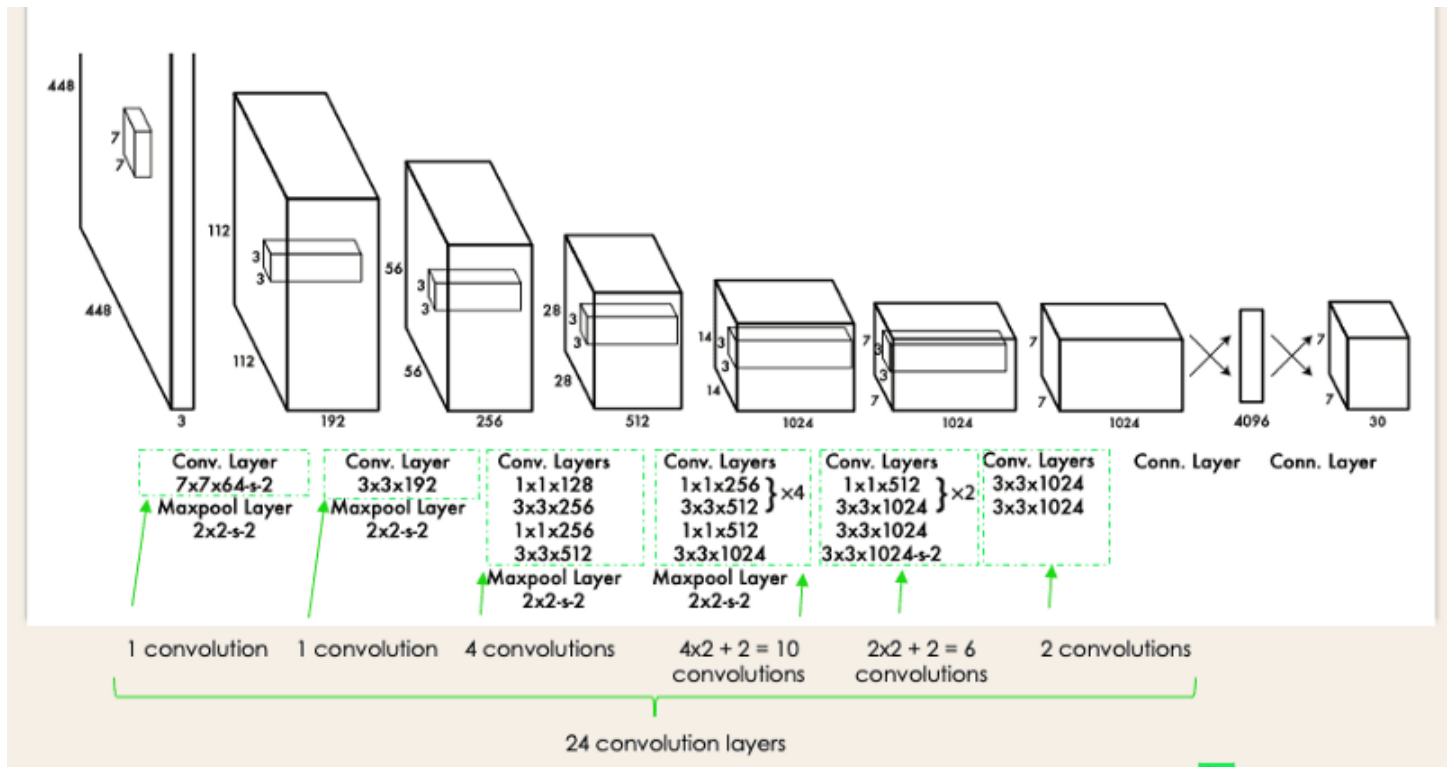
For instance, the Automatic Detection of Melanoma with Yolo Deep Convolutional Neural Networks paper shows that the first version YOLOv1 has the lowest mean average precision for the automatic detection of melanoma disease, compared to YOLOv2 and YOLOv3.

4. Open source

Making YOLO open-source led the community to constantly improve the model. This is one of the reasons why YOLO has made so many improvements in such a limited time.

YOLO Architecture

YOLO architecture is similar to Google Net. As illustrated below, it has overall 24 convolutional layers, four max-pooling layers, and two fully connected layers.



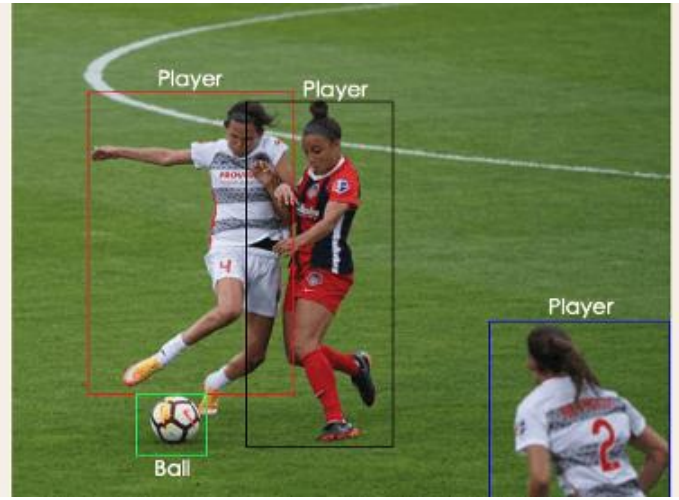
How Does YOLO Object Detection Work?

Now that you understand the architecture, let's have a high-level overview of how the YOLO algorithm performs object detection using a simple use case.

“Imagine you built a YOLO application that detects players and soccer balls from a given image.

But how can you explain this process to someone, especially non-initiated people?

→ That is the whole point of this section. You will understand the whole process of how YOLO performs object detection; how to get image (B) from image (A)



4.3 Compare Between YOLOv7 And YOLOv8

Feature	YOLOv5	YOLOv8
Release Date	2022	2023
Developer	WongKinYiu	Ultralytics
Architecture		Enhanced backbone, neck, and head

	Enhanced CSP backbone, PANet neck, E-ELAN head	
Accuracy	Very high	Higher than YOLOv5 And YOLOv7
Speed	Fast, optimized for real-time applications	Optimized for speed, often faster than YOLOv7
Sizes Supported	Anchor-based	Similar range with optimizations
Training Techniques	Advanced techniques like EMA, label smoothing, and CIOU loss	Advanced data augmentation, mosaic, self-adversarial training
Detection	Anchor-based	Anchor-free
Feature Extraction	Effective, balanced	Enhanced feature extraction and spatial information preservation
Deployment	Optimized for various hardware, efficient inference	Optimizations for various hardware
Community Support	Strong research community and adoption	Rapidly growing community and support
Use Cases	High-performance detection in diverse applications including surveillance, autonomous driving	High accuracy tasks, complex scenarios, medical imaging, surveillance

4.4 What is YOLOv8?

is the newest state-of-the-art YOLO model that can be used for object detection, image classification, and instance segmentation tasks. YOLOv8 was developed by Ultralytics, who also created the influential and industry-defining YOLOv5 model. YOLOv8 includes numerous architectural and developer experience changes and improvements over YOLOv5.

YOLOv8 is under active development as of writing this post, as Ultralytics work on new features and respond to feedback from the community. Indeed, when Ultralytics releases a model, it enjoys long-term support: the organization works with the community to make the model the best it can be.

Why Should I Use YOLOv8?

Here are a few main reasons why you should consider using YOLOv8 for your next computer vision project:

1. YOLOv8 has a high rate of accuracy measured by Microsoft COCO and Roboflow 100.
2. YOLOv8 comes with a lot of developer-convenience features, from an easy-to-use CLI to a well-structured Python package.
3. There is a large community around YOLO and a growing community around the YOLOv8 model, meaning there are many people in computer vision circles who may be able to assist you when you need guidance.

YOLOv8 achieves strong accuracy on COCO. For example, the YOLOv8m model -- the medium model -- achieves a 50.2% mAP when measured on COCO. When evaluated against Roboflow 100, a dataset that specifically evaluates model performance on various task-specific domains, YOLOv8 scored substantially better than YOLOv5. More information on this is provided in our performance analysis later in the article.

Furthermore, the developer-convenience features in YOLOv8 are significant. As opposed to other models where tasks are split across many different Python files that you can execute, YOLOv8 comes with a CLI that makes training a model more intuitive. This is in addition to a Python package that provides a more seamless coding experience than prior models.

YOLOv8 Architecture:

A Deep Dive YOLOv8 does not yet have a published paper, so we lack direct insight into the direct research methodology and ablation studies done during its creation. With that said, we analyzed the repository and information available about the model to start documenting what's new in YOLOv8.

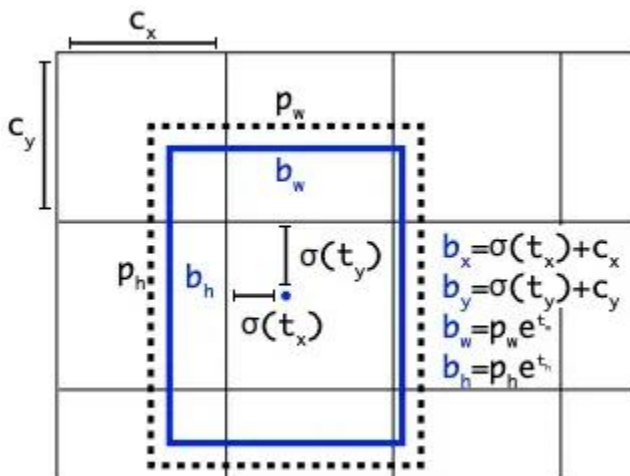
If you want to peer into the code yourself, check out the YOLOv8 repository and you view this code differential to see how some of the research was done.

Here we provide a quick summary of impactful modeling updates and then we will look at the model's evaluation, which speaks for itself.

The following image made by GitHub user RangeKing shows a detailed vizualisation of the network's architecture.

Anchor Free Detection

YOLOv8 is an anchor-free model. This means it predicts directly the center of an object instead of the offset from a known anchor box.



Visualization of an anchor box in YOLO

Anchor boxes were a notoriously tricky part of earlier YOLO models, since they may represent the distribution of the target benchmark's boxes but not the distribution of the custom dataset.

YOLOv8 COCO Accuracy

COCO (Common Objects in Context) is the industry standard benchmark for evaluating object detection models. When comparing models on COCO, we look at the mAP value and FPS measurement for inference speed. Models should be compared at similar inference speeds.

The image below shows the accuracy of YOLOv8 on COCO, using data collected by the Ultralytics team and published in their [YOLOv8 README](#):

▼ Detection

Model	size (pixels)	mAP ^{val} 50-95	Speed CPU (ms)	Speed T4 GPU (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	-	-	3.2	8.7
YOLOv8s	640	44.9	-	-	11.2	28.6
YOLOv8m	640	50.2	-	-	25.9	78.9
YOLOv8l	640	52.9	-	-	43.7	165.2
YOLOv8x	640	53.9	-	-	68.2	257.8

- mAP^{val} values are for single-model single-scale on [COCO val2017](#) dataset.
Reproduce by `yolo mode=val task=detect data=coco.yaml device=0`
- Speed averaged over COCO val images using an [Amazon EC2 P4d](#) instance.
Reproduce by `yolo mode=val task=detect data=coco128.yaml batch=1 device=0/cpu`

YOLOv8 COCO evaluation

YOLOv8 COCO accuracy is state of the art for models at comparable inference latencies as of writing this post.

