Project Report Format

INTRODUCTION

1.1 Project overview

In recent years, the advancements in computer vision and deep learning have paved the way for innovative solutions in the field of object detection and recognition. One such revolutionary approach is the You Only Look Once (YOLO) algorithm, a real-time object detection system that has gained significant attention for its efficiency and accuracy. As the demand for enhancing public safety and security continues to grow, there is an increasing need for robust and efficient systems to detect and prevent potential threats. One significant application of YOLO-based object detection is in the domain of weapon detection.

The utilization of YOLO for weapon detection represents a critical stride towards the development of intelligent and proactive security systems. Traditional methods of object detection often face challenges in real-time processing and accuracy, especially in scenarios where timely identification of weapons is crucial. YOLO addresses these challenges by providing a unified and comprehensive solution that enables simultaneous detection and classification of objects within a single frame.

1.2 Purpose

Enhancing Public Safety:

The primary purpose of this project is to contribute to the enhancement of public safety by developing an intelligent weapon detection system. By leveraging the real-time capabilities of the YOLO algorithm, the system aims to quickly identify and respond to potential threats, thereby reducing the risk of incidents in public spaces.

Proactive Threat Prevention:

The project seeks to address the critical need for proactive threat prevention in various environments, including public transportation, crowded events, and sensitive installations. By employing advanced object detection techniques, the system aims to identify weapons before they can be used, providing a proactive approach to security.

Efficient Security Infrastructure:

The implementation of a YOLO-based weapon detection system is intended to bolster existing security infrastructure. The efficient and rapid identification of weapons through real-time processing contributes to a more responsive and adaptive security framework.

Minimizing Response Time:

The project's purpose is to minimize response time in critical situations. By utilizing the YOLO algorithm, which processes the entire image in a single forward pass, the system can swiftly detect weapons, enabling security personnel to respond promptly and effectively.

Scalability and Versatility:

Another key purpose is to develop a scalable and versatile weapon detection system. The project aims to create a solution that can be easily integrated into different environments and adapted to various surveillance setups, ensuring its applicability across a range of scenarios.

Technological Advancement in Security:

The project aligns with the broader goal of advancing technology in the security domain. By employing cutting-edge deep learning techniques, it seeks to demonstrate the feasibility and effectiveness of using state-of-the-art algorithms for addressing contemporary security challenges.

User-Friendly Implementation:

The purpose includes the development of a user-friendly interface for the weapon detection system. This ensures that security personnel can easily interact with the system, interpret alerts, and make informed decisions in real-time, enhancing the overall usability of the solution.

Contribution to Research and Development:

The project contributes to the ongoing research and development efforts in the field of computer vision and deep learning. By exploring the application of YOLO to weapon detection, it adds valuable insights to the broader discourse on the use of advanced technologies for security purposes.

2. LITERATURE SURVEY

2.1 Existing problem

Dataset Limitations:

An inherent limitation lies in the variability of weapon appearances, encompassing diverse shapes, sizes, and orientations. Training a YOLO model to generalize effectively across this spectrum of variability poses a considerable challenge. Ensuring that the dataset used for training adequately represents the real-world scenarios and environments where the system will be deployed is crucial to overcoming this obstacle.

Environmental Adaptability:

Adaptability to changing environmental conditions is another pressing concern. The system must contend with fluctuations in lighting, weather, and other environmental factors that may impact its performance. Achieving accuracy under diverse conditions necessitates sophisticated training strategies and potentially the incorporation of additional sensors to enhance environmental perception.

Real-Time Processing Constraints:

While the YOLO algorithm excels in real-time object detection, the computational demands for processing video streams from multiple surveillance cameras in real-time can be formidable. Striking a balance between accuracy and computational efficiency is vital to ensure practical deployment and scalability.

Privacy and Ethical Considerations:

The deployment of weapon detection systems raises privacy concerns, as continuous monitoring in public spaces intersects with individual privacy rights. Striking a delicate balance between security imperatives and privacy rights requires a nuanced approach and the establishment of clear guidelines and regulations to mitigate potential ethical and legal issues.

Cost Implications:

The development and deployment of an effective YOLO-based weapon detection system are not without financial implications. The costs associated with hardware, software development, and ongoing maintenance may act as a barrier to widespread adoption, particularly in resource-constrained environments. Balancing the efficacy of the system with its economic feasibility is a crucial consideration.

Security Against Adversarial Threats:

A critical challenge lies in ensuring the security of the YOLO-based system against adversarial attacks. As with any deep learning model, the vulnerability to manipulation of input data for misleading the system requires ongoing research and development efforts to enhance robustness and resilience against potential threats.

2.2 References

[1] N. Kurek, L. A. Darzi and J. Maa, “A Worldwide perspective provides insights into why a US surgeon general annual report on firearm injuries is needed in America,” Current Trauma Reports, vol. 6, pp. 36–43, 2020.

[2] Y. Ren, C. Zhu and S. Xiao, “Small object detection in optical remote sensing images via modified faster R-CNN,” Applied Sciences, vol. 8, no. 5, pp. 813–818, 2018.

[3] R. Olmos, S. Tabik and F. Herrera, “Automatic handgun detection alarm in videos using deep learning,” Neurocomputing, vol. 275, no. 9, pp. 66–72, 2018.

[4] M. M. Ghazi, B. Yanikoglu and E. Aptoula, “Plant identification using deep neural networks via optimization of transfer learning parameters,” Neurocomputing, vol. 235, no. 7, pp. 228–235, 2017.

[5] X. Shu, Y. Cai, L. Yang, L. Zhang and J. Tang, “Computational face reader based on facial attribute estimation,” Neurocomputing, vol. 236, no. 10, pp. 153–163, 2017.

[6] W. Yu, K. Yang, H. Yao, X. Sun and P. Xu, “Exploiting the complementary strengths of multi-layer CNN features for image retrieval,” Neurocomputing, vol. 237, no. 2, pp. 235–241, 2017.

[7] R. K. Tiwari and G. K. Verma, “A computer vision based framework for visual gun detection using harris interest point detector,” Procedia Computer Science, vol. 54, pp. 703–712, 2015.

[8] R. Olmos, S. Tabik, A. Lamas, F. Pérez-Hernández and F. Herrera, “A binocular image fusion approach for minimizing false positives in handgun detection with deep learning,” Information Fusion, vol. 49, no. 2, pp. 271–280, 2019.

2.3 Problem statement definition

As technological advancements in computer vision and deep learning pave the way for innovative security solutions, the development of a YOLO-based weapon detection system is not without its inherent challenges. The primary problem lies in the need to create a robust and reliable system capable of accurately identifying weapons in real-time while mitigating the risks associated with false positives and false negatives. Achieving this delicate balance is essential for instilling confidence in the system's efficacy and practical deployment in diverse environments.

The variability in weapon appearances, encompassing different shapes, sizes, and orientations, poses a significant hurdle. Training a YOLO model to generalize effectively across this spectrum of variability demands a diverse and representative dataset, highlighting the need for comprehensive data collection and annotation strategies. The system's adaptability to changing environmental conditions, including lighting variations and weather fluctuations, introduces further complexity, necessitating sophisticated training approaches and potentially additional sensors for enhanced perception.

Real-time processing constraints represent a practical challenge, as the YOLO algorithm's computational demands may strain resources, impacting the system's scalability and practical deployment across multiple surveillance cameras. Privacy concerns emerge as a critical issue, prompting the need for a nuanced approach to balance security imperatives with individual privacy rights and ethical considerations. The financial implications of developing and maintaining an effective YOLO-based weapon detection system also present a challenge, requiring careful consideration of costs to ensure economic feasibility and widespread adoption.

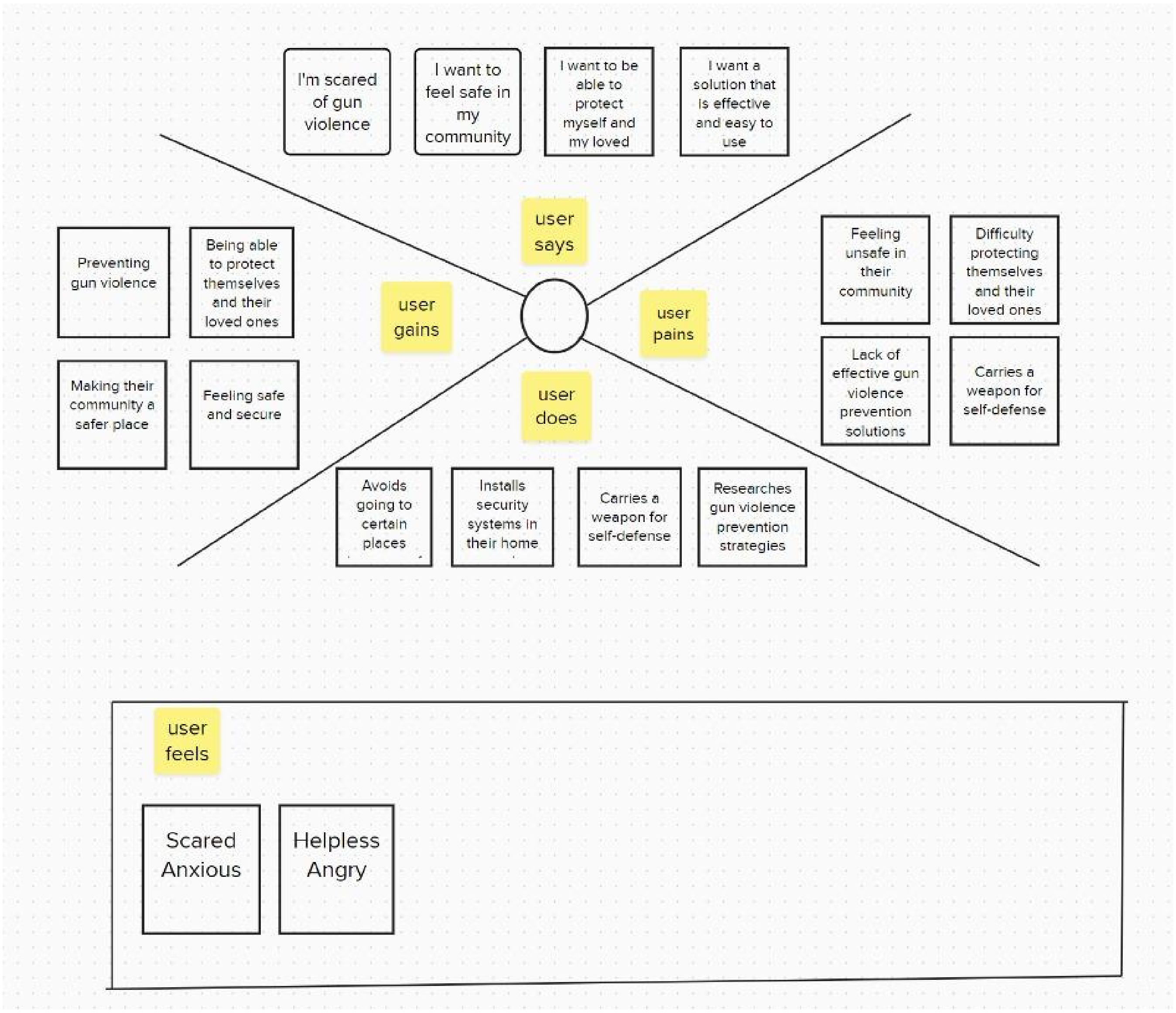
Addressing security concerns, including vulnerabilities to adversarial attacks and the nuanced understanding of human behaviors, adds further layers of complexity. Achieving a system that is resilient to manipulation and capable of distinguishing between normal and threatening behaviors is pivotal for its effectiveness and ethical deployment.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.
It is a useful tool to helps teams better understand their users.
Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

