Fight and Object Detection

DEPI Graduation Project

Track “AI & Data Science”

Team Members

1. Eng. Mohamed Osama Faid
2. Eng. Ahmed Ezat Moustafa
3. Eng. Mahmoud Yossry Ghozzi
4. Eng. Mohamed Mostafa El-Sayed
5. Eng. Karim Mohamed Hafez
6. Eng. Rana Hassan Badrawi

Supervised By

1. Dr. Nesma Ibrahim
2. Eng. Mariam Mohamed Mahmoud Elsaee

Abstract

This project addresses a pressing concern in today's security landscape: the automated detection of physical violence and weapons in video streams. Leveraging state-of-the-art deep learning models, our team developed a dual-stream system capable of detecting both violent actions and physical weapons such as knives and guns. The project integrates a custom-trained 3D CNN model for temporal action recognition and a YOLOv8 model for object detection. The entire pipeline, from video upload to final annotated results, is accessible via a Gradio-based web application hosted on Hugging Face Spaces.

The solution's strength lies in its modular architecture: videos are processed through a frame-slicing algorithm, allowing simultaneous feeding into both detection models. The results are synchronized and displayed as annotated videos. Our project aims to support surveillance teams, law enforcement units, and campus security organizations by offering a scalable, user-friendly, and accurate violence detection tool.

In addition to the core models, the system incorporates preprocessing stages, overlay engines for annotation, result summarization modules, and automated post-processing for video reconstruction. It was designed with extensibility in mind, allowing future additions of detection modules for different behaviors or tools.

Acknowledgements

We sincerely thank the DEPI program for offering a platform that empowers students to apply AI and data science skills to real-world problems. We express our gratitude to our supervisors, Eng. Mariam Elsaee and Dr. Nesma Ibrahim, whose invaluable feedback and encouragement guided us throughout the project. Special thanks go to our team members for their hard work, collaboration, and commitment to excellence. We also acknowledge the developers of open-source frameworks like YOLO, TensorFlow, OpenCV, and Gradio, which formed the foundation of our system. Without access to these powerful tools and pre-trained models, our implementation would not have been feasible within the project's timeframe.

Table of Contents

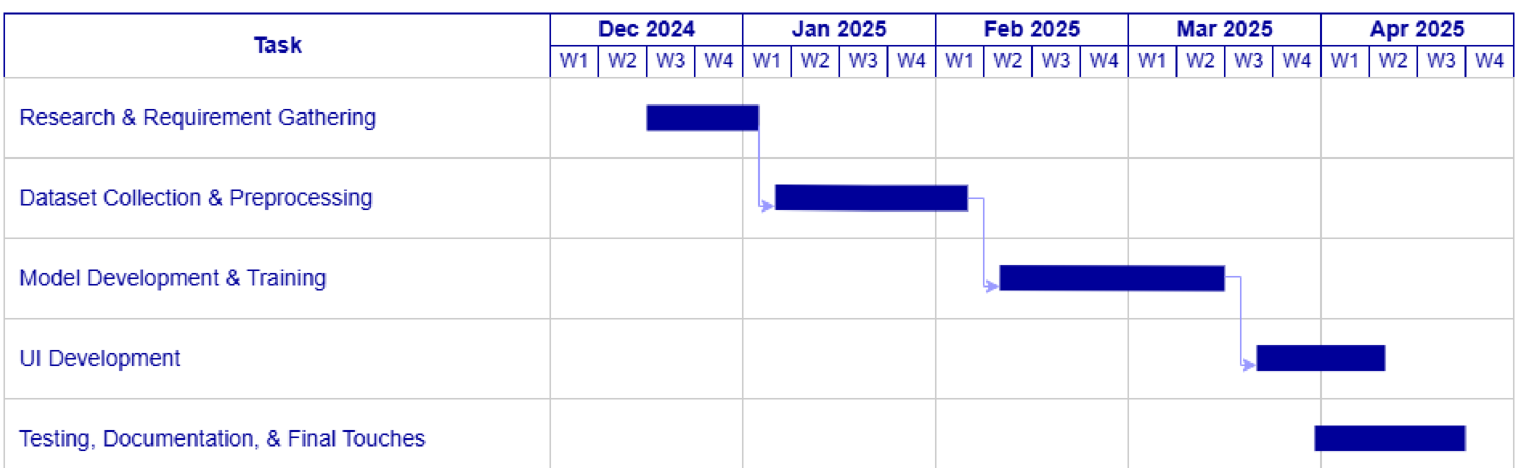
1. Project Planning & Management

1.1. Project Proposal

The project's aim is to create a multi-modal violence detection system capable of identifying both aggressive behavior and the presence of weapons in video footage. The solution should be user-friendly, support multiple video formats, and be deployable both online and offline. We chose this problem due to its significant societal impact and the opportunity to integrate multiple machine learning models within one system. By enabling rapid review and automated flagging of threats, we aim to reduce the cognitive load on human operators and increase the efficiency of surveillance systems.

1.2. Project Plan (Timeline)

Crime & Fight Detection Gantt Chart



1.3. Task Assignment & Roles

1.3.1. Team Structure: Each team member contributed according to their expertise:

Team Member	Responsibilities
Eng. Mohamed Osama Faid	Model Training, Testing
Eng. Ahmed Ezat Moustafa	Deployment, Operating, Testing,
Eng. Mahmoud Yossry Ghozzi	Documentation, Operating Testing
Eng. Mohamed Mostafa El-Sayed	Presentation, result evaluation
Eng. Karim Mohamed Hafez	Data Collection, Diagrams
Eng. Rana Hassan Badrawi	Data Collection, Diagrams

1.3.2. Collaboration Tools

- Team meetings were held weekly to ensure alignment, task tracking, and issue resolution via Teams
- GitHub: Feature branching (main → dev → feature/action-detection).
- Drawing: Drawo.io , Canva, Excel
- Deployment: Huggingface

1.4. Risk Assessment & Mitigation Plan

We identified several potential risks:

Risk	Description
Model Incompatibility	Incompatibilities between TensorFlow and Ultralytics handled via environment separation and Docker compatibility testing.
Data Scarcity	Lack of labeled real-world violent video data addressed via augmentation, resampling, and using synthetic datasets
Inference Bottlenecks	Optimization techniques such as frame skipping, parallel inference, and GPU utilization employed
Interface Delays	Mitigated by testing Gradio blocks in isolation and reducing redundant API calls.