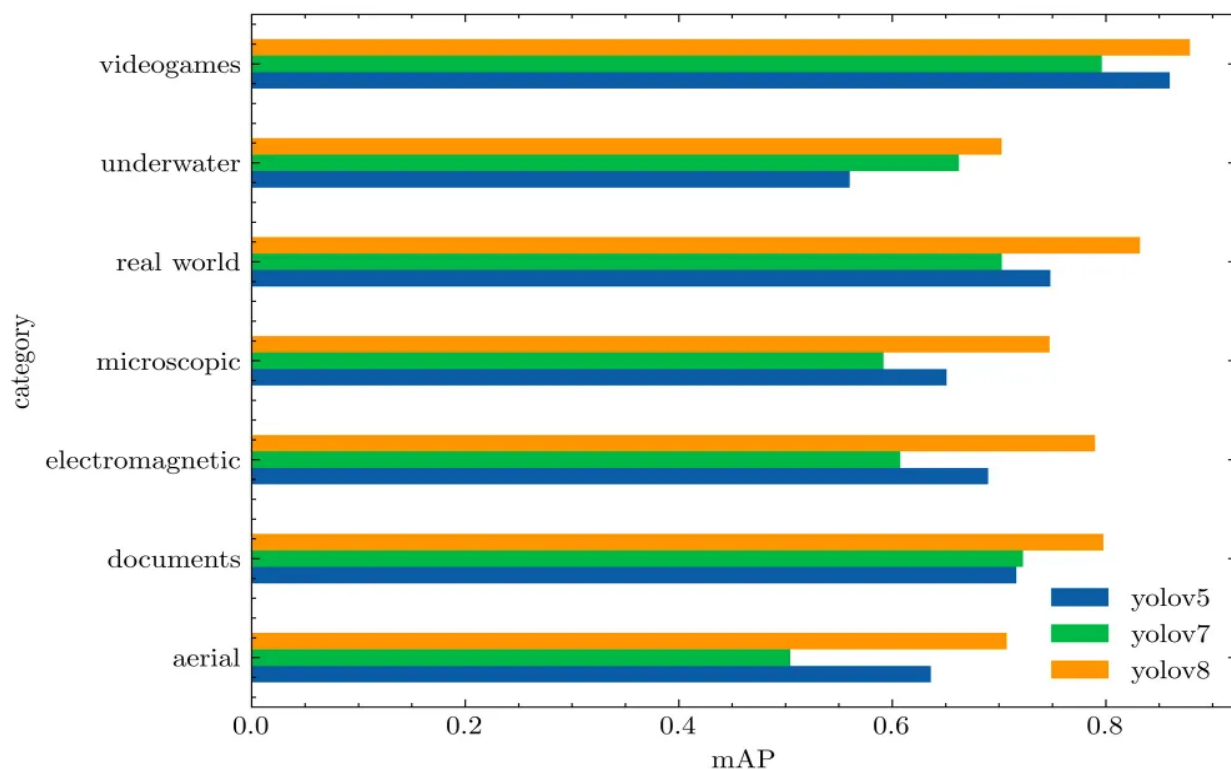
RF100 Accuracy

At Roboflow, we have drawn 100 sample datasets from [Roboflow Universe](#), a repository of over 100,000 datasets, to evaluate how well models generalize to new domains. Our benchmark, developed with support from Intel, is a benchmark for computer vision practitioners designed to provide a better answer to the question: "how well will this model work on my custom dataset?"

We evaluated YOLOv8 on our [RF100](#) benchmark alongside YOLOv5 and YOLOv7, the following box plots show each model's mAP@.50.

*We run the small version of each model for 100 epochs, we ran once with a single seed so due to gradient lottery take this result with a **grain of salt**.*

The box plot below tells us YOLOv8 had fewer outliers and an overall better mAP when measured against the Roboflow 100 benchmark.



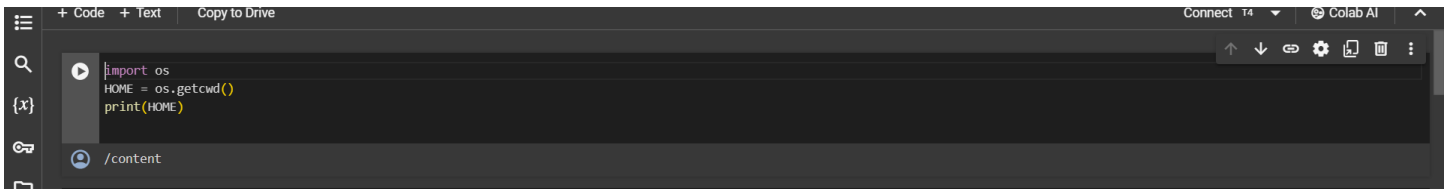
YOLOs average $mAP@.50$ against RF100 categories

Relative to the YOLOv5 evaluation, the YOLOv8 model produces a similar result on each dataset, or improves the result significantly.

4.5 Code of Project

Install library:

cvzone, ultralytics,
hydra-core, matplotlib
numpy , opencv-python
Pillow , PyYAML,
requests , scipy
torch , torchvision
tqdm , filterpy,
scikit-image , lap
pip install -r requirements. txt
from ultralytics import YOLO
we need to install this library first before start use YOLOV8



```
import os
HOME = os.getcwd()
print(HOME)
```

1. **import os**: This line imports the Python built-in module **os**, which provides a portable way of using operating system-dependent functionality. It allows Python scripts to interact with the underlying operating system, such as manipulating file paths, accessing environment variables, and executing system commands.
2. **HOME = os.getcwd()**: This line retrieves the current working directory using the **os.getcwd()** function and assigns it to the variable **HOME**.
 - **os.getcwd()** stands for "get current working directory". It returns a string representing the current working directory of the Python script.
 - The current working directory is the directory from which the Python script is being executed.
3. **print (HOME)**: This line prints the value stored in the variable **HOME** to the standard output.
 - **print ()** is a built-in Python function used to display the specified content (in this case, the value of the variable **HOME**) to the console.

```
!pip install ultralytics
from ultralytics import YOLO

Collecting ultralytics
  Downloading ultralytics-8.0.193-py3-none-any.whl (617 kB)
    617.3/617.3 kB 9.3 MB/s eta 0:00:00
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (3.7.1)
Requirement already satisfied: numpy>=1.22.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.23.5)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.8.0.76)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.4.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (6.0.1)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.31.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.11.3)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.0.1+cu118)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.15.2+cu118)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.66.1)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.5.3)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.12.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.0.0)
Collecting thop>=0.1.1 (from ultralytics)
  Downloading thop-0.1.1.post2209072238-py3-none-any.whl (15 kB)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (0.12.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (4.43.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.4.5)
```

- **!:** In a Jupyter Notebook or similar environment, the exclamation mark (!) at the beginning of a line indicates that the command should be executed in the system shell rather than in Python itself.
- **pip:** **pip** is the package installer for Python, used to install and manage Python packages.
- **install:** This is the command for **pip** to install a package.
- **ultralytics:** This is the name of the package being installed. In this case, it's the **ultralytics** package.

The **ultralytics** library is a deep learning toolkit that provides implementations of various computer vision models, including YOLO (You Only Look Once) for object detection.

```
#none https://universe.roboflow.com/ensam-melc0/ppe-detection-sk7bu/dataset/1
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="fyT7tsVcmGuYFLX050ov")
project = rf.workspace("ensam-melc0").project("ppe-detection-sk7bu")
dataset = project.version(1).download("yolov8")

Collecting roboflow
  Downloading roboflow-1.1.7-py3-none-any.whl (58 kB)
    58.8/58.8 kB 1.8 MB/s eta 0:00:00
Collecting certifi==2022.12.7 (from roboflow)
  Downloading certifi-2022.12.7-py3-none-any.whl (155 kB)
    155.3/155.3 kB 7.1 MB/s eta 0:00:00
Collecting chardet==4.0.0 (from roboflow)
  Downloading chardet-4.0.0-py2.py3-none-any.whl (178 kB)
    178.7/178.7 kB 20.3 MB/s eta 0:00:00
Collecting cycler==0.10.0 (from roboflow)
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
```

1. **!pip install roboflow:** This line is a command typically used in Jupyter Notebooks or Google Colab to install the Roboflow Python SDK via pip. It's used to ensure that the necessary package is installed before proceeding.
2. **from roboflow import Roboflow:** This imports the **Roboflow** class from the **roboflow** module. The **Roboflow** class is the main interface for interacting with Roboflow's API.

3. **rf = Roboflow(api_key="fyT7tsVCmGuYFLX050ov"):** This creates an instance of the **Roboflow** class, initializing it with an API key. The API key is used for authentication when making requests to the Roboflow API.
4. **project = rf.workspace("ensam-melc0").project("ppe-detection-sk7bu"):** This accesses a specific project within a Roboflow workspace. It identifies the project using the workspace name ("ensam-melc0") and the project slug ("ppe-detection-sk7bu").
5. **dataset = project.version(1).download("yolov8"):** This downloads a dataset version from the specified project. It selects version 1 of the dataset and specifies the format to download ("yolov8").

```

yolo task=detect mode=train model="/content/drive/MyDrive/Untitled folder/best.pt" data={dataset.location}/data.yaml epochs=80 imgsz=640
5         -1 1 1180672 ultralytics.nn.modules.conv.conv [256, 512, 3, 2]
6         -1 6 8396800 ultralytics.nn.modules.block.C2f [512, 512, 6, True]
7         -1 1 2360320 ultralytics.nn.modules.conv.conv [512, 512, 3, 2]
8         -1 3 4461568 ultralytics.nn.modules.block.C2f [512, 512, 3, True]
9         -1 1 656896 ultralytics.nn.modules.block.SPPF [512, 512, 5]
10        -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
11        [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]
12        -1 3 4723712 ultralytics.nn.modules.block.C2f [1024, 512, 3]
13        -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
14        [-1, 4] 1 0 ultralytics.nn.modules.conv.Concat [1]
15        -1 3 1247744 ultralytics.nn.modules.block.C2f [768, 256, 3]
16        -1 1 590336 ultralytics.nn.modules.conv.conv [256, 256, 3, 2]
17        [-1, 12] 1 0 ultralytics.nn.modules.conv.Concat [1]
18        -1 3 4592640 ultralytics.nn.modules.block.C2f [768, 512, 3]
19        -1 1 2360320 ultralytics.nn.modules.conv.conv [512, 512, 3, 2]
20        [-1, 9] 1 0 ultralytics.nn.modules.conv.Concat [1]
21        -1 3 4723712 ultralytics.nn.modules.block.C2f [1024, 512, 3]
22        [15, 18, 21] 1 5587426 ultralytics.nn.modules.head.Detect [6, [256, 512, 512]]
Model summary: 365 layers, 43634466 parameters, 43634450 gradients, 165.4 GFLOPs

Transferred 595/595 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train2', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
AMP: checks passed ✓
train: Scanning /content/ppe-5/train/labels.cache... 5220 images, 0 backgrounds, 0 corrupt: 100% 5220/5220 [00:00<?, ?it/s]
augmentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGrayscale(p=0.01), CLAHE(p=0.01, clip_limit=(1, 4.0), tile_grid_size=(8, 8))
val: Scanning /content/ppe-5/valid/labels.cache... 491 images, 0 backgrounds, 0 corrupt: 100% 491/491 [00:00<?, ?it/s]
Plotting labels to runs/detect/train2/labels.jpg...