EMAPTHY MAP :

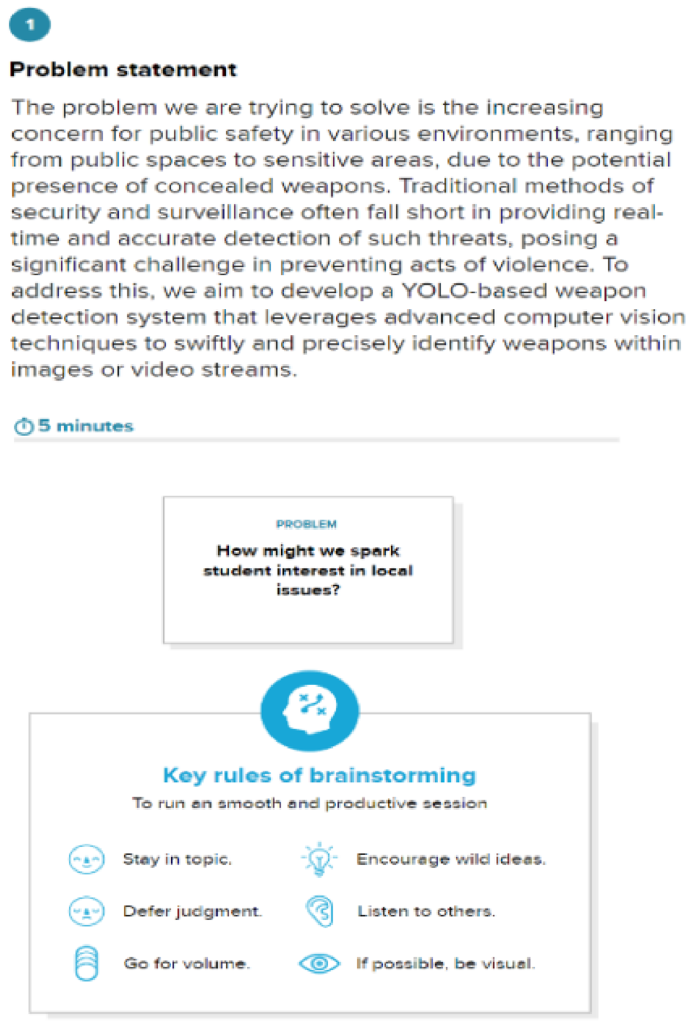


I'm scared of gun violence
I want to feel safe in my community
I want to be able to protect myself and my loved
I want a solution that is effective and easy to use
user says
user gains
user pains
user does
Preventing gun violence
Being able to protect themselves and their loved ones
Making their community a safer place
Feeling safe and secure
Feeling unsafe in their community
Difficulty protecting themselves and their loved ones
Lack of effective gun violence prevention solutions
Carries a weapon for self-defense
Avoids going to certain places
Installs security systems in their home
Carries a weapon for self-defense
Researches gun violence prevention strategies
user feels
Scared Anxious
Helpless Angry

3.2 Ideation & Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions. Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Step-1: Team Gathering, Collaboration and Select the Problem Statement



1

Problem statement

The problem we are trying to solve is the increasing concern for public safety in various environments, ranging from public spaces to sensitive areas, due to the potential presence of concealed weapons. Traditional methods of security and surveillance often fall short in providing real-time and accurate detection of such threats, posing a significant challenge in preventing acts of violence. To address this, we aim to develop a YOLO-based weapon detection system that leverages advanced computer vision techniques to swiftly and precisely identify weapons within images or video streams.

5 minutes

PROBLEM

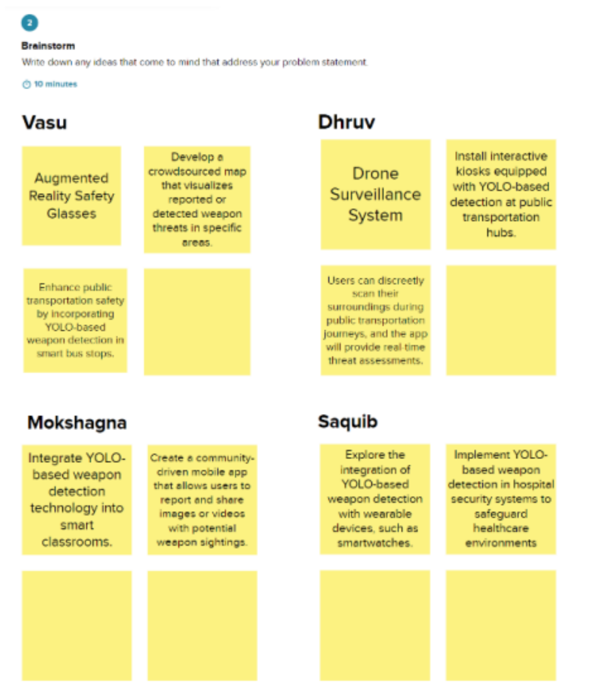
How might we spark student interest in local issues?

Key rules of brainstorming

To run an smooth and productive session

Stay in topic.	Encourage wild ideas.
Defer judgment.	Listen to others.
Go for volume.	If possible, be visual.

Step-2: Brainstorm, Idea Listing and Grouping



2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

Vasu

Augmented Reality Safety Glasses

Develop a crowdsourced map that visualizes reported or detected weapon threats in specific areas.

Enhance public transportation safety by incorporating YOLO-based weapon detection in smart bus stops.

Dhruv

Drone Surveillance System

Install interactive kiosks equipped with YOLO-based detection at public transportation hubs.

Users can discreetly scan their surroundings during public transportation journeys, and the app will provide real-time threat assessments.

Mokshagna

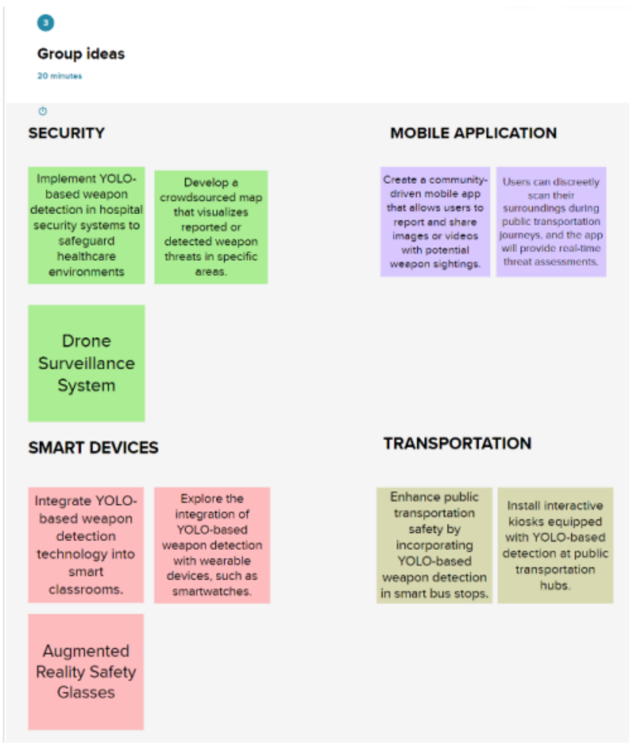
Integrate YOLO-based weapon detection technology into smart classrooms.

Create a community-driven mobile app that allows users to report and share images or videos with potential weapon sightings.

Saquib

Explore the integration of YOLO-based weapon detection with wearable devices, such as smartwatches.

Implement YOLO-based weapon detection in hospital security systems to safeguard healthcare environments



3

Group ideas

20 minutes

SECURITY

Implement YOLO-based weapon detection in hospital security systems to safeguard healthcare environments

Develop a crowdsourced map that visualizes reported or detected weapon threats in specific areas.

Drone Surveillance System

MOBILE APPLICATION

Create a community-driven mobile app that allows users to report and share images or videos with potential weapon sightings.

Users can discreetly scan their surroundings during public transportation journeys, and the app will provide real-time threat assessments.

SMART DEVICES

Integrate YOLO-based weapon detection technology into smart classrooms.

Explore the integration of YOLO-based weapon detection with wearable devices, such as smartwatches.

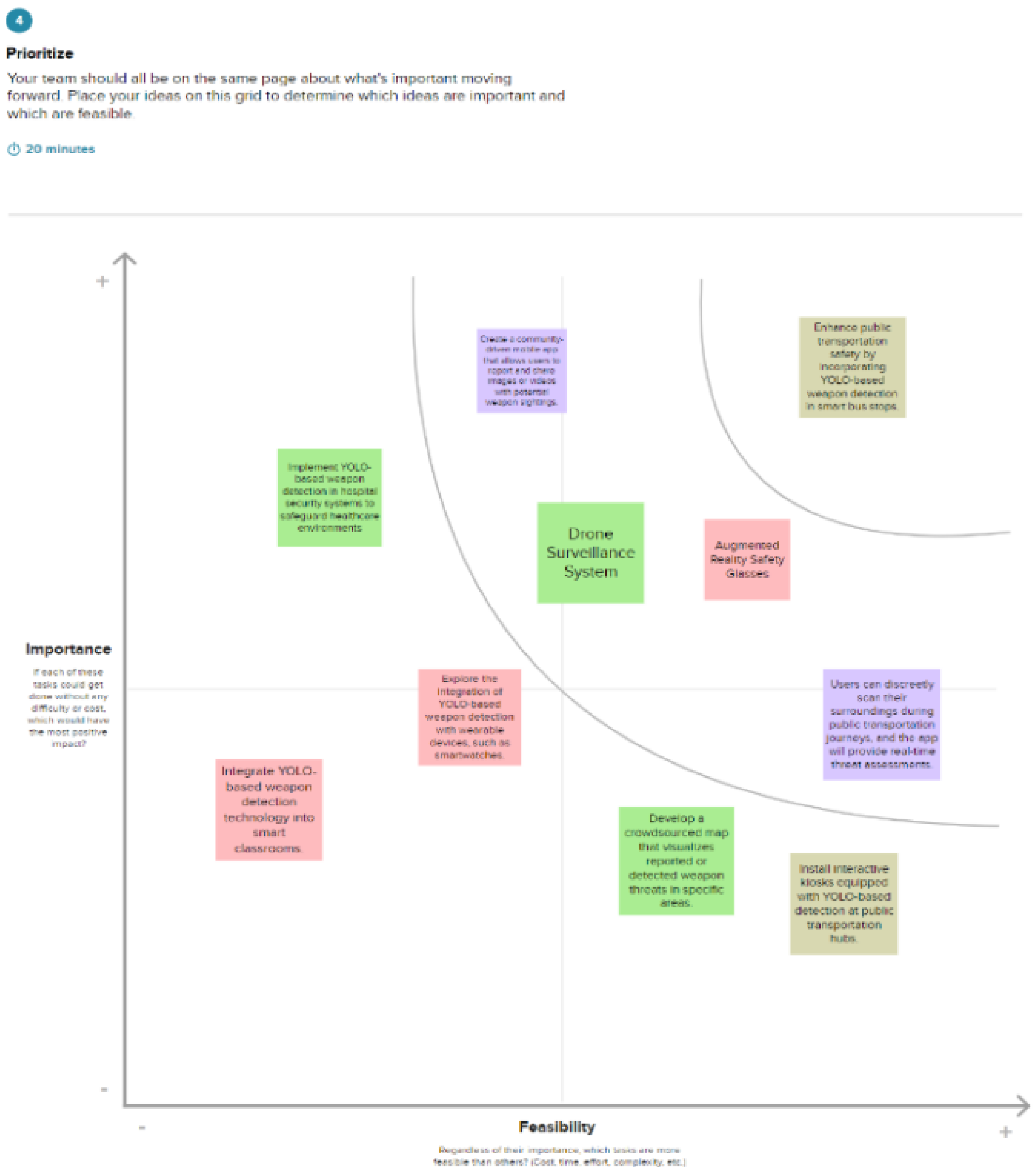
Augmented Reality Safety Glasses

TRANSPORTATION

Enhance public transportation safety by incorporating YOLO-based weapon detection in smart bus stops.