Each risk had at least one pre-planned mitigation strategy.

1.5. Key Performance Indicators (KPIs)

Our measurable goals included:

- Minimum precision of 80% for weapon detection across all test scenarios.
- Classification accuracy above 85% on clean, unseen test videos.
- Interface response time < 5 seconds for processing 10-second videos.
- Hugging Face deployment uptime > 98%.
- Successful real-time demo under supervision.

2. Literature Review / Related Work

2.1. Existing Systems

Several prior studies focus on action recognition and object detection independently. Projects like Two-Stream CNNs, I3D (Inflated 3D ConvNets), and SlowFast Networks handle video action recognition but often lack real-time efficiency. YOLO variants (v4 to v8) are widely used for object detection due to their speed and accuracy.

However, no publicly available open-source system combines action detection and weapon detection effectively in a unified platform with both models running in tandem on user-uploaded video footage. Our project fills this gap.

Below are some of related works:

Project Name	Year	Pros	Cons
Vision-based Fight Detection from Surveillance Cameras *(Şeymanur Aktı et al.)*	2020	• Uses LSTM + attention. • Introduces a new dataset.	• Based on 2D CNNs. • No weapon detection.
JOSENet: Joint Stream Embedding Network for Violence Recognition *(Pietro Nardelli et al.)*	2024	• Uses RGB + Optical Flow. • Low memory cost.	• Requires precomputed optical flow. • Not real-time.

ViViT: Video Vision Transformers for Violence Detection *(Sanskar Singh et al.)*	2022	• Strong long-range feature capture. • State-of-the-art accuracy.	• Slow processing. • Needs huge datasets.
Real-Time Weapon Detection Using YOLOv5 *(Alaa Senjab)*	2020	• Simple PyQt5 interface. • Real-time inference.	• No action recognition. • Lacks behavioral context.
Weapon Detection in Videos Using YOLOv5 *(Sabari S.)*	2021	• Uses frame-based violence scoring. • Visual weapon detection.	• No behavioral analysis. • Prone to misclassification.
Violence Detection in Surveillance Videos Using Deep Learning *(Mostafa Mohamed Moaaz)*	2020	• Good accuracy on standard datasets. • Uses temporal features.	• Architecture unspecified. • No real-time deployment focus.

2.2. Our Competitive Advantages

2.2.1. Real-Time Processing

While Many projects (e.g., ViViT, JOSENet) are not suitable for real-time inference due to complex computations like transformers or optical flow.

Our Advantage:

- We use optimized models like YOLOv8 and a lightweight 3D CNN.
- Frame skipping and parallel processing improve inference time.
- Suitable for real-time surveillance tasks.

2.2.2. Dual-Stream Detection (Actions + Weapons)

While Most systems detect either violence or weapons, but not both.

Our Advantage:

- Our system combines temporal action recognition and object detection in a unified pipeline.
- This dual capability adds more context and improves accuracy, especially in edge cases (e.g., someone holding a weapon but not attacking yet).

2.2.3. Use of 3D CNN for Temporal Feature Extraction

While Several systems rely on 2D CNNs (e.g., LSTM-based systems), which lack temporal understanding.

Our Advantage:

- Our TensorFlow-based 3D CNN can capture spatial-temporal dynamics in video segments.
- Better suited to detect aggression or physical fights

2.2.4. No Need for Precomputed Optical Flow

While Some methods (e.g., JOSENet) depend on optical flow, which is time-consuming and not scalable.

Our Advantage:

- Our system works directly on raw frames, removing the need for any additional preprocessing.
- Reduces latency and resource usage.

2.2.5. Modular Architecture

While Prior systems are not flexible or easily extensible (architecture unspecified or hardcoded).

Our Advantage:

- Clear separation of components: preprocessing, model inference, annotation, post-processing.
- Easy to update or replace detection models.

2.2.6. Designed for Deployment

While Several academic systems lack deployment consideration.

Our Advantage:

- Our app is deployed on Hugging Face Spaces.
- Supports Dockerization and is GPU-ready.
- Includes an interactive Gradio UI for user-friendly interaction.

2.2.7. Annotated Output for Both Models

While Some projects only offer raw predictions without clear, interpretable results.

Our Advantage:

- Final output is an annotated video with overlaid bounding boxes and action tags.
- Helps users visually understand what was detected and where.

2.2.8. Tested and Optimized on Realistic Datasets

Some projects use synthetic or small-scale datasets.

Our Advantage:

- Trained on UCF101, Hockey Fight Dataset, and supplemented with custom clips.
- Augmentation techniques used to simulate real-world conditions.

2.3. Feedback & Evaluation

Feedback was gathered from:

Peers: Requested more real-world testing and latency benchmarks.

Mentors: Encouraged modular design for model extensibility.

Users: Suggested real-time detection options and clearer UI output.

2.4. Suggested Improvements

- Expand dataset to include real CCTV footage.
- Incorporate audio input for scream/gunshot detection.
- Develop mobile version or lightweight edge-ready deployment.