```

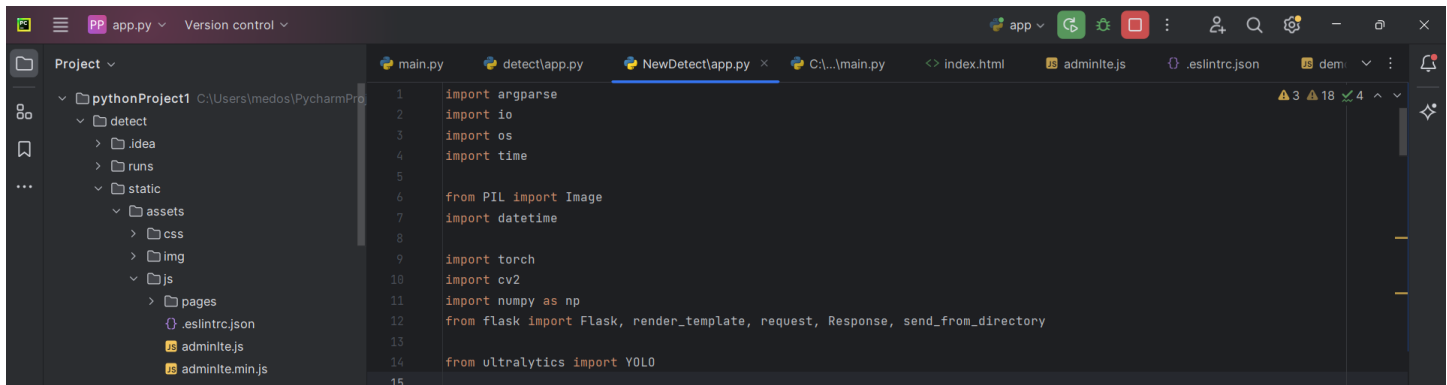
- **!yolo:** This indicates that you're using a command-line interface to interact with the YOLO tool.
- **task=detect:** Specifies that the task to be performed is object detection. This means the model will be trained to detect objects in images.
- **mode=train:** Indicates that the model will be trained. This is the training mode of operation for the YOLO tool.
- **model="/content/drive/MyDrive/Untitled folder/best.pt":** Specifies the path to the pre-trained model checkpoint file (**best.pt**) that will be used for training. The model will be initialized with the weights from this checkpoint before training begins.
- **data={dataset.location}/data.yaml:** Specifies the path to the YAML file (**data.yaml**) containing the dataset configuration. **{dataset.location}** is likely a placeholder that should be replaced with the actual path to the dataset directory.

- **epochs=80**: Sets the number of training epochs to 80. An epoch represents one complete pass through the entire training dataset during training.
- **imgsz=640**: Sets the size of the input images during training to 640x640 pixels. This is the resolution at which the images will be resized before being fed into the model during training.

```
[ ] !yolo task=detect mode=predict model=/content/runs/detect/train/weights/best.pt source="/content/6.jpg" conf=0.6
```

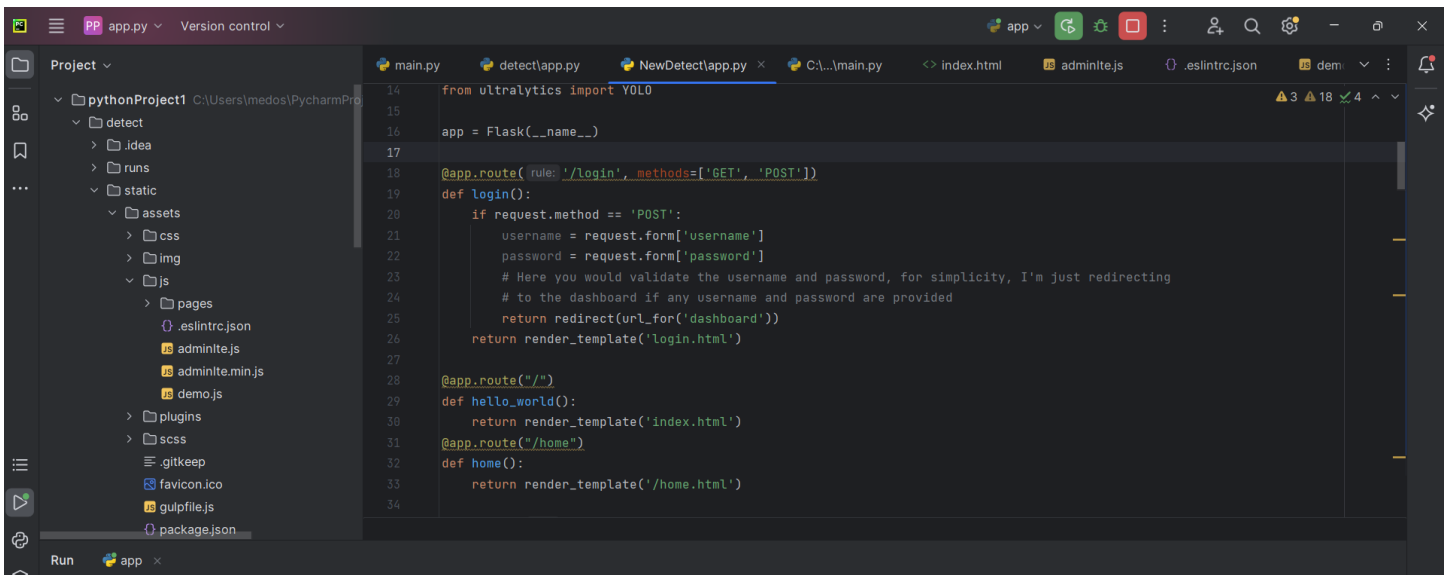
- **!yolo**: This indicates the start of the command, likely invoking the YOLO toolkit or library.
- **task=detect**: Specifies the task to be performed, which in this case is object detection.
- **mode=predict**: Indicates that the model should be used for prediction or inference.
- **model=/content/runs/detect/train/weights/best.pt**: Specifies the path to the YOLO model file (**best.pt**) that will be used for prediction. This is likely a pre-trained YOLO model stored in the directory **/content/runs/detect/train/weights/**.
- **source="/content/6.jpg"**: Specifies the input source for object detection. In this case, it's an image file located at **/content/6.jpg**. This image will be fed into the model for detection of objects.
- **conf=0.6**: Sets the confidence threshold for object detection. Objects with confidence scores greater than or equal to 0.6 will be considered valid detections. Lower confidence threshold may result in more detections but may also include more false positives.

4.6 Code API:



```
1 import argparse
2 import io
3 import os
4 import time
5
6 from PIL import Image
7 import datetime
8
9 import torch
10 import cv2
11 import numpy as np
12 from flask import Flask, render_template, request, Response, send_from_directory
13
14 from ultralytics import YOLO
15
```

- **argparse**: This module provides functions for parsing command-line arguments.
- **io**: This module provides tools for working with streams of data.
- **os**: This module provides a portable way to interact with the operating system.
- **time**: This module provides functions for working with time-related tasks.
- **PIL**: This is the Python Imaging Library, used here for working with images.
- **datetime**: This module provides classes for manipulating dates and times.
- **torch**: This is the PyTorch deep learning framework.
- **cv2**: This is the OpenCV library for computer vision tasks.
- **numpy**: This is the numerical computing library for working with arrays and matrices.
- **Flask**: This is a micro web framework for building web applications in Python.
- **send_from_directory**: This function sends a file from a given directory.
- **YOLO**: This is the YOLO object detection module from the Ultralytics library.



```
14 from ultralytics import YOLO
15
16 app = Flask(__name__)
17
18 @app.route(rule='/login', methods=['GET', 'POST'])
19 def login():
20     if request.method == 'POST':
21         username = request.form['username']
22         password = request.form['password']
23         # Here you would validate the username and password, for simplicity, I'm just redirecting
24         # to the dashboard if any username and password are provided
25         return redirect(url_for('dashboard'))
26     return render_template('login.html')
27
28 @app.route("/")
29 def hello_world():
30     return render_template('index.html')
31
32 @app.route("/home")
33 def home():
34     return render_template('home.html')
```

1. Initializing the Flask App:

This line creates a Flask application instance named **app**.

2. Login Route (/login):

This route handles both GET and POST requests to /login.

For GET requests, it renders a template named login.html, which presumably contains a login form.

For POST requests, it retrieves the username and password from the form data submitted by the user. In a real application, you would typically validate these credentials against a database or some other authentication mechanism. Here, it just redirects to the dashboard route without validation.

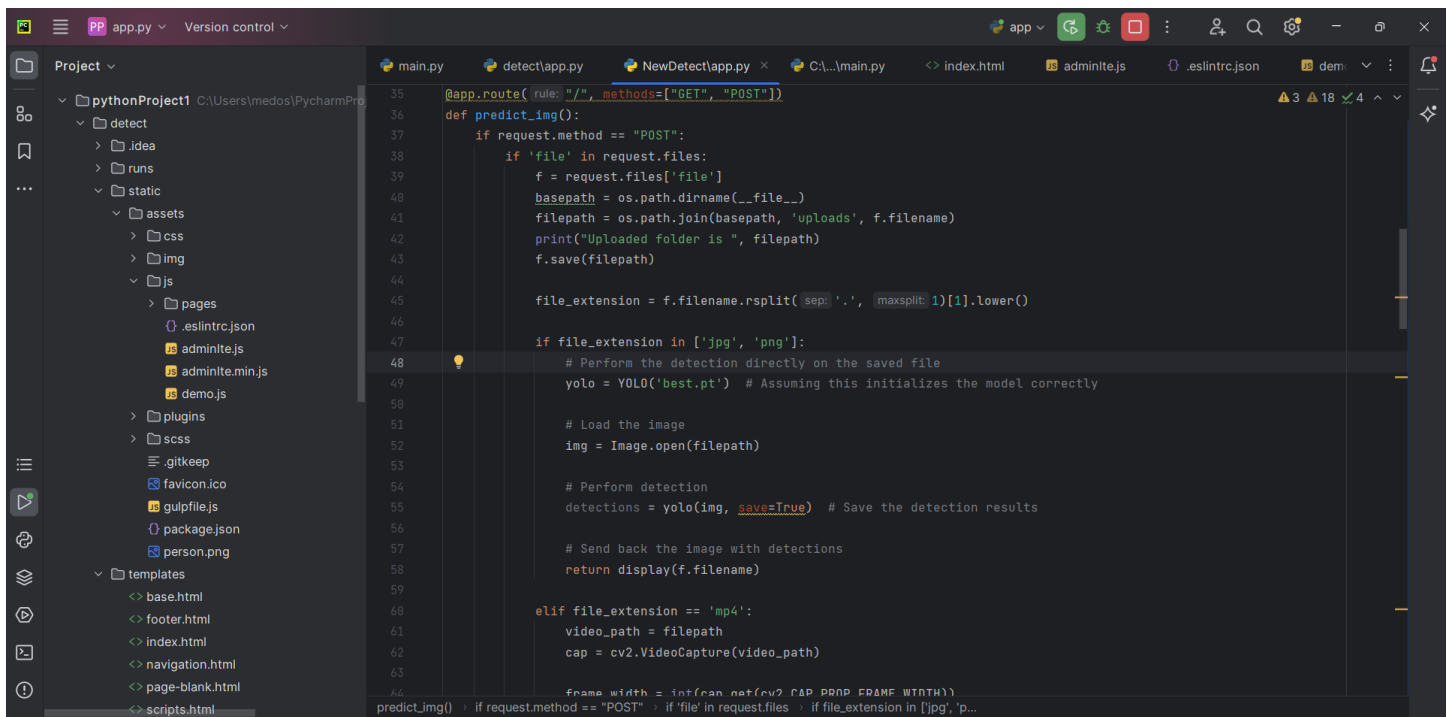
`redirect(url_for('dashboard'))` redirects the user to the dashboard route, which is not defined in the provided code.