Install interactive kiosks equipped with YOLO-based detection at public transportation hubs.

Step-3: Idea Prioritization



Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

Create a community-driven mobile app that allows users to report and share images or videos with potential weapon sightings.

Enhance public transportation safety by incorporating YOLO-based weapon detection in smart bus stops.

Implement YOLO-based weapon detection in hospital security systems to safeguard healthcare environments

Drone Surveillance System

Augmented Reality Safety Glasses

Importance

If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?

Explore the integration of YOLO-based weapon detection with wearable devices, such as smartwatches.

Users can discreetly scan their surroundings during public transportation journeys, and the app will provide real-time threat assessments.

Integrate YOLO-based weapon detection technology into smart classrooms.

Develop a crowdsourced map that visualizes reported or detected weapon threats in specific areas.

Install interactive kiosks equipped with YOLO-based detection at public transportation hubs.

Feasibility

Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

Functional requirements describe the specific functionalities and features that the system must have to meet the needs of the users.

Real-time Weapon Detection:

Description: The system must be capable of detecting weapons in real-time within a given video feed.

Example: Weapons should be identified within a fraction of a second from the moment they appear in the video feed.

YOLO Integration:

Description: The solution must integrate the YOLO (You Only Look Once) algorithm for object detection, specifically optimized for weapon recognition.

Example: The system should utilize YOLOv4 for efficient and accurate detection of weapons.

Multiple Weapon Types:

Description: The system should be able to identify various types of weapons, including firearms, knives, and other potential threats.

Example: The solution must differentiate between handguns, rifles, and bladed weapons.

Alert Generation:

Description: Upon detecting a weapon, the system must generate immediate alerts, providing information on the location and type of the detected weapon.

Example: An alert should be sent to security personnel with details such as the timestamp, camera location, and a snapshot of the detected weapon.

Integration with Security Systems:

Description: The solution should seamlessly integrate with existing security systems, enabling coordinated responses with security personnel or law enforcement.

Example: The system should be compatible with standard security protocols and communication interfaces.

User Interface:

Description: The system should have a user-friendly interface for monitoring and managing the detection process.

Example: The user interface should include a live video feed display, a log of detected incidents, and an interactive map showing the locations of detected weapons.

4.2 Non-Functional requirements

Non-functional requirements define the operational characteristics of the system, such as performance, security, and usability.

Performance:

Description: The system must operate with low latency to ensure timely detection of weapons.

Example: The average detection time should be less than 200 milliseconds.

Accuracy:

Description: The solution must achieve a high level of accuracy in weapon detection to minimize false positives and negatives.

Example: The system should have an accuracy rate of 95% or higher.

Reliability:

Description: The system should be reliable, with minimal downtime, ensuring continuous monitoring and threat detection.

Example: The solution should have a mean time between failures (MTBF) of at least 500 hours.

Security:

Description: The solution must adhere to stringent security standards to prevent unauthorized access and tampering of data.

Example: All communication between system components should be encrypted using industry-standard protocols.

Usability:

Description: The user interface should be intuitive, requiring minimal training for security personnel to operate effectively.

Example: Security personnel should be able to perform basic operations (e.g., view live feed, acknowledge alerts) without specialized training.

Scalability:

Description: The solution should scale horizontally to accommodate increased processing demands as the number of video feeds or the complexity of the environment grows.

Example: The system should support up to 100 simultaneous video feeds without a significant decrease in performance.

Compatibility:

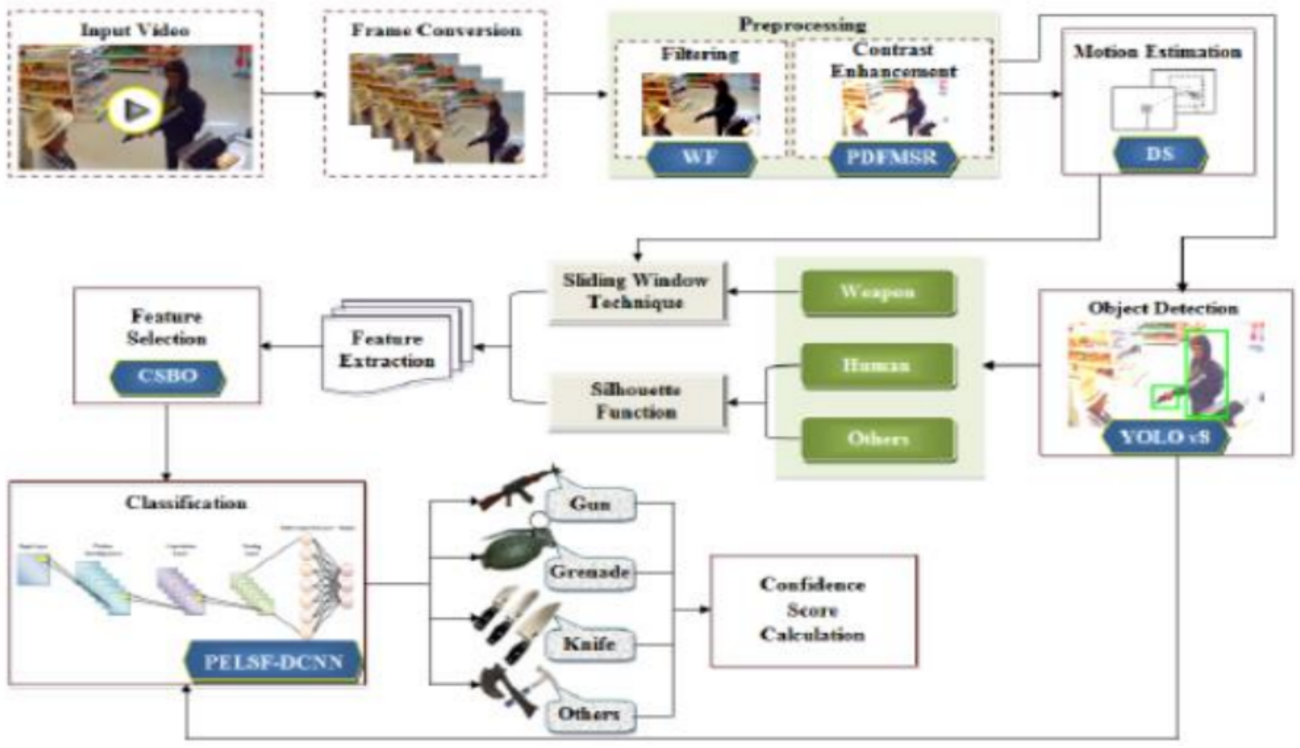
Description: The system should be compatible with a range of video input sources, ensuring flexibility in deployment.

Example: The solution should be able to process video feeds from various camera manufacturers and models.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

Proposed Model's Block Diagram :

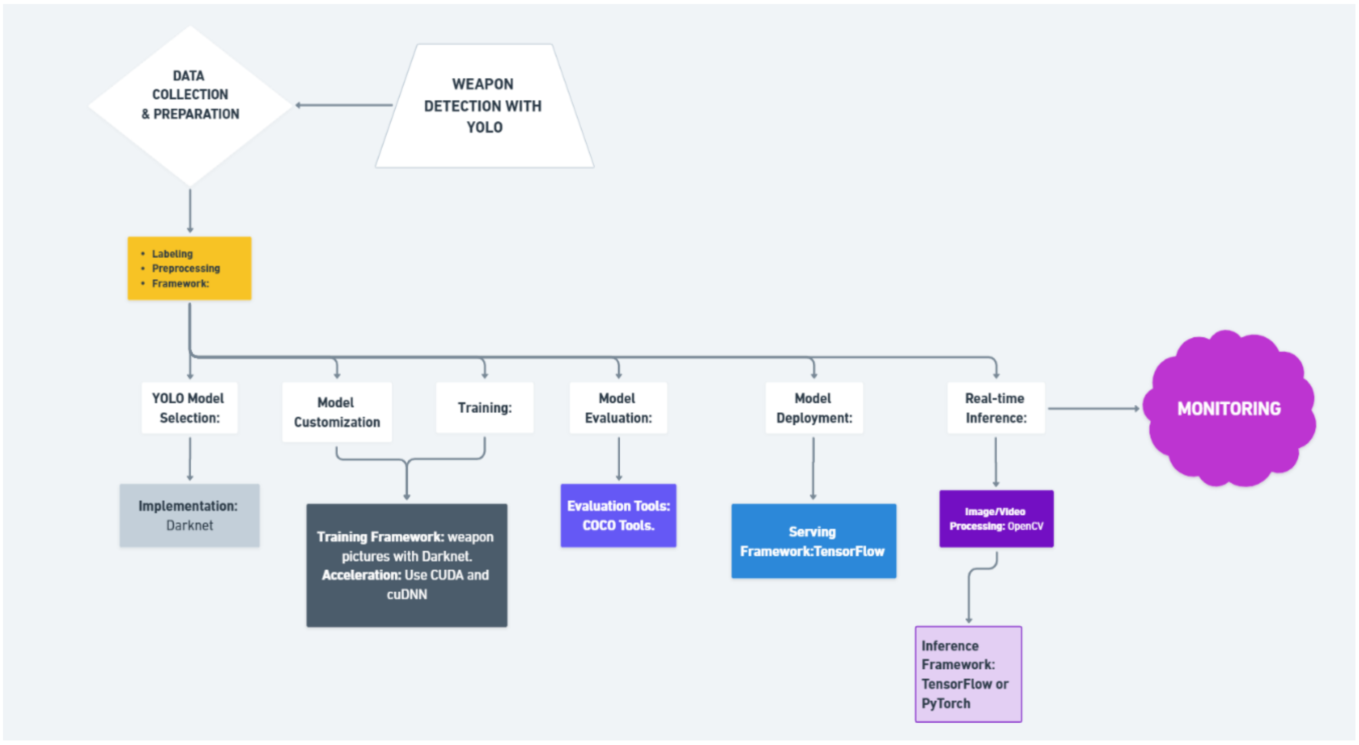


5.2 Solution Architecture

Solution Architecture :