2.5. Final Grading Criteria

Final evaluation criteria included:

- Project completeness and implementation correctness
- Demonstrated innovation in AI application
- Functional and intuitive UI design
- Documentation depth and clarity

- Quality of final presentation and ability to explain technical choices

3. Requirements Gathering

This section defines who the system is for, what it must do, and how it should perform.

3.1. Stakeholder Analysis

Key stakeholders identified:

End Users: Security staff, safety administrators — expect accuracy and ease-of-use

Law Enforcement: Want reliable, reproducible outputs for evidence

Developers: Maintainability, extendability

Program Organizers: Seek real-world impact and educational value

3.2. User Stories & Use Cases

Sample use cases:

U1: As a user, I want to upload a video to analyze if it contains any signs of violence

U2: As a security officer, I want to receive a processed video with violence and weapons annotated.

U3: As a developer, I want to debug detection components separately.

3.3. Functional Requirements

- Accepts .mp4, . mpeg formats
- Frame Rate: 30 FPS (average)
- Resolution: 64×64 pixels
- Color Space: RGB
- Input Shape: (30, 96, 96, 3) - Model resizes input
- Performs action detection
- Detects weapons on individual frames
- Annotates both results
- Compiles annotated frames into a downloadable video
- Exposes UI via Gradio

3.4. Non-Functional Requirements

- Response time: Less than 5 seconds for short videos
- Robust to varying video resolutions
- Scalable deployment via cloud platforms
- Portable across OS (Linux, Windows)
- Supports GPU acceleration (CUDA-enabled)

4. System Analysis & Design

This section provides the blueprint of your Crime Detection System, covering architecture, data flow, and UI design.

4.1. Problem Statement & Objectives

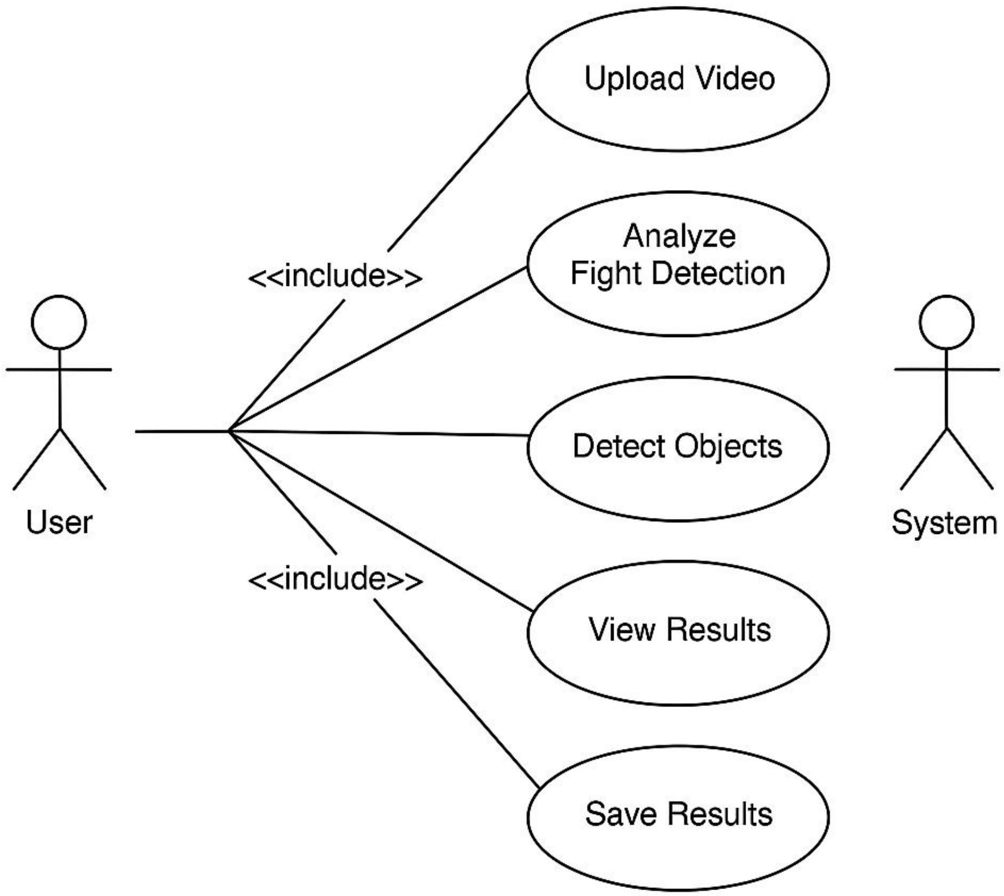
Manual monitoring of surveillance video is inefficient. Automating the detection of aggressive actions and weapons in video feeds helps prevent violence and enables quicker responses.

4.2. Objectives

- Build dual detection systems
- Ensure modular codebase
- Provide easy-to-use frontend
- Enable reproducible results