3. Index Route (/):

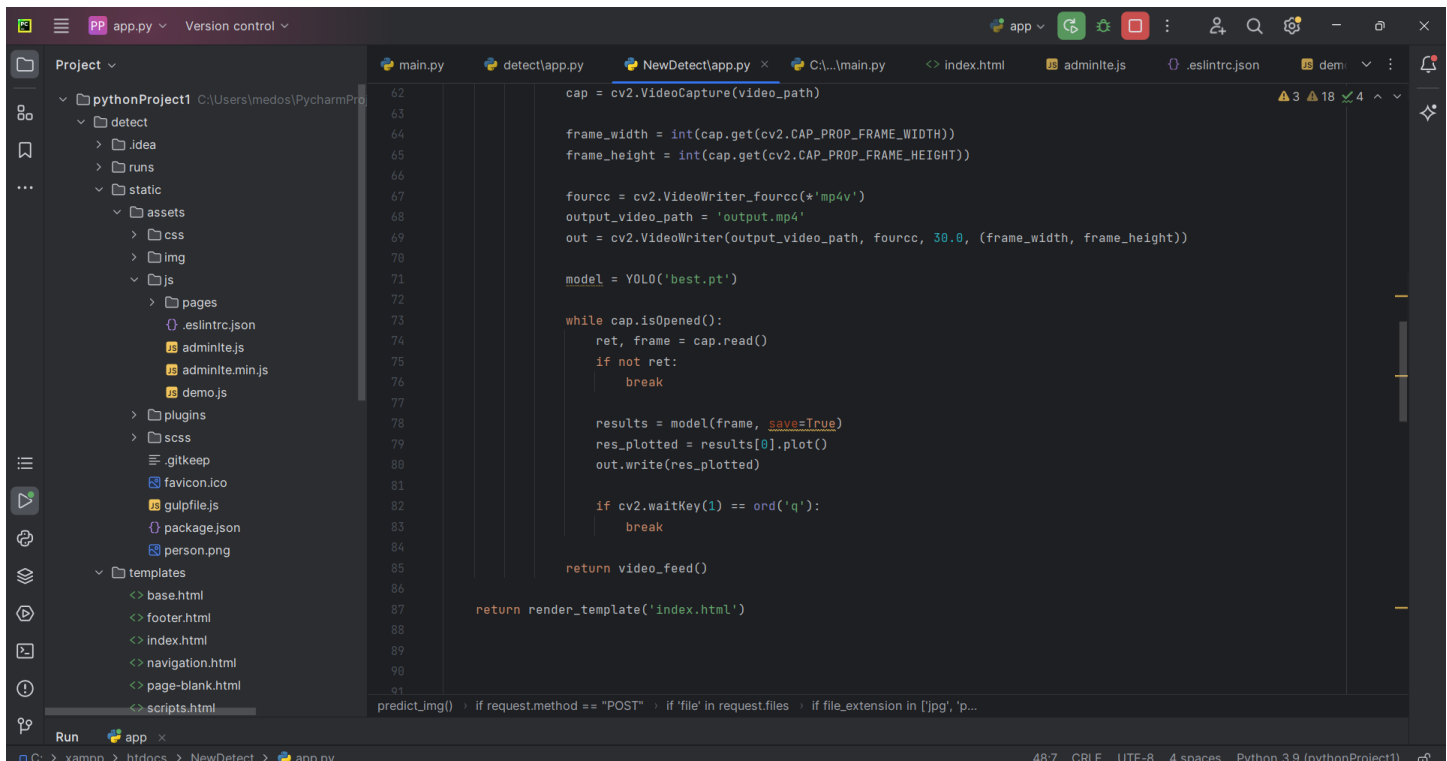
This route handles GET requests to the root URL /. It renders a template named index.html.

4. Home Route (/home):

This route handles GET requests to the /home URL. It renders a template named home.html.



```
35 @app.route(rule="/", methods=["GET", "POST"])
36 def predict_img():
37     if request.method == "POST":
38         if 'file' in request.files:
39             f = request.files['file']
40             basepath = os.path.dirname(__file__)
41             filepath = os.path.join(basepath, 'uploads', f.filename)
42             print("Uploaded folder is ", filepath)
43             f.save(filepath)
44
45             file_extension = f.filename.rsplit(sep='.', maxsplit=1)[1].lower()
46
47             if file_extension in ['jpg', 'png']:
48                 # Perform the detection directly on the saved file
49                 yolo = YOLO('best.pt') # Assuming this initializes the model correctly
50
51                 # Load the image
52                 img = Image.open(filepath)
53
54                 # Perform detection
55                 detections = yolo(img, save=True) # Save the detection results
56
57                 # Send back the image with detections
58                 return display(f.filename)
59
60             elif file_extension == 'mp4':
61                 video_path = filepath
62                 cap = cv2.VideoCapture(video_path)
63
64                 frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
65                 frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
66                 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
67                 output_video_path = 'output.mp4'
68                 out = cv2.VideoWriter(output_video_path, fourcc, 30.0, (frame_width, frame_height))
69
70                 model = YOLO('best.pt')
71
72                 while cap.isOpened():
73                     ret, frame = cap.read()
74                     if not ret:
75                         break
76
77                     results = model(frame, save=True)
78                     res_plotted = results[0].plot()
79                     out.write(res_plotted)
80
81                     if cv2.waitKey(1) == ord('q'):
82                         break
83
84                 return video_feed()
85
86             return render_template('index.html')
```



```
62 cap = cv2.VideoCapture(video_path)
63
64 frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
65 frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
66
67 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
68 output_video_path = 'output.mp4'
69 out = cv2.VideoWriter(output_video_path, fourcc, 30.0, (frame_width, frame_height))
70
71 model = YOLO('best.pt')
72
73 while cap.isOpened():
74     ret, frame = cap.read()
75     if not ret:
76         break
77
78     results = model(frame, save=True)
79     res_plotted = results[0].plot()
80     out.write(res_plotted)
81
82     if cv2.waitKey(1) == ord('q'):
83         break
84
85     return video_feed()
86
87 return render_template('index.html')
```

1. Route Definition:

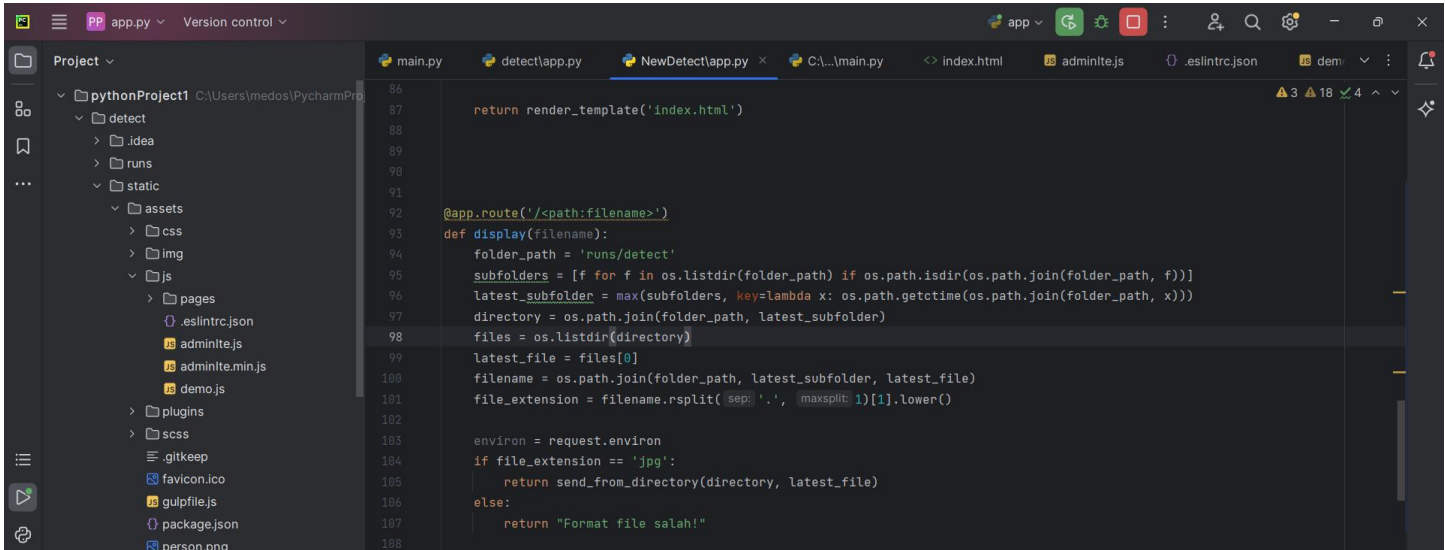
- `@app.route("/", methods=["GET", "POST"])`: Defines a route for the root URL ("/") of the web application. It accepts both GET and POST requests.

2. predict_img Function:

- This function handles the logic for uploading files and performing object detection.
- Checks if the request method is POST.
- If a file is included in the request, it saves the file to the server's filesystem in the 'uploads' directory.
- Determines the file extension to identify whether it's an image (JPEG or PNG) or a video (MP4).
- If it's an image:
 - Initializes a YOLO object detection model using the pre-trained weights file ('best.pt').
 - Loads the uploaded image using PIL (**Image.open()**).
 - Performs object detection on the image and saves the detection results.
 - Returns the filename of the uploaded image.
- If it's a video:
 - Opens the uploaded video file using OpenCV (**cv2.VideoCapture()**).
 - Initializes a YOLO object detection model using the pre-trained weights file ('best.pt').
 - Iterates through each frame of the video:
 - Performs object detection on the frame.
 - Plots the detection results on the frame.
 - Writes the frame with detection results to an output video file.
 - Returns a video feed of the processed video.

3. Return Statements:

- Returns the filename of the uploaded image or a video feed depending on the uploaded file type.
- If the request method is GET or if no file is uploaded, it renders the 'index.html' template.



```
86
87     return render_template('index.html')
88
89
90
91
92 @app.route('/<path:filename>')
93 def display(filename):
94     folder_path = 'runs/detect'
95     subfolders = [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, f))]
96     latest_subfolder = max(subfolders, key=lambda x: os.path.getctime(os.path.join(folder_path, x)))
97     directory = os.path.join(folder_path, latest_subfolder)
98     files = os.listdir(directory)
99     latest_file = files[0]
100     filename = os.path.join(folder_path, latest_subfolder, latest_file)
101     file_extension = filename.rsplit('.', 1)[1].lower()
102
103     environ = request.environ
104     if file_extension == 'jpg':
105         return send_from_directory(directory, latest_file)
106     else:
107         return "Format file salah!"
108
```

This decorates a function to be the handler for requests to the specified route, which includes a dynamic part **<path:filename>**. This means that any path after the initial route / will be captured and passed to the **filename** parameter of the function.