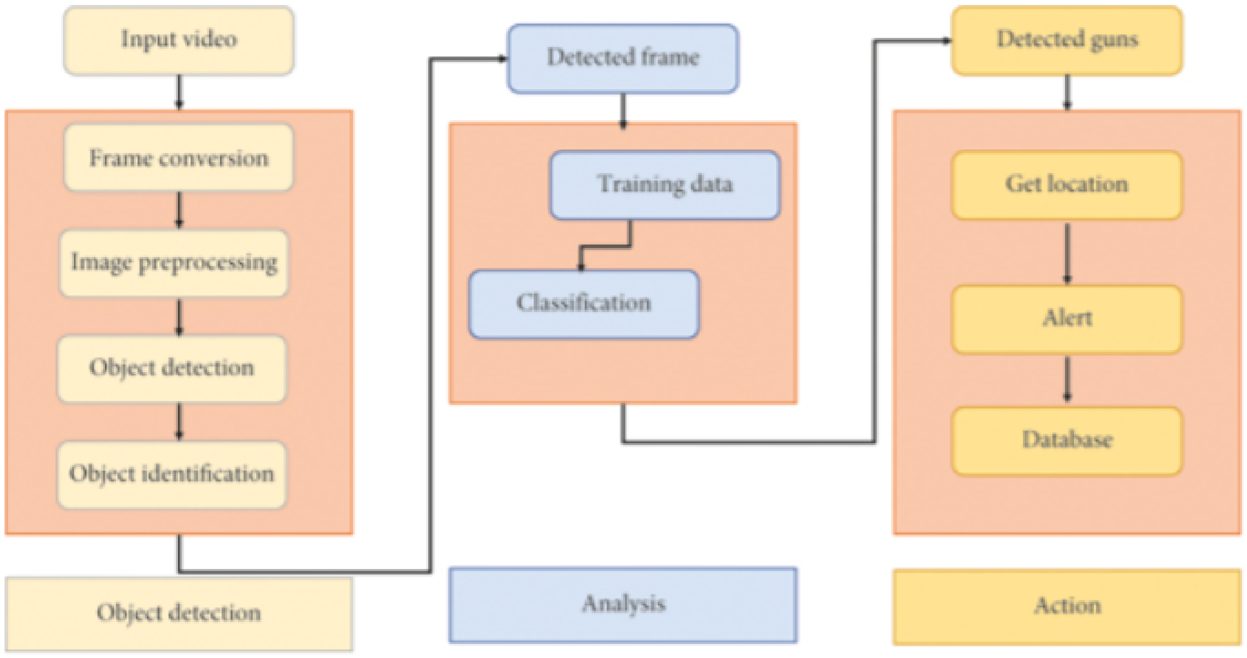
Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

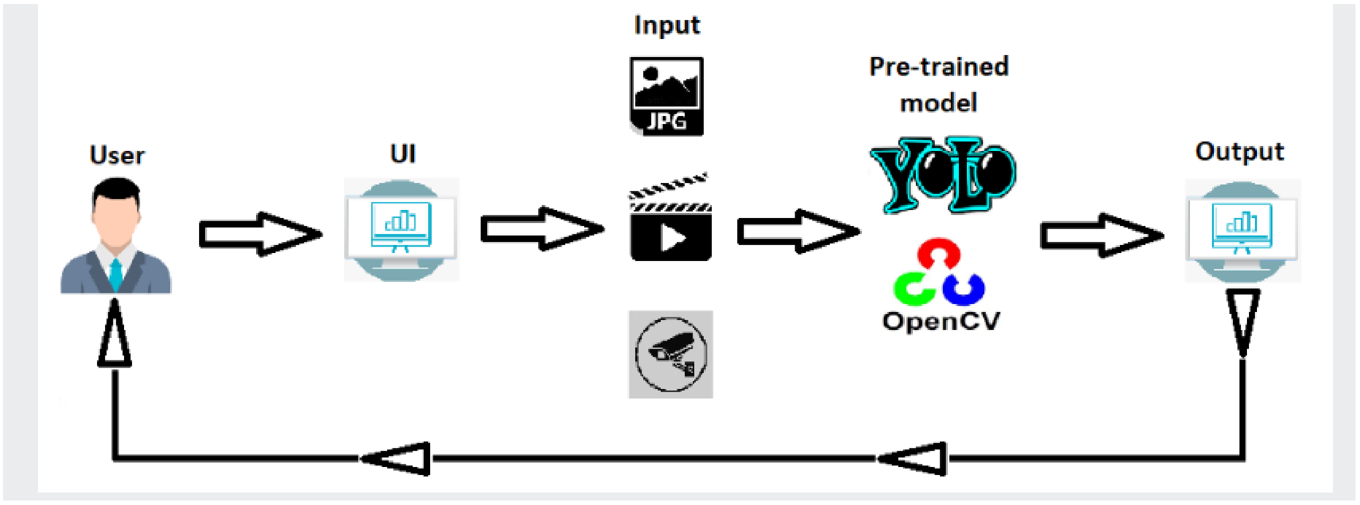
- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behavior , and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.


WEAPON DETECTION WITH YOLO
DATA COLLECTION & PREPARATION
Labeling
Preprocessing
Framework:
YOLO Model Selection:
Model Customization
Training:
Model Evaluation:
Model Deployment:
Real-time Inference:
MONITORING
Implementation: Darknet
Training Framework: weapon pictures with Darknet. Acceleration: Use CUDA and cuDNN
Evaluation Tools: COCO Tools.
Serving Framework:TensorFlow
Image/Video Processing: OpenCV
Inference Framework: TensorFlow or PyTorch

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture





6.2 Sprint Planning & Estimation

Sprint Planning:
Sprint Planning for our Weapon Detection project using YOLOv8 is a crucial step in achieving our goals efficiently.

Objective: Define sprint goals focused on refining the YOLOv8 model, implementing new features, and optimizing detection accuracy.

Key Steps:

Review Backlog: Prioritize tasks, with a focus on model enhancement, real-time detection improvements, and integration refinements.

Sprint Goal: Clearly define the sprint goal, such as achieving a specified increase in detection accuracy or optimizing processing speed.

Task Breakdown: Decompose user stories into tasks, including model training, algorithm optimization, and UI enhancements.

Capacity Planning: Allocate tasks based on team members' expertise, emphasizing YOLOv8 model proficiency.

Dependencies: Identify and address dependencies, ensuring smooth collaboration between development and testing.

Sprint Estimation:
Sprint Estimation ensures a realistic approach to implementing improvements in our YOLOv8-based Weapon Detection system.

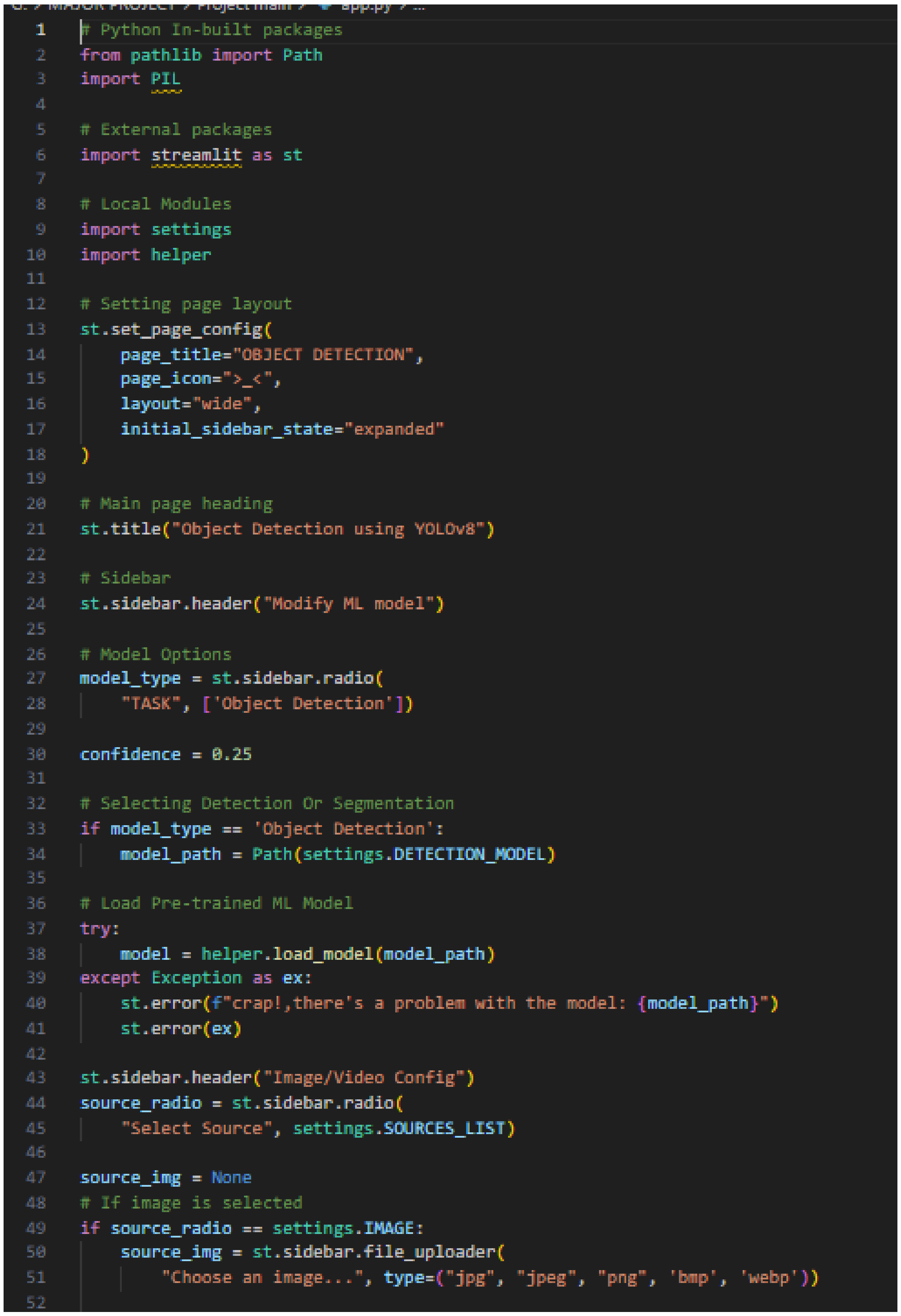
Story Points:
Use story points to estimate effort, considering factors like model training, integration, and testing.
Relative Sizing:
Employ techniques like Planning Poker to comparatively size tasks, ensuring a balanced workload.
Velocity:
Calculate velocity based on previous sprints, providing insights into achievable task completion rates.
Estimation Meeting:
Conduct collaborative estimation meetings to gather input on effort estimates for YOLOv8 model enhancements.

Buffer Time:
Include buffer time to accommodate unexpected challenges during model refinement and integration.
Documentation:
Document estimates for each task, facilitating transparent communication and progress tracking.

6.3 Sprint Delivery Schedule

The Sprint Delivery Schedule outlines the timeline for the refinement and delivery of our YOLOv8-based Weapon Detection project.
Start and End Dates:
Communicate sprint start and end dates, ensuring all team members are aligned with the sprint timeline.
Daily Standups:
Hold daily standup meetings to discuss progress, challenges, and ensure continuous collaboration.
Mid-Sprint Review:
Conduct a mid-sprint review to assess progress, focusing on YOLOv8 model enhancements and early integration results.
Delivery Date:
Set a specific delivery date for the completion of YOLOv8 model improvements and integration refinements.
Retrospective:
Schedule a retrospective meeting at the end of the sprint to reflect on the effectiveness of YOLOv8 model enhancements and identify areas for improvement.
Continuous Communication:
Emphasize ongoing communication to promptly address any issues and maintain alignment with our objective of enhancing weapon detection using YOLOv8.
By efficiently planning and estimating tasks and adhering to a clear delivery schedule, we aim to continuously improve the accuracy and efficiency of our YOLOv8-based Weapon Detection system.