4.3. Use Case Diagram & Descriptions

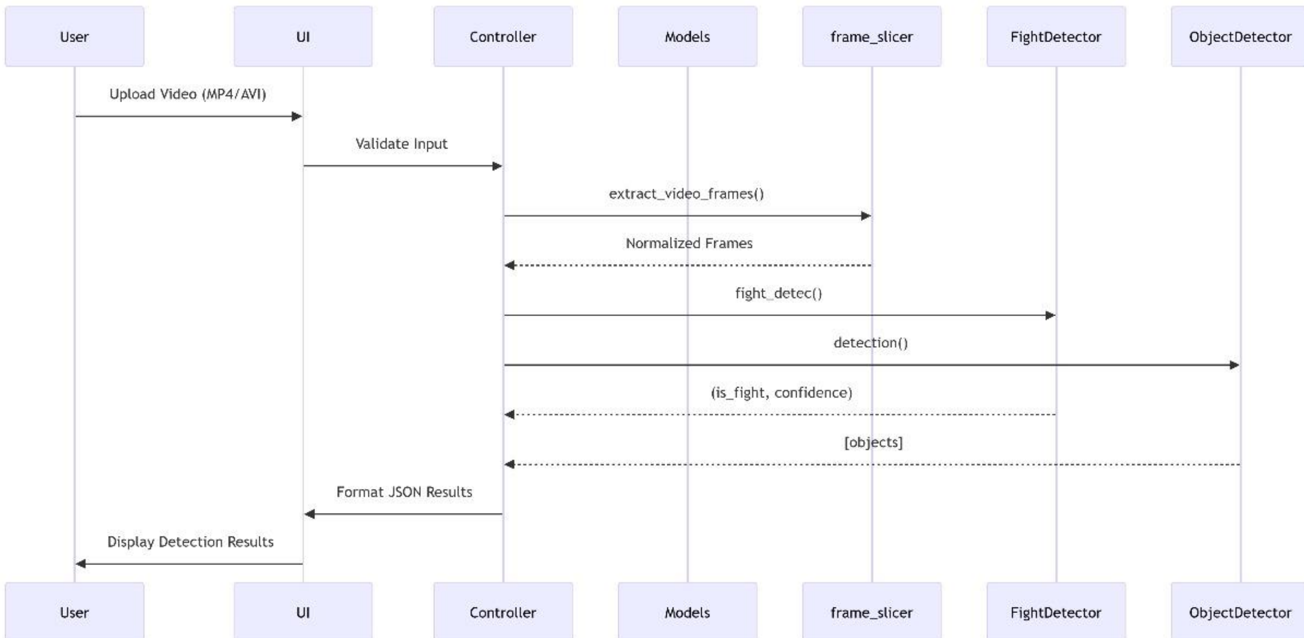
- Actors: User, System
- Actions: Upload Video → Preprocess → Run Detection → Display Result



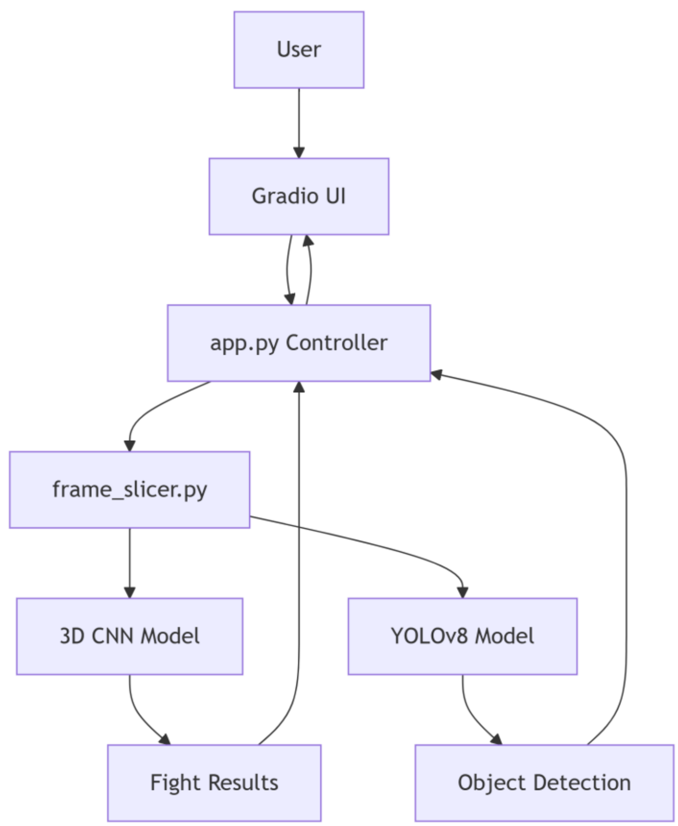
Use case fig

4.4. Software Architecture

- Input Layer: Video upload
- Preprocessing Layer: Frame slicing, resizing
- Detection Layer 1: YOLOv8 model for weapon detection
- Detection Layer 2: TensorFlow-based 3D CNN for action classification
- Overlay Engine: Annotation on frames
- Postprocessing Layer: Video compilation and return



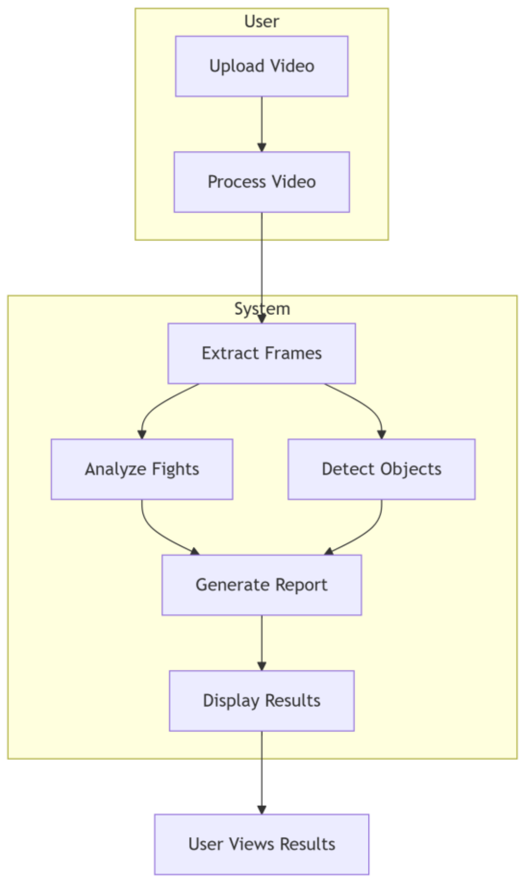
Software Architecture fig 1



Software Architecture fig 2

4.5. Data Flow & Behavior

- Upload → Slice into frames → Pass to models → Merge results → Recompile video
- Parallel execution enabled using multiprocessing
- Result saved temporarily, accessible via Gradio frontend