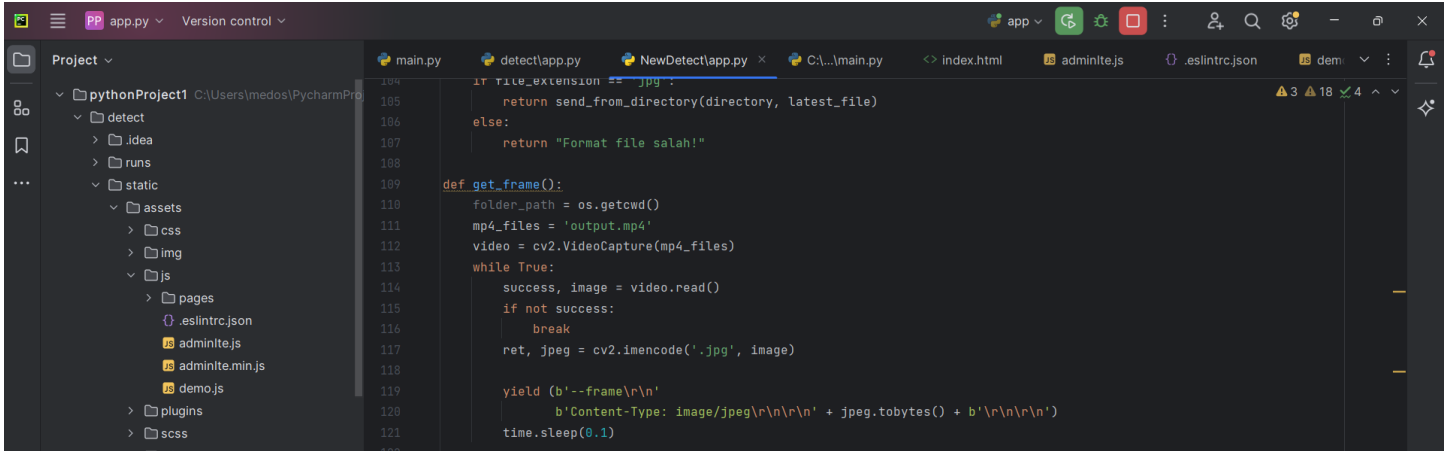
Here, the code sets up the directory path where the files are stored. It finds the most recent subfolder within the **runs/detect** directory by sorting the subfolders based on their creation time (**os.path.getctime**) and selecting the one with the maximum creation time. Then, it constructs the full path to the directory containing the latest files.

It lists all files within the latest directory and selects the first file as the **latest_file**. It constructs the full path to this file and extracts its extension, converting it to lowercase.

This line fetches the environment variables of the HTTP request using Flask's **request.environ**.

Finally, depending on the file extension, the function either serves the file using Flask's **send_from_directory** method, which sends a file from a given directory, or it returns an error message if the file format is not supported (in this case, only JPEG files are supported).

In summary, this Flask route handler dynamically serves the latest JPEG file from a specified directory (**runs/detect**). It ensures that the latest subfolder and file within that directory are selected before serving the file.



```
104 if file_extension == '.jpg':
105     return send_from_directory(directory, latest_file)
106 else:
107     return "Format file salah!"
108
109 def get_frame():
110     folder_path = os.getcwd()
111     mp4_files = 'output.mp4'
112     video = cv2.VideoCapture(mp4_files)
113     while True:
114         success, image = video.read()
115         if not success:
116             break
117         ret, jpeg = cv2.imencode('.jpg', image)
118
119         yield (b'--frame\r\n'
120               b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n\r\n')
121         time.sleep(0.1)
```

These lines import the necessary modules: **os** for interacting with the operating system, **cv2** for computer vision tasks, and **time** for introducing a delay.

This line defines the **get_frame()** function.

These lines:

- Get the current working directory using **os.getcwd()**.
- Specify the path to the video file (**output.mp4**), which is assumed to be in the current working directory.
- Create a **VideoCapture** object **video** to read frames from the video file.

This while loop:

Iterates continuously until there are no more frames to read (success becomes False).

Reads each frame from the video using **video.read()**.

Breaks out of the loop if there are no more frames (success is False).

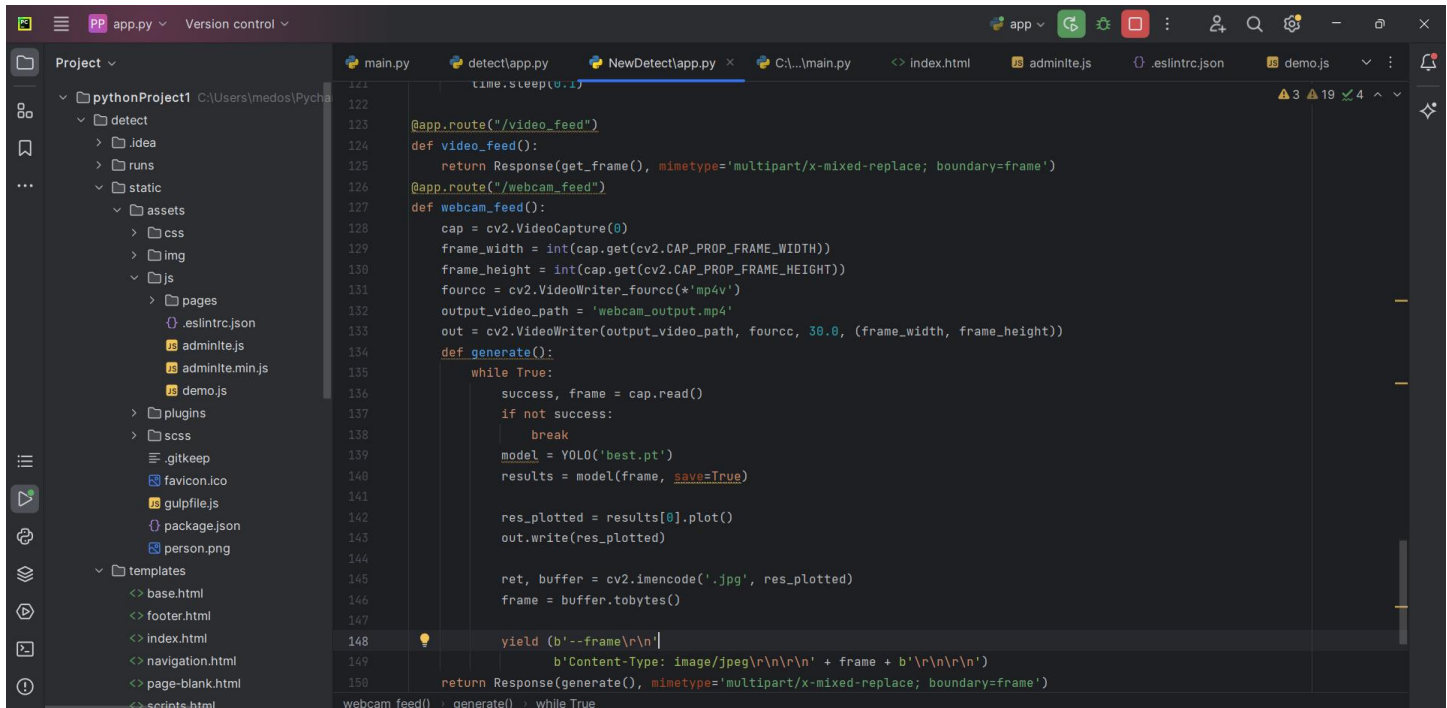
This line encodes the frame (**image**) into JPEG format using **cv2.imencode()**. It returns a tuple (**ret, jpeg**) where **ret** indicates whether the encoding was successful (usually **True**) and **jpeg** contains the JPEG-encoded image data.

This **yield** statement:

- Generates a byte string representing an HTTP multipart response, indicating the start of a new frame (**--frame**) and specifying the content type as JPEG (**Content-Type: image/jpeg**).

- Concatenates the JPEG-encoded image data (**jpeg.tobytes()**) within the multipart response.
- Delays execution for a short period (0.1 seconds) using **time.sleep(0.1)** to control the frame rate.

This line introduces a delay of 0.1 seconds between each frame to control the frame rate of the video stream.



```

121 time.sleep(0.1)
122
123 @app.route("/video_feed")
124 def video_feed():
125     return Response(get_frame(), mimetype='multipart/x-mixed-replace; boundary=frame')
126
127 @app.route("/webcam_feed")
128 def webcam_feed():
129     cap = cv2.VideoCapture(0)
130     frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
131     frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
132     fourcc = cv2.VideoWriter_fourcc(*'mp4v')
133     output_video_path = 'webcam_output.mp4'
134     out = cv2.VideoWriter(output_video_path, fourcc, 30.0, (frame_width, frame_height))
135
136     def generate():
137         while True:
138             success, frame = cap.read()
139             if not success:
140                 break
141             model = YOLO('best.pt')
142             results = model(frame, save=True)
143
144             res_plotted = results[0].plot()
145             out.write(res_plotted)
146
147             ret, buffer = cv2.imencode('.jpg', res_plotted)
148             frame = buffer.tobytes()
149
150             yield (b'--frame\r\n'
151                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
152
153     return Response(generate(), mimetype='multipart/x-mixed-replace; boundary=frame')
154
155 webcam_feed()
156 generate()
157 while True:

```

This route (**/video_feed**) returns a response that streams video frames from a source (likely a camera feed). It calls the **get_frame()** function to retrieve video frames, and the **Response** object is used to send these frames as a multipart response. The **mimetype** is specified as **'multipart/x-mixed-replace; boundary=frame'**, indicating that multiple parts of the response will be sent with each part representing a new frame.

This route (**/webcam_feed**) streams video from the webcam while performing real-time object detection using a YOLO model. Here's what happens in the function:

- It initializes the webcam capture using OpenCV (**cv2.VideoCapture(0)**).
- The dimensions of the frame are retrieved (**frame_width** and **frame_height**).
- A video writer is initialized to save the processed video.
- Inside the **generate()** function:
 - It continuously reads frames from the webcam (**cap.read()**).
 - For each frame:

- Object detection is performed using a YOLO model loaded from **best.pt**.
- The detected objects are plotted on the frame, and the result is saved to the output video.
- The frame is encoded as a JPEG image (**cv2.imencode('.jpg', res_plotted)**) for streaming.
- The encoded frame is yielded as part of the multipart response.

```
152 if __name__ == "__main__":  
153     parser = argparse.ArgumentParser(description="Flask app exposing yolov8 models")  
154     parser.add_argument("--port", default=5000, type=int, help="port number")  
155     args = parser.parse_args()  
156  
157     model = YOLO('best.pt')  
158     app.run(host="0.0.0.0", port=args.port)
```

This line checks whether the script is being run directly as the main program. If it is, the following code block will be executed.

Here, an argument parser (**argparse.ArgumentParser**) is initialized to handle command-line arguments. It defines a single argument **--port**, which specifies the port number on which the Flask app will listen for incoming requests. The default port number is set to 5000. When the script is run, it parses the command-line arguments provided and stores the values in the **args** variable.

This line initializes a YOLOv8 object detection model using the pre-trained weights loaded from the file named '**best.pt**'. The model will be used for making predictions on images sent to the Flask app.

Finally, this line starts the Flask web application. The **run** method is called on the Flask app (**app**). It specifies the host IP address as "**0.0.0.0**", which means the app will be accessible from any external IP address, and the port number is set to the value provided via the **--port** command-line argument or the default value of 5000 if no port is specified.

4.7 Testing And Result:



CHAPTER FIVE: WEBSITE IMPLEMENTATION

5.1 Overview

This chapter introduces the project components' implementation processes and the user Interface by which the user interacts with the AI model using various programming languages, techniques and tools.

5.2 System Integration

Integrating various system components through three major Parts:

Part 1:- web application system

Part 2:- Application Programming Interface (API)

Part 3:- AI Model (yolo)

5.3 Tools, Technologies and Techniques

PyCharm

An Integrated Development Environment (IDE) which will be used to develop the AI model using the Python programming language providing and it includes a built-in server for local testing Debugging services. It supports web development frameworks such as Flask, enabling seamless integration of web templates.

HTML (Hypertext Markup Language):

HTML forms the backbone of web pages, providing the structure and content of the UI. It uses tags to define different elements such as headings, paragraphs, images, buttons, and input fields. HTML is essential for creating the layout and organizing the various components of your web application.