7. CODING & SOLUTIONING



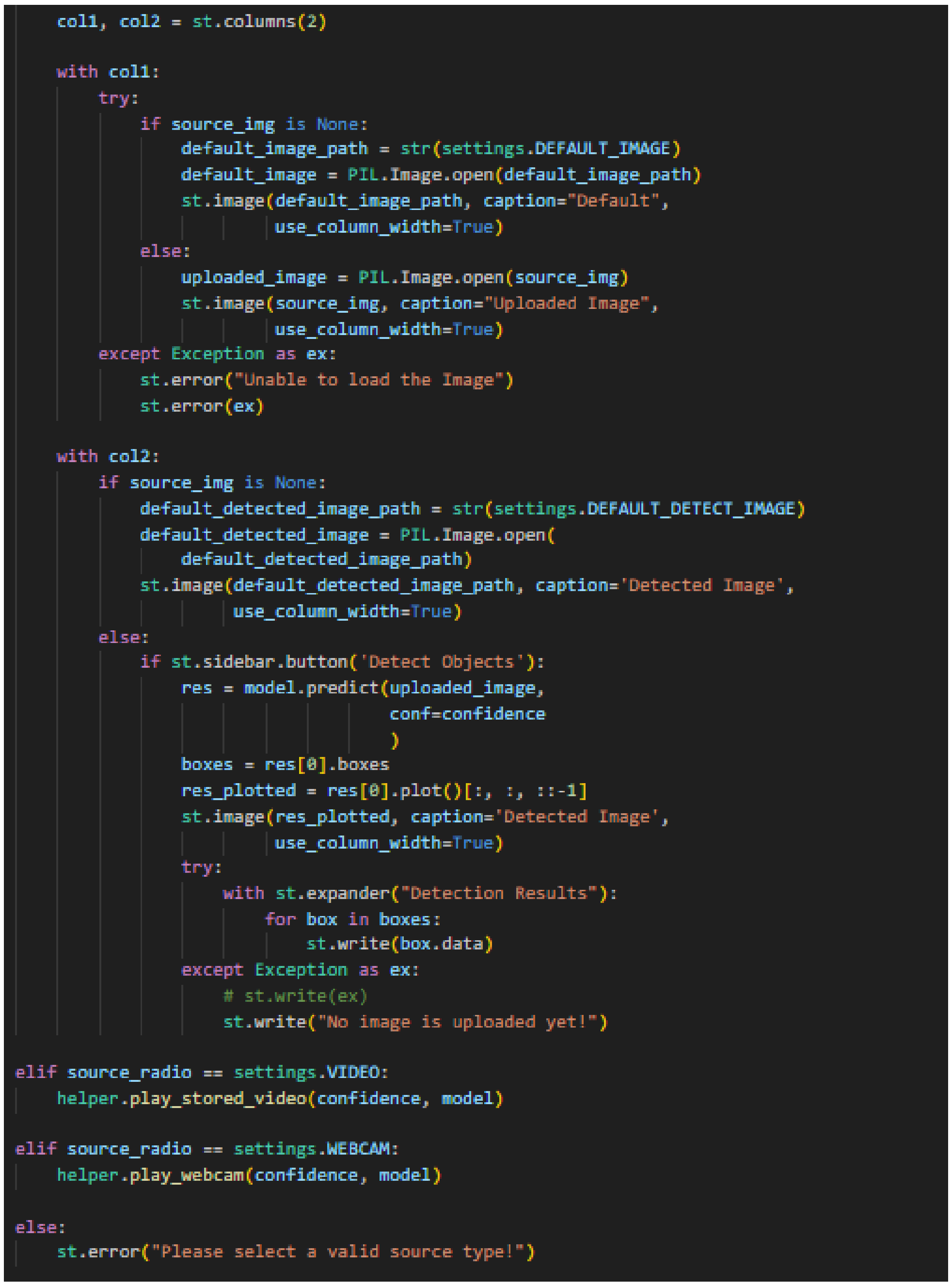
```python
# Python In-built packages
from pathlib import Path
import PIL

# External packages
import streamlit as st

# Local Modules
import settings
import helper

# Setting page layout
st.set_page_config(
    page_title="OBJECT DETECTION",
    page_icon=">_<",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Main page heading
st.title("Object Detection using YOLOv8")

# Sidebar
st.sidebar.header("Modify ML model")

# Model Options
model_type = st.sidebar.radio(
    "TASK", ['Object Detection'])

confidence = 0.25

# Selecting Detection Or Segmentation
if model_type == 'Object Detection':
    model_path = Path(settings.DETECTION_MODEL)

# Load Pre-trained ML Model
try:
    model = helper.load_model(model_path)
except Exception as ex:
    st.error(f"crap!,there's a problem with the model: {model_path}")
    st.error(ex)

st.sidebar.header("Image/Video Config")
source_radio = st.sidebar.radio(
    "Select Source", settings.SOURCES_LIST)

source_img = None
# If image is selected
if source_radio == settings.IMAGE:
    source_img = st.sidebar.file_uploader(
        "Choose an image...", type=("jpg", "jpeg", "png", 'bmp', 'webp'))

```

Imports:Imports necessary modules: pathlib, PIL, streamlit, settings, and helper.

Streamlit Configuration:Configures the Streamlit app's page title, icon, layout, and initial sidebar state.

Sidebar for Image/Video Configuration:Adds a sidebar header for image/video configuration.

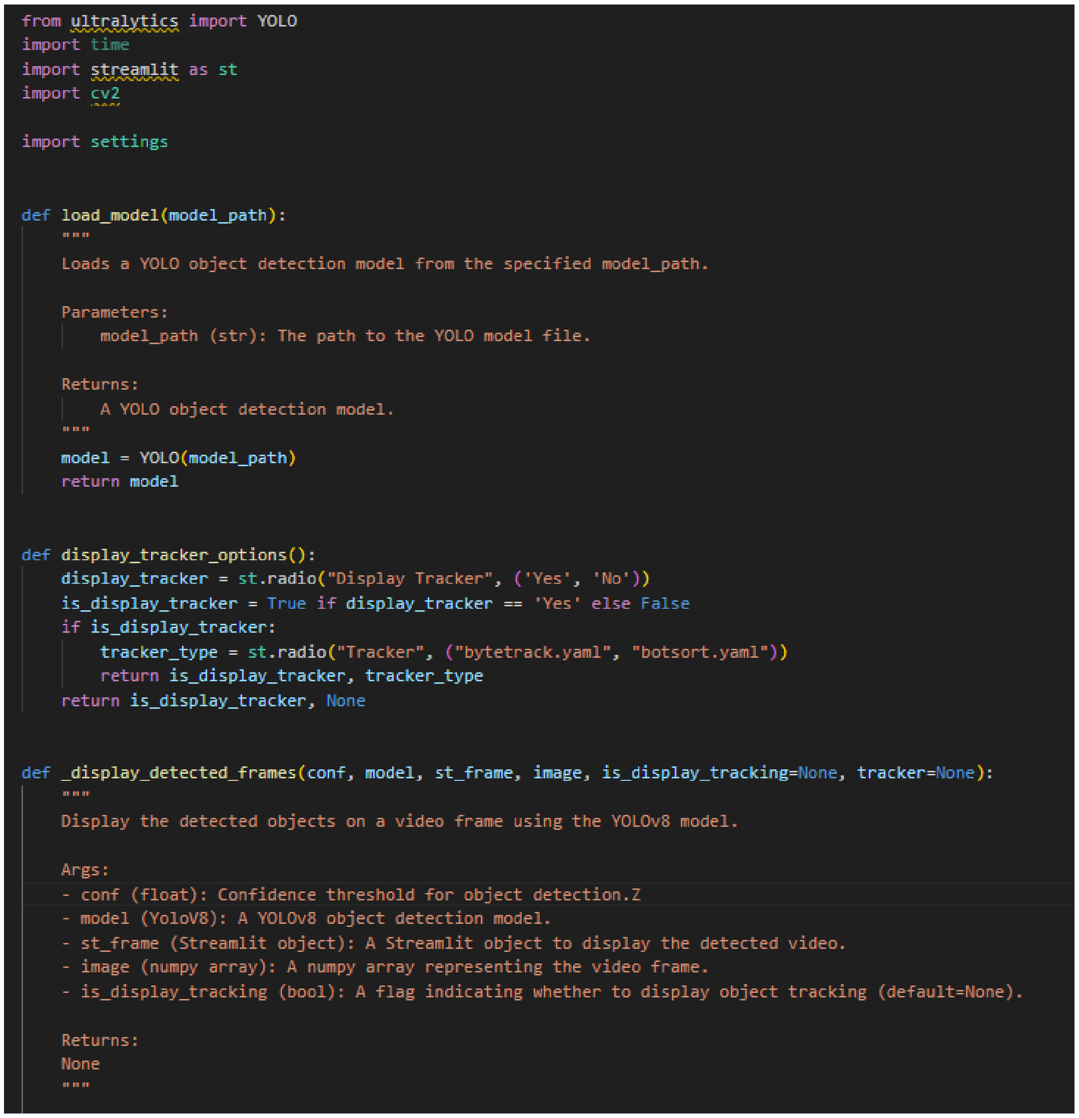
Select Source:Allows users to choose the source (image or video) from available options.
File Uploader for Images:If the source is an image, provides a file uploader to choose an image.



```python
    col1, col2 = st.columns(2)

    with col1:
        try:
            if source_img is None:
                default_image_path = str(settings.DEFAULT_IMAGE)
                default_image = PIL.Image.open(default_image_path)
                st.image(default_image_path, caption="Default",
                         use_column_width=True)
            else:
                uploaded_image = PIL.Image.open(source_img)
                st.image(source_img, caption="Uploaded Image",
                         use_column_width=True)
        except Exception as ex:
            st.error("Unable to load the Image")
            st.error(ex)

    with col2:
        if source_img is None:
            default_detected_image_path = str(settings.DEFAULT_DETECT_IMAGE)
            default_detected_image = PIL.Image.open(
                default_detected_image_path)
            st.image(default_detected_image_path, caption='Detected Image',
                     use_column_width=True)
        else:
            if st.sidebar.button('Detect Objects'):
                res = model.predict(uploaded_image,
                                    conf=confidence
                                    )
                boxes = res[0].boxes
                res_plotted = res[0].plot()[:, :, ::-1]
                st.image(res_plotted, caption='Detected Image',
                         use_column_width=True)
                try:
                    with st.expander("Detection Results"):
                        for box in boxes:
                            st.write(box.data)
                except Exception as ex:
                    # st.write(ex)
                    st.write("No image is uploaded yet!")

elif source_radio == settings.VIDEO:
    helper.play_stored_video(confidence, model)

elif source_radio == settings.WEBCAM:
    helper.play_webcam(confidence, model)

else:
    st.error("Please select a valid source type!")
```

Source Types:Adjusts the content based on the selected source type (image, video, or webcam).Utilizes helper functions for video and webcam sources.