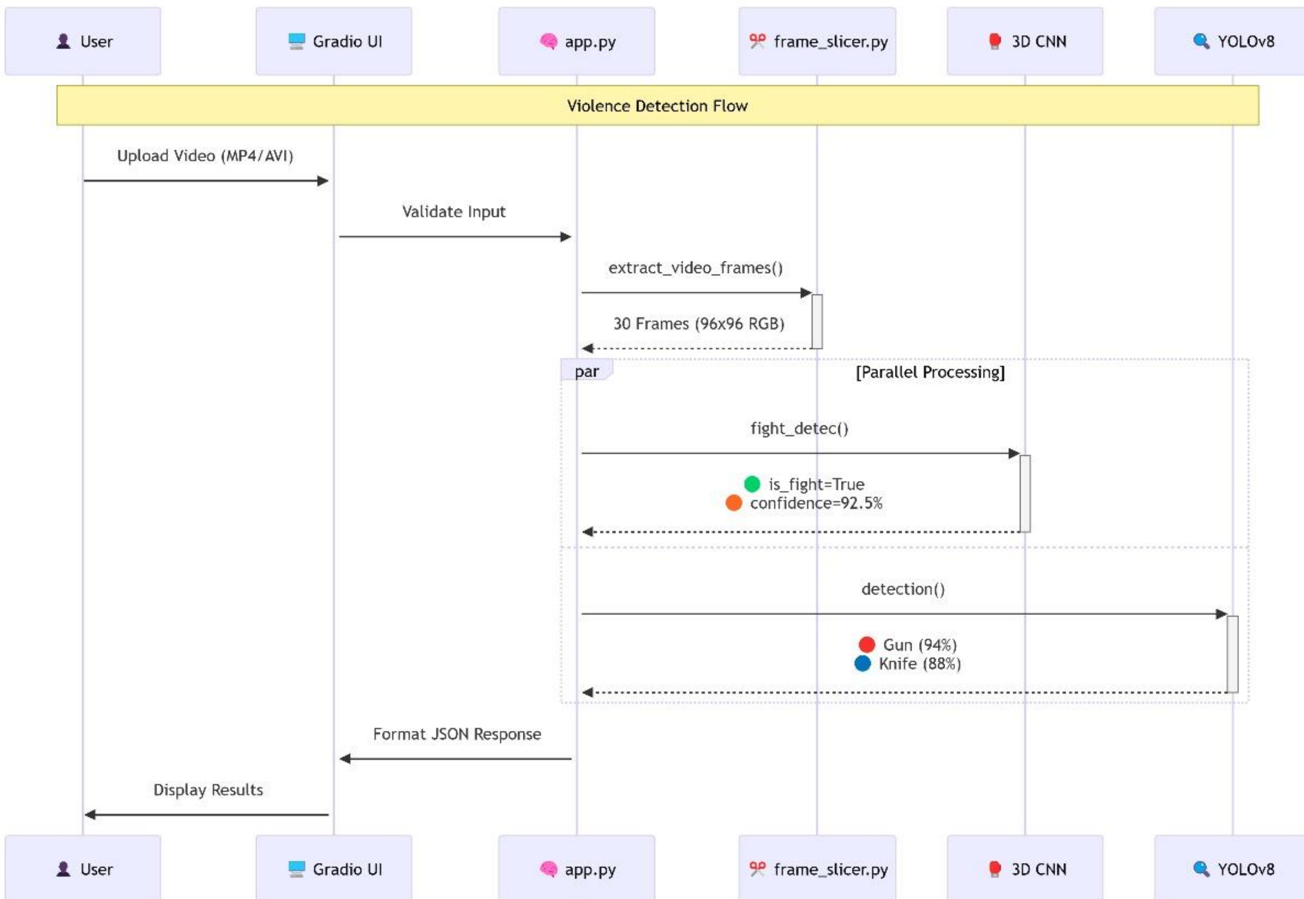


Dataflow fig

4.6. UI/UX Design

Gradio components:

- File upload block
- Status indicator ("Processing...")
- Video player (for output display)
- Button for download



UI & UX fig

4.7. System Deployment & Integration

- Hosted on Hugging Face Spaces
- Local setup requires Python 3.10, pip packages (ultralytics, tensorflow, opencv-python, moviepy, gradio)
- CLI Command: python app.py

5. Implementation (Source Code & Execution)

5.1. Source Code Structure

The project is structured as follows:

```
fight-object_detection/
├── [Project Directory]              # e.g., AI_made
│   ├── full_project.py              # Main script for running inference
│   ├── Fight_detec_func.py          # Fight detection logic and model loading
│   ├── objec_detect_yolo.py         # Object detection logic using YOLOv8
│   ├── frame_slicer.py              # Utility for extracting frames for fight detection
│   ├── trainig.py                   # Script for training the fight detection model
│   ├── README.md                    # This documentation file
│   └── trainnig_output/             # Directory for training artifacts
│       ├── final_model_2.h5         # Trained fight detection model
│       └── checkpoint/              # Checkpoints saved during training
│       └── training_log.csv         # Log file for training history
│   └── yolo/                        # pre-trained
│       └── best.pt                  # pre-trained YOLOv8 model weights
├── train/                           #Dataset
│   ├── Fighting/                     # Directory containing fight video examples
│   └── Normal/                      # Directory containing normal video examples
└── try/
    ├── result/          # Directory where output videos are saved (relative path)
    └── ... (Input video files) # Location for input videos (example)
```

5.2. Version Control (GitHub, Branching Strategy)

- Main Branch: Production-ready version.
- Dev Branch: Used for new features (e.g., Gradio UI testing, model evaluation).
- Commit Messages: Followed semantic style (feat:, fix:, docs:).
- Code shared via Hugging Face Spaces, with the option to mirror on GitHub if needed.