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css"
      integrity="sha384-iYQeCzEYFbkjA/T2uDLTpkwGzCiq6soy8tYaI1GyVh/UjpbCx/TYkiZhlZB6+fzT"
      crossorigin="anonymous" />
    <link rel="stylesheet" href="/static/assets/css/style.css">

    <title>index</title>
  </head>

  <body>
    <nav class="navbar navbar-expand-lg text-white bg-dark bg-opacity-50 fixed-top py-3" id="navbar">
      <div class="container">
        <a class="h2 m-0 fw-bold text-decoration-none logo" href="/home">AllSafe</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
          data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
          aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
```

CSS (Cascading Style Sheets):

CSS is used to style and design the appearance of HTML elements. It allows you to define colors, fonts, spacing, layout, and other visual aspects of the UI. By applying CSS rules to HTML elements, you can customize the look and feel of your web application, ensuring a visually appealing and consistent user experience across different devices.

```
:root {
  --color-main: #ffcb0f;
}

* {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}

nav a.active {
  color: var(--color-main) !important;
  font-weight: 700;
}

.home {
  background-image: linear-gradient(#000a, rgba(130, 115, 115, 0.667)),
    url(./images/img3.jpg);
  background-size: cover;
}
```

Bootstrap:

Bootstrap is a popular front-end framework that provides pre-built components and styles for responsive web design. It offers a library of CSS classes and JavaScript plugins that streamline UI development. With Bootstrap, you can quickly create responsive layouts, navigation bars, buttons, forms, and other UI elements, saving time and effort in the design process.

```
<!-- Bootstrap 4 -->
<link rel="stylesheet" href="/static/assets/plugins/tempusdominus-bootstrap-4/css/tempusdominus-bootstrap-4.min.css">
<!-- iCheck -->
<link rel="stylesheet" href="/static/assets/plugins/ichack-bootstrap/ichack-bootstrap.min.css">
{% endblock stylesheets %}

{% block content %}
|
<div class="content-wrapper">

  <!-- Content Header (Page header) -->
  <div class="content-header">
    <div class="container-fluid">
      <div class="row mb-2">
        <div class="col-sm-6">
          <h1 class="m-0 text-dark">Y0L0v8 Dashboard - Personal Protective Equipment Detection</h1>
        </div><!-- /.col -->
        <div class="col-sm-6">
          <ol class="breadcrumb float-sm-right">
            <li class="breadcrumb-item">Home</li>
            <li class="breadcrumb-item active"><a href="/home.html">View Detected Image</a></li>
          </ol>
        </div>
      </div>
    </div>
  </div>
</div>
```

Flask (Python Web Framework):

Flask is a lightweight and flexible web framework for Python, used for building web applications. It integrates seamlessly with HTML templates, allowing you to generate dynamic content and render HTML pages based on user interactions or backend data. Flask handles routing, request handling, and template rendering, making it an ideal choice for developing UI components in Python-based web applications.

```
from flask import Flask, render_template, request, Response, send_from_directory, redirect, session, url_for

@app.route('/login', methods=['Get', 'Post'])
def login():
    form = LoginForm()

    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user:
            if check_password_hash(user.password, form.password.data):
                login_user(user, remember=form.remember.data)
                return redirect(url_for('aa'))

        return '<h1>Invalid Username or password</h1>'
        # return '<h1>' + form.username.data + ' ' + form.password.data + '</h1>'

    return render_template(template_name_or_list='login.html', form=form)

@app.route("/")
@login_required
def home():
    return render_template('home.html')

@app.route("/home")
@login_required
def aa():
    return render_template('home.html')
```

Flask Extensions:

1. Flask-WTF:

Flask-WTF simplifies working with web forms in Flask applications by integrating WTForms, a powerful form handling library for Python. It allows developers to define forms as Python classes, streamlining form creation, validation, and rendering processes. Flask-WTF also provides built-in protection against Cross-Site Request Forgery (CSRF) attacks, ensuring the security of form submissions in Flask applications.

2. Flask-Login:

Flask-Login enhances Flask by providing user session management capabilities, specifically designed for handling user authentication and authorization. It simplifies the implementation of user login/logout functionality, session tracking, and access control in Flask applications. Flask-Login facilitates user authentication processes, allowing developers to focus on building secure and feature-rich web applications without worrying about low-level session management tasks.

3. Flask-Bootstrap:

Flask-Bootstrap integrates Bootstrap, a popular front-end framework, with Flask, simplifying the creation of visually appealing and responsive web interfaces. It provides pre-built templates and helper functions, streamlining the incorporation of Bootstrap components into Flask applications. This extension facilitates the development of modern and user-friendly web interfaces within Flask, offering flexibility for customization and extension.

4. Flask-SQLAlchemy:

Flask-SQLAlchemy integrates SQLAlchemy, a powerful SQL toolkit and ORM library, with Flask, enabling seamless interaction with databases in Flask applications. It simplifies database operations by allowing developers to define database models as Python classes and perform CRUD (Create, Read, Update, Delete) operations using high-level Python APIs. Flask-SQLAlchemy also provides support for database migrations, query building, and transaction management, making it a versatile tool for building database-driven Flask applications

```
from flask import Flask, render_template, request, Response, send_from_directory, redirect, session, url_for
from flask_bootstrap import Bootstrap
from flask_sqlalchemy import SQLAlchemy
from flask_wtf import FlaskForm
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user

app = Flask(__name__)
Bootstrap(app)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db = SQLAlchemy(app)
app.secret_key = 'secret_key'
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(15), unique=True)
    email = db.Column(db.String(20), unique=True)
    password = db.Column(db.String(80))

with app.app_context():
    db.create_all()

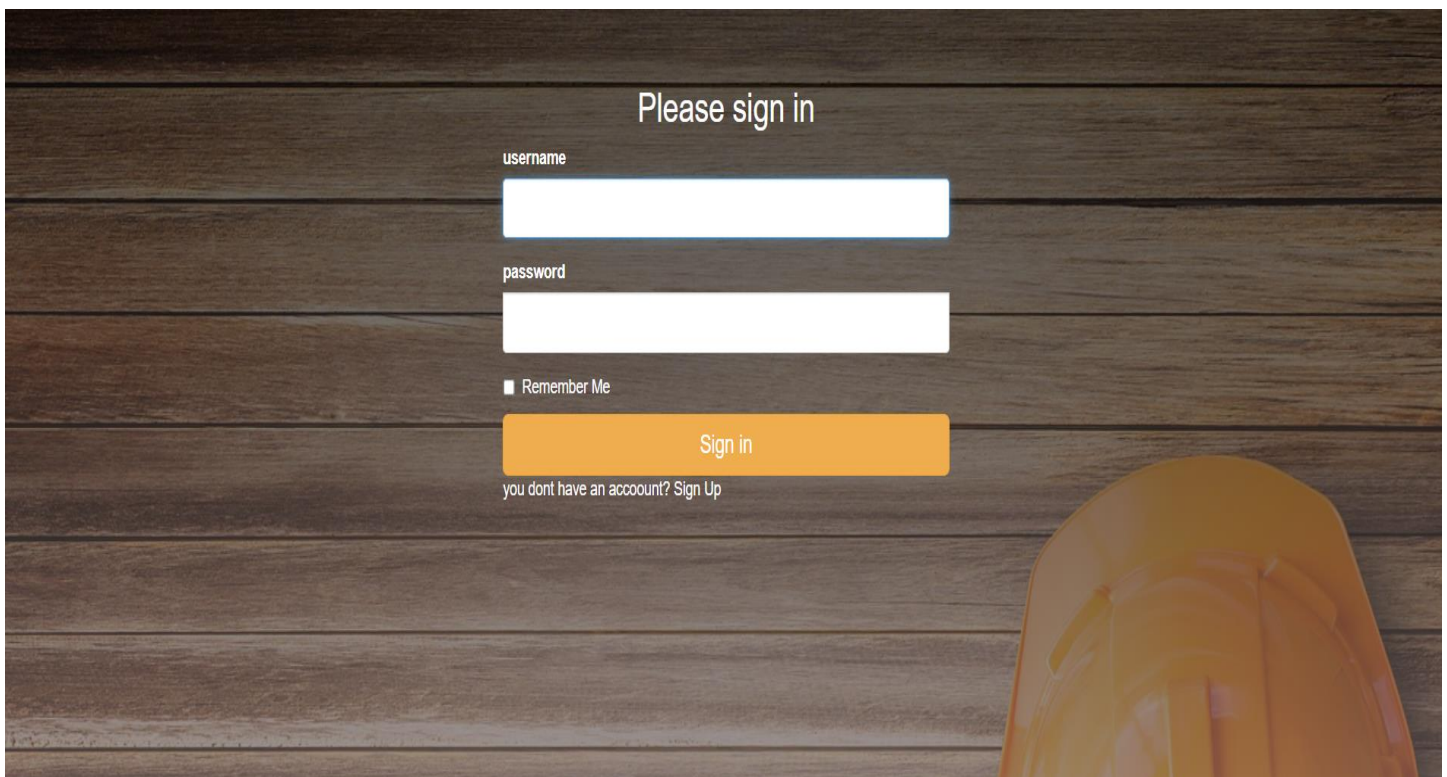
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

5.4 Website Implementation

The Website is built using Html, CSS and Bootstrap, SQLAlchemy and flask to integrate the pages

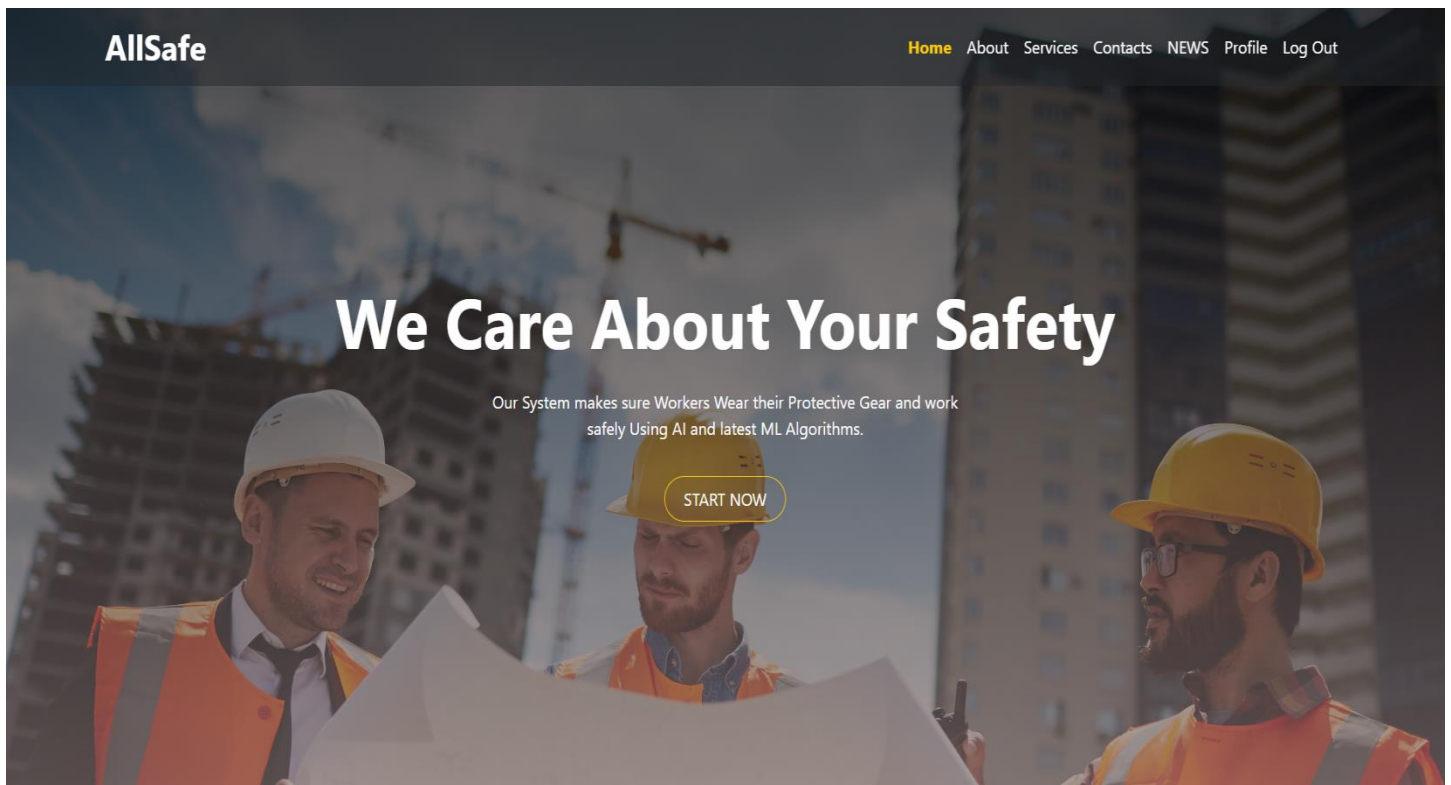
Advantages of using a website as User Interface:

- Websites are Simple & User Friendly.
- Websites do not depend on device hardware.
- Websites can be accessed through Android Devices as well as iPhone and PC



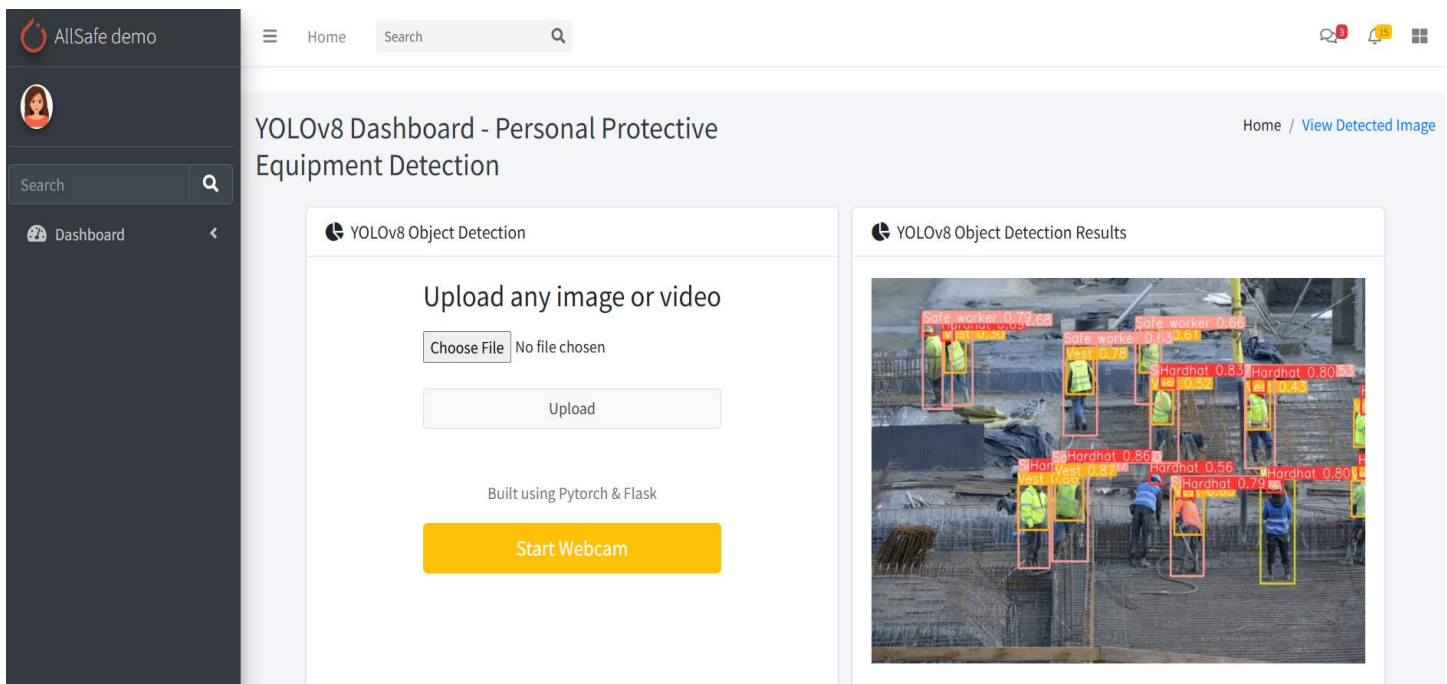
Part I: Login and Register

- This is the login & Register Page which is a requirement to access the website.
- It's implemented using Flask Login and Flask-WTF Forms



Part II: Home Page

- This is the welcome screen page that contains the application logo and a button (Start Now) To direct the user To Try the Safety Detection Model.
- The user can navigate the Home Page for more information and Details about the project and Usage of AI in Work Environments.
- It also provides Information about The Services we provide and the means to Contact the Company



Part III: AI Model Page

- This is the Safety Detection model page which contains a User Interface to facilitate the interaction between the user and the AI model.
- The user can easily upload any image or video, then the model will return the image or video with all highlighted Safety measurement to address safe and unsafe Workers
- The model can also run in real life using any Webcam

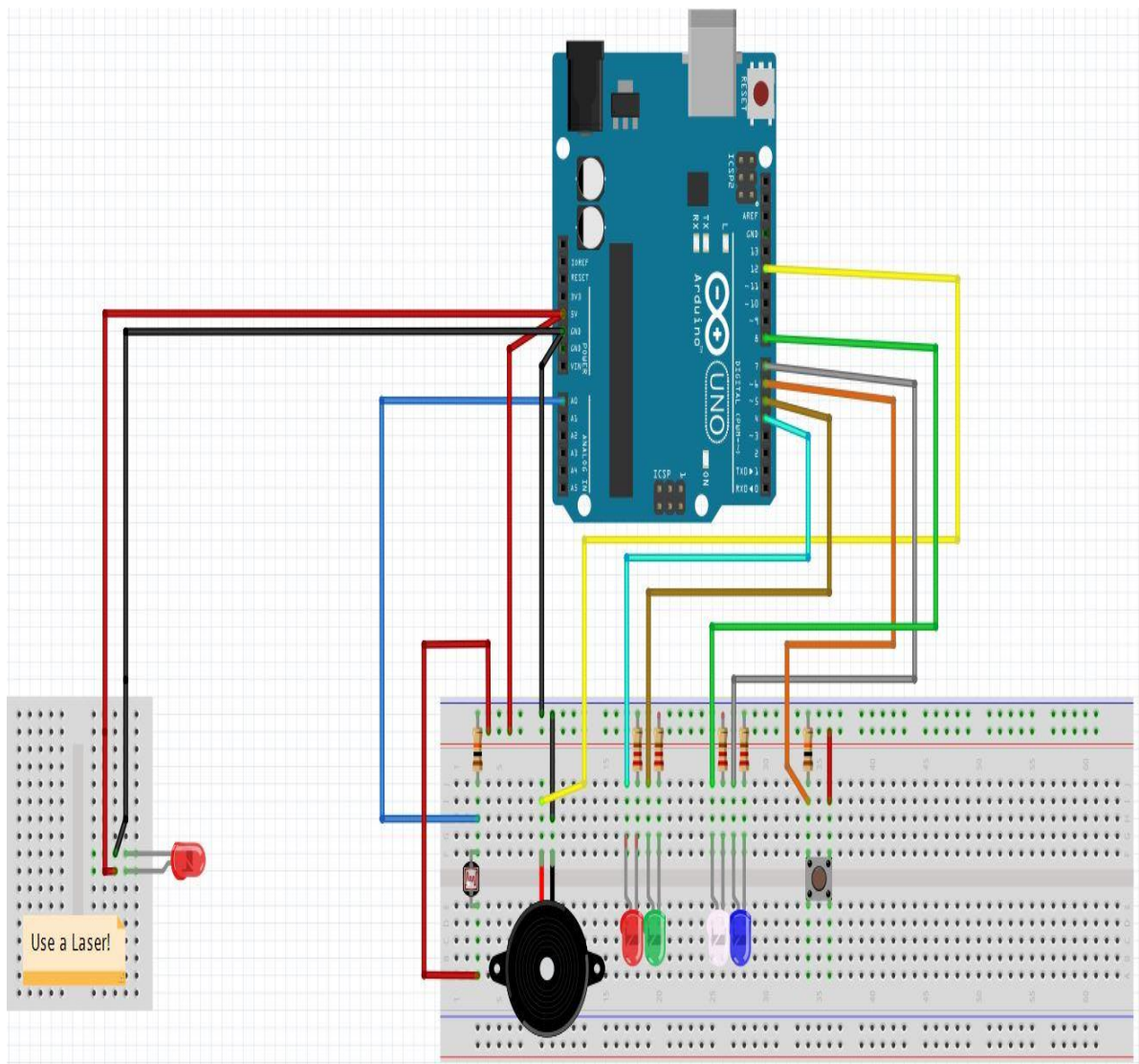
CHAPTER SIX: LASER SECURITY SYSTEM

6.1 Introduction:

Purpose: Explain the goal of the project, which might be a simple light-based detection system using an Arduino.

Overview: Briefly describe each component and its role in the circuit.

6.2 Architecture Diagram:



6.3 Components List:

Components and Their Functions

1. Arduino Uno

- **Function:** Acts as the brain of the project, controlling all other components based on programmed instructions.
- **Connections:** Provides digital and analog pins for inputs and outputs, and power to other components.

2. Breadboard

- **Function:** A platform for constructing the circuit without soldering.
- **Connections:** Used to connect components together through rows and columns of interconnected holes.

3. Laser Diode (or LED)

- **Function:** Used to emit a laser beam (or light) for detection by the photoresistor.
- **Connections:**
 - **Cathode (-):** Connected to GND (Ground).
 - **Anode (+):** Connected to the power rail (+5V) on the breadboard.

4. Photoresistor (LDR - Light Dependent Resistor)

- **Function:** Detects the presence or absence of light from the laser diode.
- **Connections:**
 - One side connected to GND.
 - The other side connected to an analog input pin on the Arduino (A0) and through a resistor to +5V. This creates a voltage divider.

5. Push Button

- **Function:** Allows user input, typically used to trigger events in the code.
- **Connections:**
 - One side connected to GND.
 - The other side connected to a digital input pin on the Arduino (D2) and through a pull-up resistor to +5V.

6. LEDs (Red, Green, Blue)

- **Function:** Indicate different states or outputs, such as alerts or status signals.
- **Connections:**
 - **Cathode (-):** Connected to GND.
 - **Anode (+):** Each connected through a resistor to specific digital pins on the Arduino.
 - Red LED: Digital Pin D11
 - Green LED: Digital Pin D10
 - Blue LED: Digital Pin D9

7. Buzzer

- **Function:** Produces sound for alerts or notifications.
- **Connections:**
 - One side connected to GND.
 - The other side connected to a digital output pin on the Arduino (D8).

8. Resistors

- **Function:** Limit current to the LEDs and other components to prevent damage.
- **Connections:** Placed in series with LEDs and the photoresistor.

Arduino Pin Connections

- **Analog Input:**

- A0: Connected to the junction of the photoresistor and the series resistor. It reads the voltage which changes based on the light intensity.