Error Handling:Displays an error message if an invalid source type is selected. This script provides a concise user interface for image and video object det

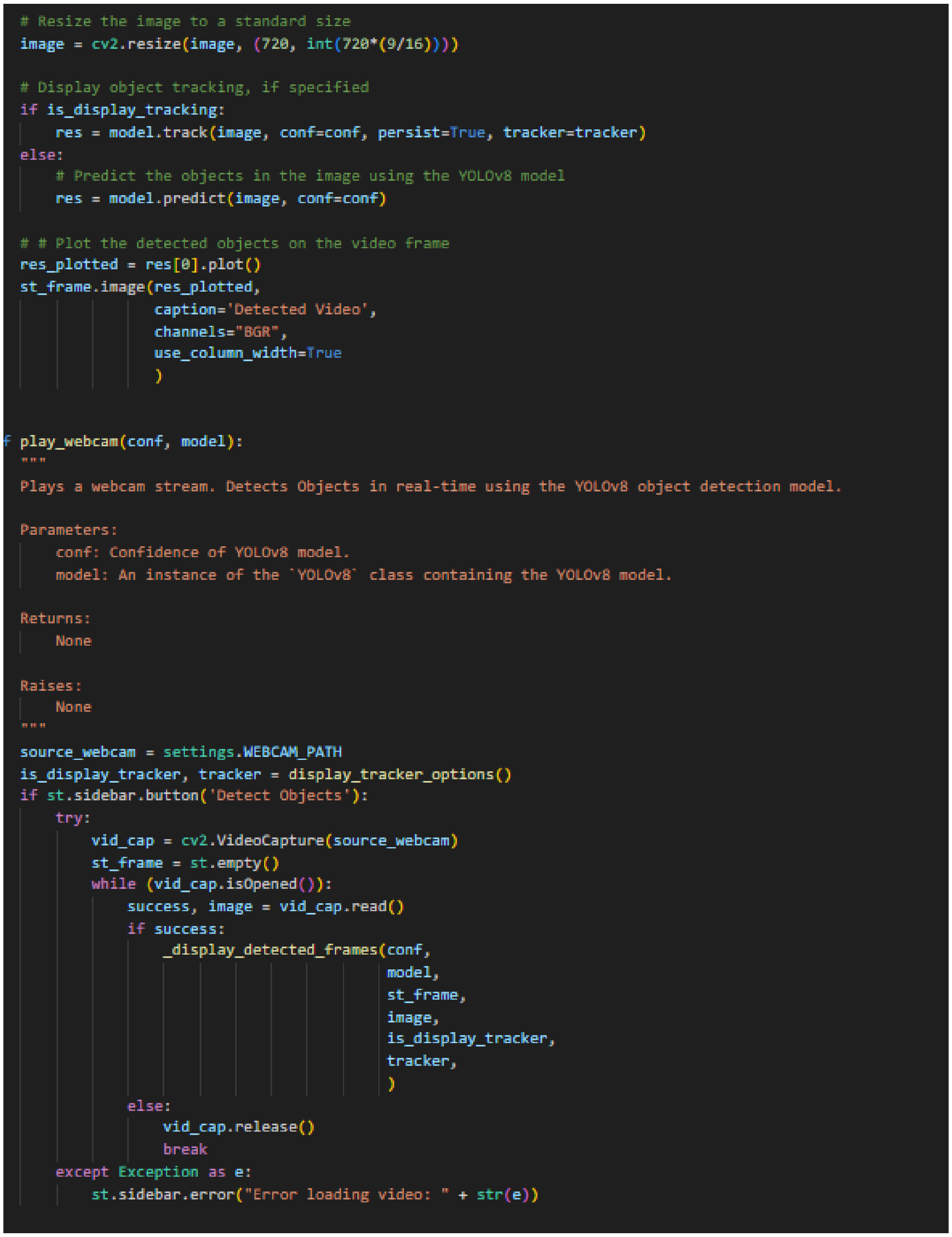


```python
from ultralytics import YOLO
import time
import streamlit as st
import cv2

import settings


def load_model(model_path):
    """
    Loads a YOLO object detection model from the specified model_path.

    Parameters:
        model_path (str): The path to the YOLO model file.

    Returns:
        A YOLO object detection model.
    """
    model = YOLO(model_path)
    return model


def display_tracker_options():
    display_tracker = st.radio("Display Tracker", ('Yes', 'No'))
    is_display_tracker = True if display_tracker == 'Yes' else False
    if is_display_tracker:
        tracker_type = st.radio("Tracker", ("bytetrack.yaml", "botsort.yaml"))
        return is_display_tracker, tracker_type
    return is_display_tracker, None


def _display_detected_frames(conf, model, st_frame, image, is_display_tracking=None, tracker=None):
    """
    Display the detected objects on a video frame using the YOLOv8 model.

    Args:
    - conf (float): Confidence threshold for object detection.Z
    - model (YoloV8): A YOLOv8 object detection model.
    - st_frame (Streamlit object): A Streamlit object to display the detected video.
    - image (numpy array): A numpy array representing the video frame.
    - is_display_tracking (bool): A flag indicating whether to display object tracking (default=None).

    Returns:
    None
    """
```

Load YOLO Model:The load_model function loads a YOLO object detection model from the specified file path using Ultralytics.

Display Tracker Options:The display_tracker_options function uses Streamlit to present radio buttons for choosing whether to display object tracking and, if selected, which tracker type to use.
Display Detected Frames:

The _display_detected_frames function showcases detected objects on a video frame using the YOLOv8 model.



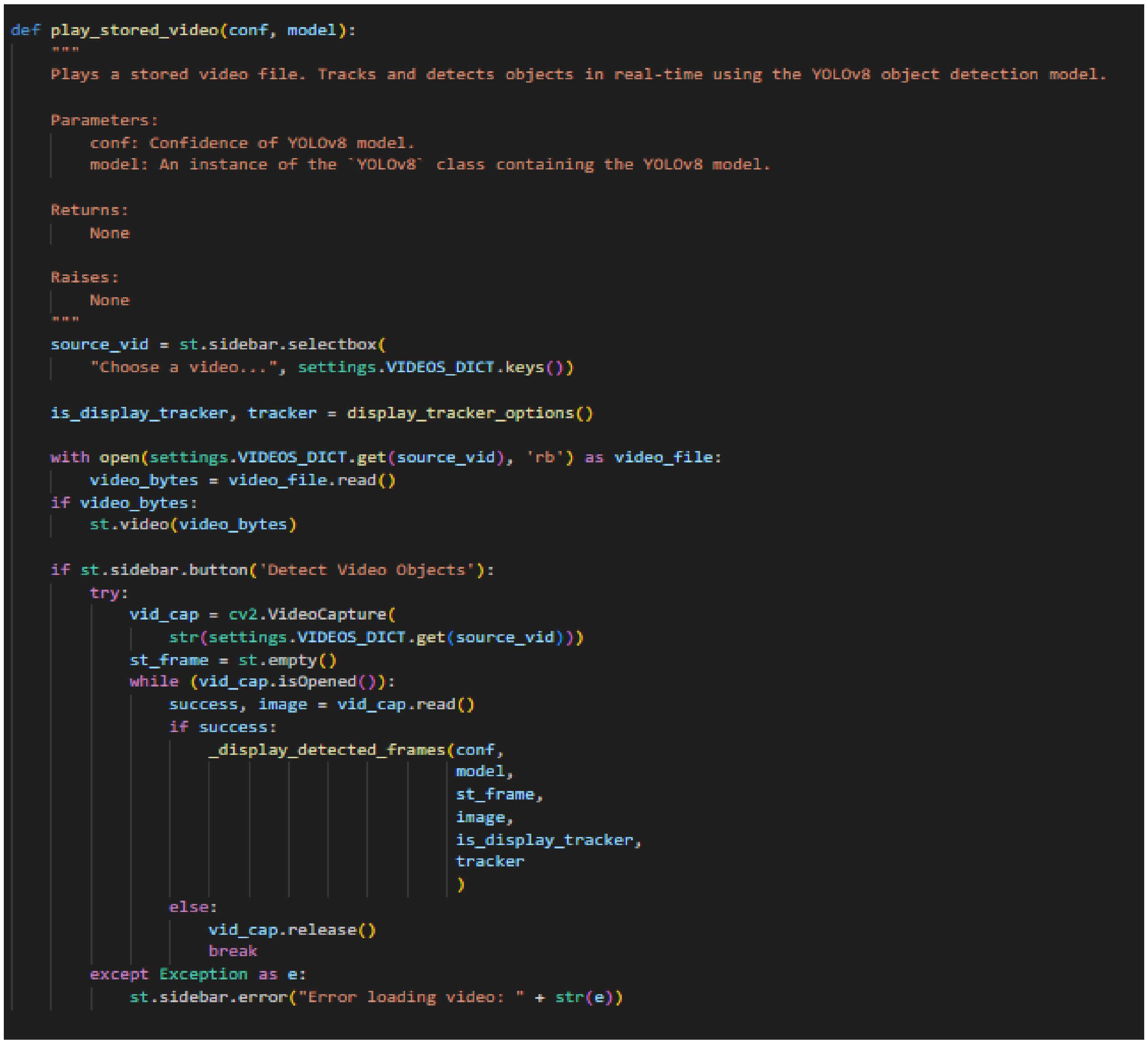
```
    # Resize the image to a standard size
    image = cv2.resize(image, (720, int(720*(9/16))))

    # Display object tracking, if specified
    if is_display_tracking:
        res = model.track(image, conf=conf, persist=True, tracker=tracker)
    else:
        # Predict the objects in the image using the YOLOv8 model
        res = model.predict(image, conf=conf)

    # # Plot the detected objects on the video frame
    res_plotted = res[0].plot()
    st_frame.image(res_plotted,
                   caption='Detected Video',
                   channels="BGR",
                   use_column_width=True
                   )


f play_webcam(conf, model):
    """
    Plays a webcam stream. Detects Objects in real-time using the YOLOv8 object detection model.

    Parameters:
        conf: Confidence of YOLOv8 model.
        model: An instance of the `YOLOv8` class containing the YOLOv8 model.

    Returns:
        None

    Raises:
        None
    """
    source_webcam = settings.WEBCAM_PATH
    is_display_tracker, tracker = display_tracker_options()
    if st.sidebar.button('Detect Objects'):
        try:
            vid_cap = cv2.VideoCapture(source_webcam)
            st_frame = st.empty()
            while (vid_cap.isOpened()):
                success, image = vid_cap.read()
                if success:
                    _display_detected_frames(conf,
                                             model,
                                             st_frame,
                                             image,
                                             is_display_tracker,
                                             tracker,
                                             )
                else:
                    vid_cap.release()
                    break
        except Exception as e:
            st.sidebar.error("Error loading video: " + str(e))
```

Resize Image:The script resizes the input image to a standard size (720 pixels width, maintaining the aspect ratio).

Object Detection and Tracking:If object tracking is specified (is_display_tracking is True), it uses the specified tracker (tracker) to track objects in the video frames using the model.track method.
If object tracking is not specified, it uses the model.predict method to predict objects in the image.

Display Detected Objects:The detected objects are plotted on the video frame using the plot method from the YOLOv8 model.

Webcam Stream Processing:The play_webcam function captures the webcam stream, displays options for object tracking, and, when the "Detect Objects" button is pressed, continuously reads frames from the webcam.