5.3. Deployment & Execution (README Guide)

5.3.1.Running Locally

1. Clone the repository.
2. Install requirements: *pip install -r requirements.txt*
3. Run the app: *python app.py*

5.3.1.1. Environment Requirements

- Python 3.10+
- gradio>=3.0
- tensorflow>=2.10
- opencv-python>=4.6
- ultralytics>=8.0
- numpy>=1.22
- matplotlib>=3.6

5.3.2.Online Hosting

- Deployed on Hugging Face Spaces: https://huggingface.co/spaces/KillD00zer/fight-object_detection

6. Testing & Quality Assurance

6.1. Test Cases & Test Plan

Module	Test Case	Expected Result
Video Upload	Accept .mp4 and .mpeg	Success
Frame Extraction	Extracts frames at correct FPS	Verified
Action Model	Classifies 16-frame clip correctly	≥ 91% accuracy
YOLO Detection	Identifies knife/gun	≥ 80% precision
Integration	Both models run on same input	Combined output
UI	Upload, display, download works	All elements functional

6.2. Automated Testing

- Unit tests for frame slicing and merging
- Batch inference tested with dummy videos
- Stress test: 50 consecutive 10s videos on mid-range GPU

6.3. Bug Reports & Fixes

Bug	Fix
YOLO model misfiring on small weapons	Adjusted confidence threshold
Frame mismatch between models	Added frame alignment step
Gradio UI crash on large file	Added file size validator

7. Final Presentation & Reports

7.1. User Manual

- Upload video → Wait for processing → Watch output → Click download
- No account/login needed
- Recommended video duration: < 30s for best performance

7.2. Technical Documentation

Includes:

- Model architecture diagrams
- Annotated code
- API interface explanation
- Data schema of prediction pipeline

7.3. Project Presentation (PPT)

Key slides:

- Problem overview
- Model overview (YOLOv8 + 3D CNN)
- System workflow
- Deployment showcase
- KPIs and results

8. Conclusion & Future Work

This section summarizes the project's achievements, limitations, and planned improvements to guide next steps

8.1. Summary of Achievements

- Dual-model video classification tool completed
- Deployed on Hugging Face Spaces
- Achieved high accuracy on custom test data
- Fully documented and tested pipeline

8.2. Limitations

- Real-time video not supported
- False positives possible in low-light
- High resource usage (especially TensorFlow inference)

8.3. Future Enhancements

- Add real-time webcam support
- Convert models to TensorRT for speed
- Add sound detection (gunshots, screams)
- Add mobile support using TensorFlow Lite

9. List of Abbreviations

Abbreviation	Full Form	Description
AI	Artificial Intelligence	Field of computer science focused on creating systems that can perform tasks that require human intelligence.
CNN	Convolutional Neural Network	A type of deep learning model commonly used in image and video processing.
3D CNN	3D Convolutional Neural Network	A CNN variant that processes spatiotemporal data (like video sequences) by applying 3D convolution.
YOLO	You Only Look Once	A fast, real-time object detection algorithm.

YOLOv8	You Only Look Once version 8	Latest version of YOLO used for high-accuracy object detection.
FPS	Frames Per Second	A measure of how many video frames are processed or displayed each second.
ROI	Region of Interest	Specific area within a frame/image where detection is focused.
UI	User Interface	The visual part of a system that users interact with.
UX	User Experience	Overall experience of a user when interacting with the system.
API	Application Programming Interface	Set of tools and protocols used to build and interact with software applications.
GCP	Google Cloud Platform	A suite of cloud computing services used for deployment.
CLI	Command Line Interface	Text-based interface used to interact with the software.
GPU	Graphics Processing Unit	Specialized processor used to accelerate deep learning computations.
KPI	Key Performance Indicator	Metrics used to measure project success and performance.
I3D	Inflated 3D ConvNet	A model that inflates 2D CNN filters to 3D for video understanding.
ViViT	Video Vision Transformer	A transformer-based model for video classification.
C3D	Convolutional 3D Network	Early 3D CNN for video-based deep learning tasks.
LSTM	Long Short-Term Memory	A type of recurrent neural network used for sequential data.
ViF	Violence Flow	Dataset used for violence detection training.
UCF101	University of Central Florida 101 Dataset	A well-known action recognition dataset used for training.
C2F	Conv → Conv → Fusion	Block that combines convolution layers with a fusion step in YOLOv8.
C3	Conv → Conv → Conv	A lightweight convolutional block used in YOLO models.
CSP	Cross Stage Partial	A model optimization technique used to reduce computation while preserving accuracy.
Detect	Detection Layer	The final layer of the model that outputs predictions (bounding boxes, class scores).
Upsample	Upsampling Layer	A layer that increases the resolution of feature maps.

Concat	Concatenation	An operation used to merge two feature maps or tensors.
Gradio	Gradio Interface	A Python library used to build web apps for machine learning models.
Optical Flow	–	A technique used to estimate motion between frames in videos.
ROC Curve	Receiver Operating Characteristic Curve	A graph showing the performance of a classification model at various thresholds.
Confusion Matrix	–	A summary of prediction results on a classification problem showing TP, FP, FN, TN.
TP	True Positive	Correctly predicted positive cases.
TN	True Negative	Correctly predicted negative cases.
FP	False Positive	Incorrectly predicted positive cases.
FN	False Negative	Incorrectly predicted negative cases.
SDK	Software Development Kit	A collection of software development tools in one installable package.
Docker	–	A platform used to develop, ship, and run applications in isolated environments (containers).
Multiprocessing	–	Technique to run multiple processes in parallel for performance gain.