- **Digital Inputs/Outputs:**

- D2: Connected to the push button, reads the button state (HIGH when not pressed, LOW when pressed).
- D8: Connected to the buzzer, used to control sound.
- D9: Connected to the blue LED, used to control its state.
- D10: Connected to the green LED, used to control its state.
- D11: Connected to the red LED, used to control its state.

- **Power:**

- 5V: Supplies power to the components on the breadboard, such as the photoresistor circuit and the laser diode/LED.
- GND: Common ground connection for all components, ensuring a complete circuit.

Power and Ground Connections

- Power Rail (Red Line): Connected to the 5V pin on the Arduino.
- Ground Rail (Black Line): Connected to the GND pin on the Arduino.

6.4 Arduino Code :

- **Function:** Code that reads the photoresistor value, monitors the button state, and controls the LEDs and buzzer.
- **Example Code:** Provided below.

6.5 Testing and Troubleshooting :

- **Steps:** How to test each component, starting with individual components and then integrating them.
- **Common Issues:** Possible problems and their solutions (e.g., no light detected, LEDs not lighting up).

6.6 Sample Arduino Code :

```
Laser_Project | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Arduino Uno

Laser_Project.ino
1  const int triggeredLED = 7;
2  const int triggeredLED2 = 8;
3  const int RedLED = 4;
4  const int GreenLED = 5;
5  const int inputPin = A0;
6  const int speakerPin = 12;
7  const int armButton = 6;
8
9  boolean isArmed = true;
10 boolean isTriggered = false;
11 int buttonVal = 0;
12 int prev_buttonVal = 0;
13 int reading = 0;
14 int threshold = 0;
15
16
17 const int lowrange = 2000;
18 const int highrange = 4000;
19
20 void setup(){
21
22   pinMode(triggeredLED, OUTPUT);
23   pinMode(triggeredLED2, OUTPUT);
24   pinMode(RedLED, OUTPUT);
25   pinMode(GreenLED, OUTPUT);
26   pinMode(armButton, INPUT);
27   digitalWrite(triggeredLED, HIGH);
28   delay(500);
29   digitalWrite(triggeredLED, LOW);
30
31   calibrate();
32   setArmedState();
33 }
34
35 void loop(){
36
37   reading = analogRead(inputPin);
38
39
40
41   int buttonVal = digitalRead(armButton);
42   if ((buttonVal == HIGH) && (prev_buttonVal == LOW)){
43     setArmedState();
44     delay(500);
45   }
46
47   if ((isArmed) && (reading < threshold)){
```

```
Laser_Project | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Arduino Uno

Laser_Project.ino
46
47   if ((isArmed) && (reading < threshold)){
48     isTriggered = true;
49
50   if (isTriggered){
51
52     for (int i = lowrange; i <= highrange; i++)
53     {
54       tone (speakerPin, i, 250);
55     }
56
57     for (int i = highrange; i >= lowrange; i--)
58     {
59       tone (speakerPin, i, 250);
60     }
61
62     digitalWrite(triggeredLED, HIGH);
63     delay(50);
64     digitalWrite(triggeredLED, LOW);
65     delay (50);
66     digitalWrite(triggeredLED2, HIGH);
67     delay (50);
68     digitalWrite(triggeredLED2, LOW);
69     delay (50);
70   }
71   delay(20);
72 }
73
74 void setArmedState(){
75
76   if (isArmed){
77     digitalWrite(GreenLED, HIGH);
78     digitalWrite(RedLED, LOW);
79     isTriggered = false;
80     isArmed = false;
81   } else {
82     digitalWrite(GreenLED, LOW);
83     digitalWrite(RedLED, HIGH);
84     tone(speakerPin, 220, 125);
85     delay(200);
86     tone(speakerPin, 190, 250);
87     isArmed = true;
88   }
89 }
90
91 }
92
```




```
91 }
92
93 void calibrate(){
94
95   int sample = 0;
96   int baseline = 0;
97   const int min_diff = 200;
98   const int sensitivity = 50;
99   int success_count = 0;
100
101   digitalWrite(RedLED, LOW);
102   digitalWrite(GreenLED, LOW);
103
104   for (int i=0; i<10; i++){
105     sample += analogRead(inputPin);
106     digitalWrite(GreenLED, HIGH);
107     delay (50);
108     digitalWrite(GreenLED, LOW);
109     delay (50);
110   }
111
112   do
113   {
114     sample = analogRead(inputPin);
115
116     if (sample > baseline + min_diff){
117       success_count++;
118       threshold += sample;
119
120       digitalWrite(GreenLED, HIGH);
121       delay (100);
122       digitalWrite(GreenLED, LOW);
123       delay (100);
124     } else {
125       success_count = 0;
126       threshold = 0;
127     }
128   } while (success_count < 3);
129
130   threshold = (threshold/3) - sensitivity;
131
132   tone(speakerPin, 196, 250);
133   delay(200);
134   tone(speakerPin, 220, 125);
135 }
136
137
```

6.7 Explain The Previous Code:

Variables and Pin Assignments

- **Pins for LEDs and Speaker:**

const int triggeredLED = 7; // LED for triggered state

const int triggeredLED2 = 8; // Second LED for triggered state

const int RedLED = 4; // LED indicating disarmed state

const int GreenLED = 5; // LED indicating armed state

const int inputPin = A0; // Analog sensor input pin

const int speakerPin = 12; // Pin for the speaker

const int armButton = 6; // Pin for the arming button

- **Flags and Variables for System State :**

boolean isArmed = true; // Indicates if the system is armed

boolean isTriggered = false; // Indicates if the alarm is triggered

```
int buttonVal = 0;          // Current value of the arming button
int prev_buttonVal = 0;     // Previous value of the arming button
int reading = 0;           // Current sensor reading
int threshold = 0;         // Threshold for triggering the alarm
```

- **Frequency Range for Speaker :**

```
const int lowrange = 2000;  // Lower frequency limit for the speaker
const int highrange = 4000; // Upper frequency limit for the speaker
```

- **setup() Function :**

This function initializes the pins and performs initial calibration and setting of the armed state:

```
void setup(){
    pinMode(triggeredLED, OUTPUT);
    pinMode(triggeredLED2, OUTPUT);
    pinMode(RedLED, OUTPUT);
    pinMode(GreenLED, OUTPUT);
    pinMode(armButton, INPUT);

    // Briefly turn on the triggered LED to show the system is starting up
    digitalWrite(triggeredLED, HIGH);
    delay(500);
    digitalWrite(triggeredLED, LOW);

    calibrate();    // Calibrate the sensor
    setArmedState(); // Set the initial armed state
```

```
}
```

- **loop() Function :**

This function continuously checks the sensor reading and button state to determine the system's behavior:

```
void loop(){  
    reading = analogRead(inputPin); // Read the sensor value  
  
    // Check if the arming button is pressed  
    int buttonVal = digitalRead(armButton);  
    if ((buttonVal == HIGH) && (prev_buttonVal == LOW)){  
        setArmedState(); // Toggle armed state  
        delay(500);  
    }  
  
    // If the system is armed and the reading is below the threshold, trigger the alarm  
    if ((isArmed) && (reading < threshold)){  
        isTriggered = true;  
    }  
  
    // If the alarm is triggered, run the alarm routines  
    if (isTriggered){  
        // Make a sweeping sound with the speaker  
        for (int i = lowrange; i <= highrange; i++){  
            tone(speakerPin, i, 250);  
        }  
    }  
}
```

```
for (int i = highrange; i >= lowrange; i--){  
    tone(speakerPin, i, 250);  
}
```

```
// Blink the triggered LEDs  
digitalWrite(triggeredLED, HIGH);  
delay(50);  
digitalWrite(triggeredLED, LOW);  
delay (50);  
digitalWrite(triggeredLED2, HIGH);  
delay (50);  
digitalWrite(triggeredLED2, LOW);  
delay (50);  
}
```

```
delay(20); // Small delay to avoid button bounce issues  
}
```

- **setArmedState() Function :**

This function toggles the armed state of the system and updates the LED indicators accordingly:

```
void setArmedState(){  
    if (isArmed){  
        digitalWrite(GreenLED, HIGH); // Turn on the Green LED (armed)  
        digitalWrite(RedLED, LOW);    // Turn off the Red LED  
        isTriggered = false;          // Reset triggered state
```

```

    isArmed = false;          // Disarm the system
} else {
    digitalWrite(GreenLED, LOW); // Turn off the Green LED
    digitalWrite(RedLED, HIGH);  // Turn on the Red LED (disarmed)
    tone(speakerPin, 220, 125);  // Make a tone to indicate state change
    delay(200);
    tone(speakerPin, 196, 250);
    isArmed = true;            // Arm the system
}
}

```

- **calibrate() Function :**

This function determines the threshold value for triggering the alarm by sampling the sensor value:

```

void calibrate(){
    int sample = 0;
    int baseline = 0;
    const int min_diff = 200; // Minimum difference to consider a valid change
    const int sensitivity = 50; // Sensitivity adjustment
    int success_count = 0;

    digitalWrite(RedLED, LOW);
    digitalWrite(GreenLED, LOW);

    // Average of initial readings for baseline
    for (int i=0; i<10; i++){

```

```

sample += analogRead(inputPin);
digitalWrite(GreenLED, HIGH);
delay (50);
digitalWrite(GreenLED, LOW);
delay (50);
}

do {
    sample = analogRead(inputPin);

    if (sample > baseline + min_diff){
        success_count++;
        threshold += sample;

        digitalWrite(GreenLED, HIGH);
        delay (100);
        digitalWrite(GreenLED, LOW);
        delay (100);
    } else {
        success_count = 0;
        threshold = 0;
    }
} while (success_count < 3);

threshold = (threshold/3) - sensitivity;

```

```
tone(speakerPin, 196, 250);  
delay(200);  
tone(speakerPin, 220, 125);  
}
```

Detailed Explanation:

1. Initialization:

- **LEDs and speaker pins are set to output.** The LEDs will visually indicate states, while the speaker will make sounds.
- **triggeredLED is briefly turned on** during startup as a visual cue that the system is booting.

2. Calibration:

- **Reads initial sensor values** and averages them to get a baseline.
- **Waits for significant changes** (above min_diff) to determine a threshold.
- **Uses this threshold** to determine when the sensor value indicates a trigger event.

3. Main Loop:

- **Continuously reads sensor values.**
- **Checks the state of the arming button.** If pressed, toggles the armed state and waits 500ms to avoid bouncing.
- **If the system is armed** and the sensor reading is below the threshold, sets isTriggered to true.
- **If isTriggered is true**, the system:
 - **Makes a sweeping sound** using the speaker.
 - **Blinks triggeredLED and triggeredLED2** as a visual indicator.
- **State Management:**
 - **setArmedState toggles the armed/disarmed state.**
 - **LEDs and speaker tones** indicate the current state visually and audibly

7.8 Conclusion :

We have successfully built an AI model capable of detecting whether a person is wearing a helmet or a vest. If the person is wearing both, they are considered safe. If they are wearing only one or neither, it will notify us accordingly, with accuracy 95%.

We converted this model into an API, allowing us to integrate it with our website. Additionally, we developed a web application connected to the model via the API, enabling regular users to interact with the site and use the model more easily by uploading an image or video, or by operating it in real-time.

References

[Frist Paper](#)

[Second Paper](#)

[Third Paper](#)

[Fourth Paper](#)

[fifth Paper](#)

[YOLOV5 DETECTION](#)

[YOLO EXPLAIN](#)

[YOLOV7 DETECTION](#)

[YOLOV7 DETECTION](#)

[YOLOV8 DETECTION](#)