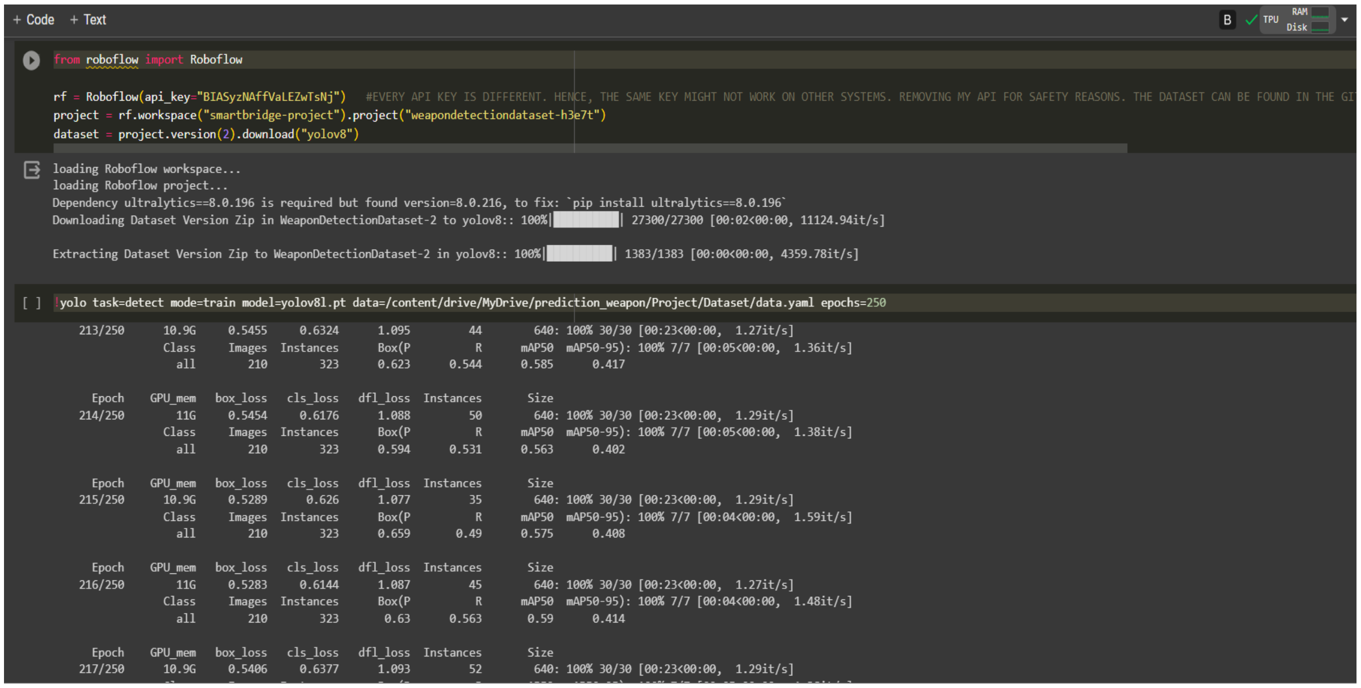
```python
def play_stored_video(conf, model):
    """
    Plays a stored video file. Tracks and detects objects in real-time using the YOLOv8 object detection model.

    Parameters:
        conf: Confidence of YOLOv8 model.
        model: An instance of the `YOLOv8` class containing the YOLOv8 model.

    Returns:
        None

    Raises:
        None
    """
    source_vid = st.sidebar.selectbox(
        "Choose a video...", settings.VIDEOS_DICT.keys())

    is_display_tracker, tracker = display_tracker_options()

    with open(settings.VIDEOS_DICT.get(source_vid), 'rb') as video_file:
        video_bytes = video_file.read()
    if video_bytes:
        st.video(video_bytes)

    if st.sidebar.button('Detect Video Objects'):
        try:
            vid_cap = cv2.VideoCapture(
                str(settings.VIDEOS_DICT.get(source_vid)))
            st_frame = st.empty()
            while (vid_cap.isOpened()):
                success, image = vid_cap.read()
                if success:
                    _display_detected_frames(conf,
                                             model,
                                             st_frame,
                                             image,
                                             is_display_tracker,
                                             tracker
                                             )
                else:
                    vid_cap.release()
                    break
        except Exception as e:
            st.sidebar.error("Error loading video: " + str(e))
```

Video Selection:The user can choose a stored video file from a sidebar select box, where available videos are listed based on the settings.VIDEOS_DICT.

Display Selected Video:The selected video is displayed using st.video in the Streamlit app.

Object Detection on Stored Video:When the "Detect Video Objects" button is pressed, the script reads frames from the selected video file (source_vid). Each frame is processed using the _display_detected_frames function, which performs object detection using the YOLOv8 model.

9. RESULTS

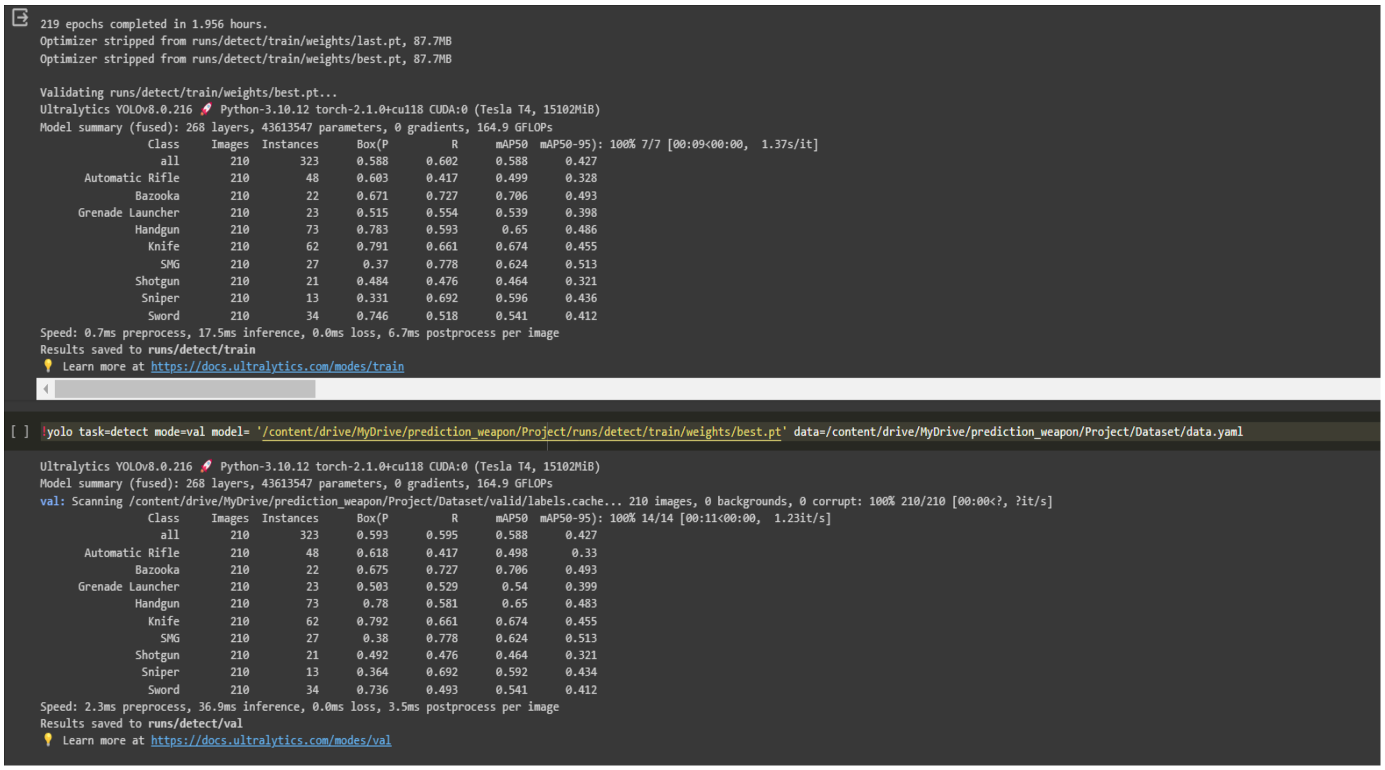


```
from roboflow import Roboflow

rf = Roboflow(api_key="BIASyzNAffVaLEZwTsNj")   #EVERY API KEY IS DIFFERENT. HENCE, THE SAME KEY MIGHT NOT WORK ON OTHER SYSTEMS. REMOVING MY API FOR SAFETY REASONS. THE DATASET CAN BE FOUND IN THE GI
project = rf.workspace("smartbridge-project").project("weapondetectiondataset-h3e7t")
dataset = project.version(2).download("yolov8")
```

```
loading Roboflow workspace...
loading Roboflow project...
Dependency ultralytics==8.0.196 is required but found version=8.0.216, to fix: `pip install ultralytics==8.0.196`
Downloading Dataset Version Zip in WeaponDetectionDataset-2 to yolov8:: 100%|██████████| 27300/27300 [00:02<00:00, 11124.94it/s]

Extracting Dataset Version Zip to WeaponDetectionDataset-2 in yolov8:: 100%|██████████| 1383/1383 [00:00<00:00, 4359.78it/s]
```