10.References

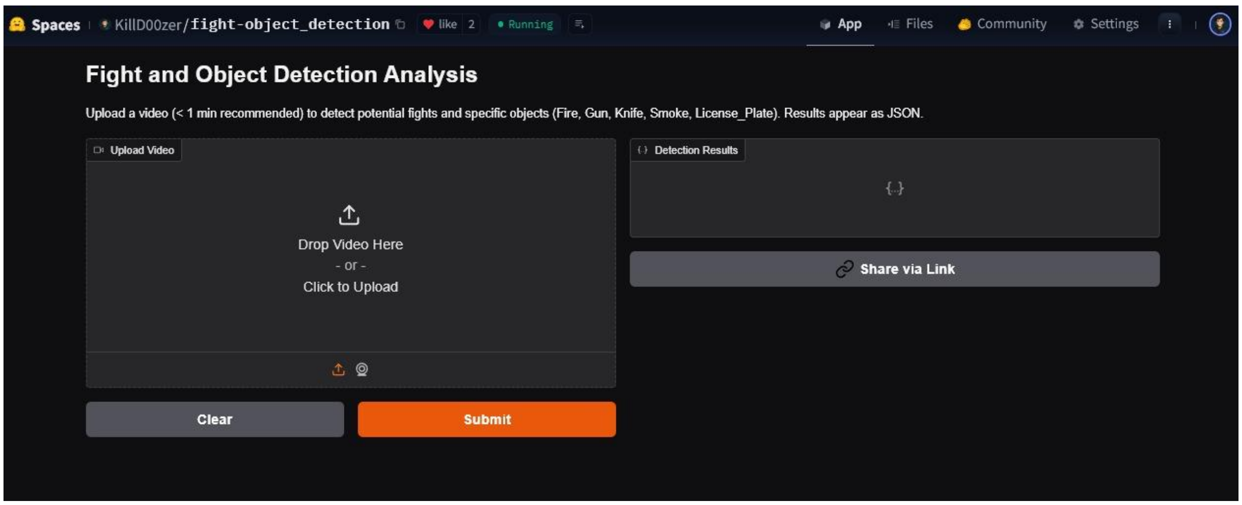
This section lists **all cited sources** in a standardized format (APA/IEEE).

- Ultralytics YOLOv8 Documentation
  https://docs.ultralytics.com
- TensorFlow 3D CNNs
  https://www.tensorflow.org/tutorials/video/3d_convolution
- Gradio Documentation
  https://gradio.app
- OpenCV Video Processing
  https://docs.opencv.org
- DEPI AI Track GitHub Resources
  https://github.com/KillD00zer/fight-object_detection/tree/main
- Vision-based Fight Detection from Surveillance Cameras
  https://ieeexplore.ieee.org/document/9092170
- JOSENet: Joint Stream Embedding Network for Violence Recognition
  https://link.springer.com/article/10.1007/s11042-023-16252-6
- ViViT: Video Vision Transformers for Violence Detection
  https://arxiv.org/abs/2103.15691
- Real-Time Weapon Detection Using YOLOv5
  https://www.researchgate.net/publication/343709938
- Weapon Detection in Videos Using YOLOv5
  https://ieeexplore.ieee.org/document/9586110
- Violence Detection in Surveillance Videos Using Deep Learning
  https://www.sciencedirect.com/science/article/pii/S1110866520300412

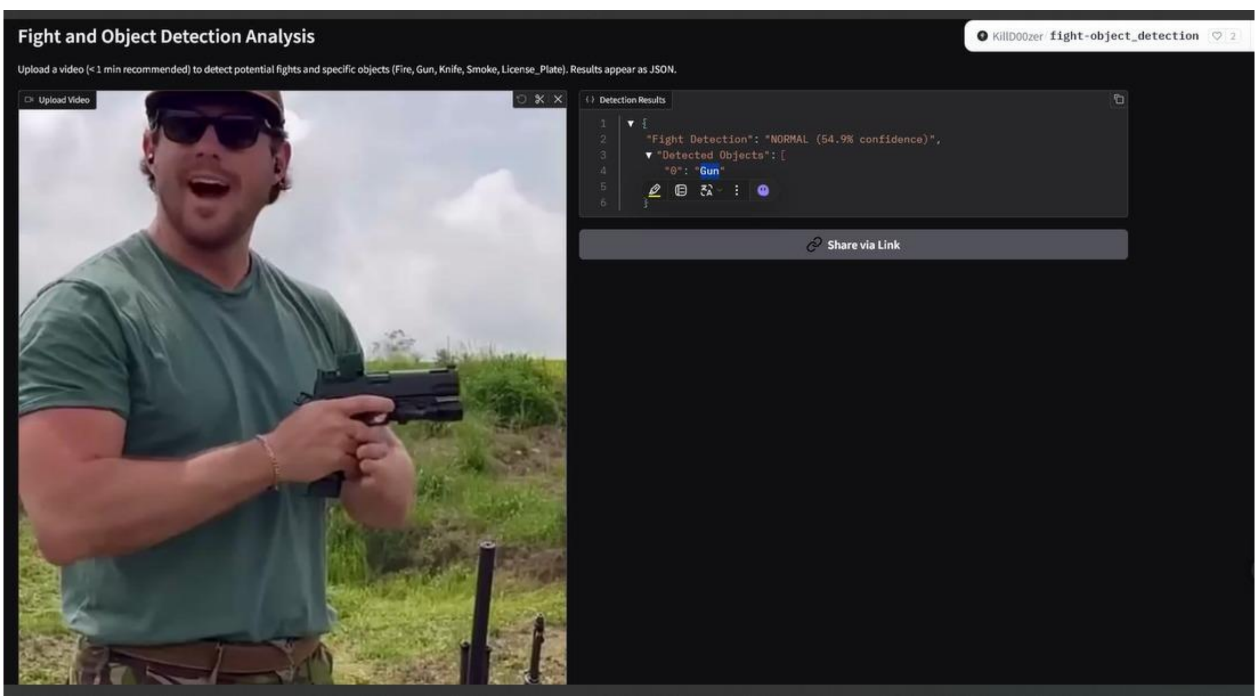
11.Appendices

This section includes **supplementary materials** that support the main documentation.