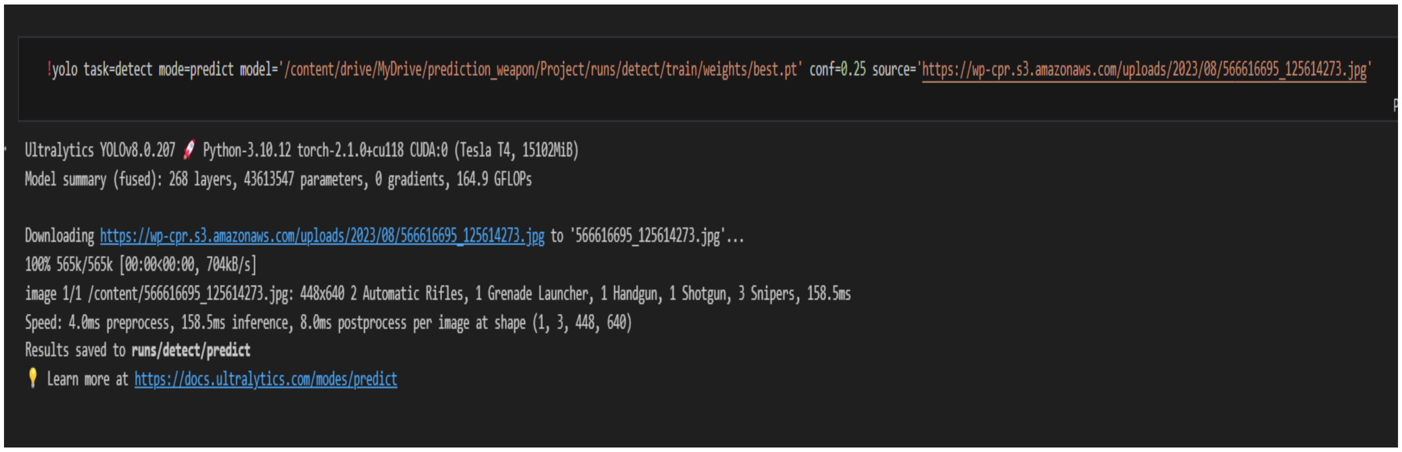
```
!yolo task=detect mode=train model=yolov8l.pt data=/content/drive/MyDrive/prediction_weapon/Project/Dataset/data.yaml epochs=250
```

```
    213/250      10.9G     0.5455     0.6324      1.095         44        640: 100% 30/30 [00:23<00:00,  1.27it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 7/7 [00:05<00:00,  1.36it/s]
                   all        210        323      0.623      0.544      0.585      0.417

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    214/250        11G     0.5454     0.6176      1.088         50        640: 100% 30/30 [00:23<00:00,  1.29it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 7/7 [00:05<00:00,  1.38it/s]
                   all        210        323      0.594      0.531      0.563      0.402

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    215/250      10.9G     0.5289      0.626      1.077         35        640: 100% 30/30 [00:23<00:00,  1.29it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 7/7 [00:04<00:00,  1.59it/s]
                   all        210        323      0.659       0.49      0.575      0.408

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    216/250        11G     0.5283     0.6144      1.087         45        640: 100% 30/30 [00:23<00:00,  1.27it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 7/7 [00:04<00:00,  1.48it/s]
                   all        210        323       0.63      0.563       0.59      0.414

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    217/250      10.9G     0.5406     0.6377      1.093         52        640: 100% 30/30 [00:23<00:00,  1.29it/s]
```



```
219 epochs completed in 1.956 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 87.7MB
Optimizer stripped from runs/detect/train/weights/best.pt, 87.7MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.216 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43613547 parameters, 0 gradients, 164.9 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 7/7 [00:09<00:00,  1.37s/it]
                   all        210        323      0.588      0.602      0.588      0.427
        Automatic Rifle        210         48      0.603      0.417      0.499      0.328
               Bazooka        210         22      0.671      0.727      0.706      0.493
       Grenade Launcher        210         23      0.515      0.554      0.539      0.398
               Handgun        210         73      0.783      0.593       0.65      0.486
                 Knife        210         62      0.791      0.661      0.674      0.455
                   SMG        210         27       0.37      0.778      0.624      0.513
               Shotgun        210         21      0.484      0.476      0.464      0.321
                Sniper        210         13      0.331      0.692      0.596      0.436
                 Sword        210         34      0.746      0.518      0.541      0.412
Speed: 0.7ms preprocess, 17.5ms inference, 0.0ms loss, 6.7ms postprocess per image
Results saved to runs/detect/train
💡 Learn more at https://docs.ultralytics.com/modes/train
```

```
[ ] !yolo task=detect mode=val model= '/content/drive/MyDrive/prediction_weapon/Project/runs/detect/train/weights/best.pt' data=/content/drive/MyDrive/prediction_weapon/Project/Dataset/data.yaml

Ultralytics YOLOv8.0.216 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43613547 parameters, 0 gradients, 164.9 GFLOPs
val: Scanning /content/drive/MyDrive/prediction_weapon/Project/Dataset/valid/labels.cache... 210 images, 0 backgrounds, 0 corrupt: 100% 210/210 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 14/14 [00:11<00:00,  1.23it/s]
                   all        210        323      0.593      0.595      0.588      0.427
        Automatic Rifle        210         48      0.618      0.417      0.498       0.33
               Bazooka        210         22      0.675      0.727      0.706      0.493
       Grenade Launcher        210         23      0.503      0.529       0.54      0.399
               Handgun        210         73       0.78      0.581       0.65      0.483
                 Knife        210         62      0.792      0.661      0.674      0.455
                   SMG        210         27       0.38      0.778      0.624      0.513
               Shotgun        210         21      0.492      0.476      0.464      0.321
                Sniper        210         13      0.364      0.692      0.592      0.434
                 Sword        210         34      0.736      0.493      0.541      0.412
Speed: 2.3ms preprocess, 36.9ms inference, 0.0ms loss, 3.5ms postprocess per image
Results saved to runs/detect/val
💡 Learn more at https://docs.ultralytics.com/modes/val
```



```
!yolo task=detect mode=predict model='/content/drive/MyDrive/prediction_weapon/Project/runs/detect/train/weights/best.pt' conf=0.25 source='https://wp-cpr.s3.amazonaws.com/uploads/2023/08/566616695_125614273.jpg'

Ultralytics YOLOv8.0.207 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43613547 parameters, 0 gradients, 164.9 GFLOPs

Downloading https://wp-cpr.s3.amazonaws.com/uploads/2023/08/566616695_125614273.jpg to '566616695_125614273.jpg'...
100% 565k/565k [00:00<00:00, 704kB/s]
image 1/1 /content/566616695_125614273.jpg: 448x640 2 Automatic Rifles, 1 Grenade Launcher, 1 Handgun, 1 Shotgun, 3 Snipers, 158.5ms
Speed: 4.0ms preprocess, 158.5ms inference, 8.0ms postprocess per image at shape (1, 3, 448, 640)
Results saved to runs/detect/predict
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

10.ADVANATAGES AND DISADVANTAGES

Advantages of Developing a YOLO-Based Weapon Detection System:

Real-Time Detection:

One of the primary advantages of using YOLO for weapon detection is its ability to process images and video frames in real-time. This enables immediate identification of weapons, allowing for swift responses to potential threats.

Unified Detection and Classification:

YOLO provides a unified approach to object detection and classification within a single pass through the neural network. This not only simplifies the implementation but also ensures that the system can identify and classify weapons simultaneously, streamlining the decision-making process.

Efficient Resource Utilization:

YOLO's efficiency in resource utilization is a significant advantage, particularly in scenarios where computational resources are limited. The algorithm's ability to perform object detection without the need for multiple passes through the network contributes to its computational efficiency.

Adaptability to Various Environments:

YOLO-based systems demonstrate adaptability to diverse environmental conditions. The model can be trained on datasets that encompass a wide range of scenarios, making it robust to variations in lighting, weather, and other environmental factors.

Minimal False Negatives:

YOLO's single-pass architecture helps minimize false negatives, reducing the likelihood of missing actual weapons in the detection process. This is crucial for maintaining a high level of accuracy and ensuring that potential threats are not overlooked.

User-Friendly Deployment:

The YOLO-based weapon detection system can be implemented with a user-friendly interface, facilitating seamless integration into existing security infrastructure. The real-time alerts and visualizations enhance the system's usability for security personnel.

Potential for Transfer Learning:

YOLO supports transfer learning, allowing the model to leverage pre-trained weights on large datasets. This is advantageous for weapon detection applications, as it enables the system to benefit from knowledge gained in other object recognition tasks.

Technological Advancement in Security:

Implementing a YOLO-based weapon detection system represents a technological advancement in security solutions. Leveraging deep learning for threat detection contributes to the evolution of intelligent security systems, aligning with the need for cutting-edge technologies in public safety.

Disadvantages of Developing a YOLO-Based Weapon Detection System:

False Positives and Negatives:

One of the primary drawbacks of YOLO-based weapon detection systems is the potential for false positives (incorrectly identifying non-weapons as weapons) and false negatives (missing actual weapons). Achieving a balance between precision and recall can be challenging, impacting the system's overall reliability.

Variability in Weapon Appearances:

Weapons come in diverse shapes, sizes, and orientations. Training a YOLO model to accurately detect this variability requires a comprehensive and representative dataset. Inadequate coverage of weapon types may lead to suboptimal performance.

Environmental Sensitivity:

YOLO-based systems may be sensitive to changes in environmental conditions such as lighting, weather, and occlusions. Adapting the model to handle varying conditions and ensuring robust performance in diverse settings is a complex task.

Computationally Demanding:

Implementing real-time processing for multiple surveillance cameras can be computationally demanding. The need for high processing power may limit the scalability of the system, particularly in resource-constrained environments.

Privacy Concerns:

The continuous monitoring required for weapon detection raises privacy concerns. Individuals within the surveillance area may feel their privacy is compromised, necessitating careful consideration of ethical and legal implications.

Adversarial Attacks:

YOLO-based systems, like any deep learning models, are susceptible to adversarial attacks. Malicious actors can manipulate input data to deceive the system, leading to incorrect weapon detections. Developing models that are robust against such attacks is an ongoing challenge.

Training Data Requirements:

Achieving optimal performance necessitates a large and diverse dataset for training. Obtaining and annotating such datasets can be time-consuming and resource-intensive, potentially posing challenges in scenarios with limited access to representative data.

<u>11.CONCLUSION</u>

In embarking on the journey to develop and implement a YOLO-based weapon detection system, we have navigated the complexities of cutting-edge technology in the pursuit of enhancing public safety and security. The project's endeavors have underscored both the promises and challenges inherent in harnessing the power of deep learning for real-time threat detection.

The advantages of adopting the YOLO algorithm for weapon detection are evident. The system's ability to provide rapid, unified detection and classification in real-time, coupled with its computational efficiency, presents a compelling case for its deployment in various security scenarios. The scalability across multiple cameras and the potential for transfer learning further position the YOLO-based system as a technological advancement in the domain of intelligent security solutions.

However, the journey has not been without its share of challenges. The nuanced variability in weapon appearances, environmental sensitivities, and the need for comprehensive datasets have demanded meticulous attention to detail during the development and training phases. Striking a balance between minimizing false positives and avoiding false negatives has proven to be a complex task, requiring ongoing refinement and optimization.

Moreover, ethical considerations, privacy concerns, and the potential for adversarial attacks underscore the importance of responsible deployment. As we tread into the realm of continuous monitoring for threat detection, it becomes imperative to address the ethical implications of surveillance and privacy infringement. The system's susceptibility to adversarial attacks necessitates robust security measures to ensure its reliability and effectiveness in the face of potential threats.

In conclusion, the journey of building and performing a YOLO-based weapon detection system has been a testament to the convergence of technology, security imperatives, and ethical considerations. It is a stride forward in leveraging the capabilities of deep learning for the greater good of public safety. The project's outcomes, with their strengths and limitations, contribute valuable insights to the broader discourse on intelligent security systems.

12. FUTURE SCOPE

The successful development and deployment of our YOLO-based weapon detection system lays a robust foundation for future advancements in the field of intelligent security. Moving forward, a key avenue for exploration involves the continuous refinement of the deep learning model through ongoing training with diverse and expansive datasets. This iterative process aims to improve the system's accuracy and extend its capabilities to handle an even broader spectrum of weapon types, environmental conditions, and complex scenarios.

A promising direction for future enhancement is the integration of multi-sensor systems. Incorporating additional sensors, such as infrared or acoustic sensors, holds the potential to further augment the system's capabilities. This expansion would make the system more robust in challenging lighting conditions and enable the detection of concealed weapons that may not be visible in standard visual imagery. The synergy of multiple sensor modalities can contribute to a more comprehensive and reliable threat detection system.

As we look ahead, there is a notable opportunity to advance the system's sophistication by incorporating advanced behavioral analysis and context understanding. This entails integrating additional machine learning models that can discern normal and abnormal human behaviors, adding a layer of nuance to the threat detection process. Such advancements would not only enhance the system's accuracy but also contribute to a more intelligent and context-aware security framework.

The future evolution of the YOLO-based weapon detection system should also consider the ongoing advancements in hardware technologies. Particularly, developments in graphics processing units (GPUs) and dedicated accelerators can expedite real-time processing. This ensures that the system maintains its efficiency even in scenarios with a higher volume of surveillance cameras, contributing to its scalability and applicability in diverse deployment scenarios.

To foster continuous improvement and adaptability, establishing a feedback loop mechanism is crucial. This mechanism would collect and analyze system performance data in real-world deployments, offering valuable insights for refining the model, addressing emerging challenges, and adapting to evolving threats. This iterative feedback loop ensures that the system remains dynamic and responsive to the evolving landscape of security challenges.

Additionally, the integration of explainable AI techniques represents a noteworthy avenue for future exploration. Enhancing the transparency of the system by providing insights into how the model arrives at its decisions can contribute to building trust among users. In applications with high stakes such as security, understanding the rationale behind the system's outputs becomes paramount, and explainable AI can address this need.

Exploring the feasibility of deploying the weapon detection system on edge devices is another avenue for future development. This shift towards edge computing can contribute to decentralized security frameworks, minimizing latency and bandwidth requirements. The adaptability of the system to edge computing opens new possibilities for deployment in remote or resource-constrained environments, enhancing its versatility.

Furthermore, the future scope extends to global collaboration for threat intelligence. Establishing collaborative frameworks that facilitate the sharing of threat intelligence can significantly enhance the system's capabilities. By integrating insights from various sources and security agencies, the system gains a more comprehensive understanding of emerging threats and patterns, contributing to more effective threat detection.

GitHub Link-

https://github.com/smartinternz02/SI-GuidedProject-615799-1700593412