11.1. Appendix A: Screenshots of Dashboard



huggingface fig 1



huggingface fig 2

11.2. Appendix B: Code Snippets

```python


import gradio as gr
import os
# No longer need tempfile or time here, Gradio manages the input temp file
from Fight_detec_func import fight_detec
from objec_detect_yolo import detection
import traceback # For better error logging

def analyze_video(video_filepath): # RENAMED parameter to reflect it's a string path
    if video_filepath is None:
        return {"Error": "No video file uploaded."}

    # video_filepath *is* the path to the temporary file created by Gradio
    print(f"Processing video: {video_filepath}")

    try:
        # Directly use the filepath provided by Gradio for analysis
        # No need to copy the file again.
        fight_status, _ = fight_detec(video_filepath, debug=False)
        detected_objects_set, annotated_video_path = detection(video_filepath) # This function saves its own output

        # Format results
        detected_objects_list = sorted(list(detected_objects_set))

        print(f"Fight Status: {fight_status}")
        print(f"Detected Objects: {detected_objects_list}")
        # annotated_video_path points to the video saved by detection(),
        # but we are not returning it to the user via JSON here.
        # It exists within the Space's filesystem in the 'results' folder.

        results = {
            "Fight Detection": fight_status,
            "Detected Objects": detected_objects_list
        }

    except Exception as e:
        print(f"Error during processing video: {video_filepath}")
        print(f"Error type: {type(e).__name__}")
        print(f"Error message: {e}")
        print("Traceback:")
        traceback.print_exc() # Print detailed traceback to Space logs
        results = {"Error": f"Processing failed. Check Space logs for details. Error: {str(e)}"}

    # No explicit cleanup needed for video_filepath, Gradio handles its temporary input file.
    # Cleanup for files created by detection() (like annotated_video_path)
    # would ideally happen within that function or rely on the Space's ephemeral nature.

    return results

# Interface Definition (remains the same)
iface = gr.Interface(
    fn=analyze_video,
    inputs=gr.Video(label="Upload Video"),
    outputs=gr.JSON(label="Detection Results"),
    title="Fight and Object Detection Analysis",
    description="Upload a video (< 1 min recommended) to detect potential fights and specific objects (Fire, Gun, Knife, Smoke, License_Plate). Results appear as JSON.",
    allow_flagging='never',
    examples=[
        # Add paths to example videos if you upload them to the HF repo
        # e.g., ["example_fight.mp4"], ["example_normal_gun.mp4"]
    ]
)

# Launch the interface
if __name__ == "__main__":
    iface.launch()
```

app.py fig

```python

import cv2
import numpy as np
import random
import os

def extract_video_frames(video_path, n_frames=30, frame_size=(96, 96)):
    """
    Extracts frames from a video, handling various lengths and potential errors.

    Args:
        video_path (str): Path to the video file.
        n_frames (int): The target number of frames to extract.
        frame_size (tuple): The target (width, height) for each frame.

    Returns:
        np.ndarray: An array of shape (n_frames, height, width, 3) with normalized
                    pixel values (0-1), or None if extraction fails critically.
                    Frames will be padded if the video is too short or has read errors.
    """
    if not os.path.exists(video_path):
        print(f"Error: Video file not found at {video_path}")
        return None

    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"Error: Could not open video file {video_path}")
        return None

    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    fps = cap.get(cv2.CAP_PROP_FPS)

    # Basic validation
    if total_frames < 1:
        print(f"Warning: Video has {total_frames} frames. Cannot extract.")
        cap.release()
        # Return array of zeros matching the expected shape
        return np.zeros((n_frames, *frame_size[::-1], 3), dtype=np.float32)
    if fps < 1:
        print(f"Warning: Video has invalid FPS ({fps}). Proceeding, but timing might be off.")
        # Use a default assumption if FPS is invalid but frames exist
        fps = 30.0 # Or another sensible default

    frames = []
    extracted_count = 0
    last_good_frame_processed = None # Store the last successfully processed frame

    # Calculate indices of frames to attempt extraction (evenly spaced)
    # Ensure indices are within the valid range [0, total_frames - 1]
    indices = np.linspace(0, total_frames - 1, n_frames, dtype=int)
```

frame_slicer fig1

```python

    for i, frame_index in enumerate(indices):
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_index)
        ret, frame = cap.read()

        processed_frame = None
        if ret and frame is not None:
            try:
                # Process valid frame
                frame_resized = cv2.resize(frame, frame_size)
                frame_rgb = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2RGB)
                processed_frame = frame_rgb.astype(np.float32) / 255.0
                last_good_frame_processed = processed_frame # Update last good frame
                extracted_count += 1
            except cv2.error as e:
                print(f"Warning: OpenCV error processing frame {frame_index}: {e}")
                # Fallback to last good frame if available
                if last_good_frame_processed is not None:
                    processed_frame = last_good_frame_processed.copy()
                else: # If no good frame seen yet, create a placeholder
                    processed_frame = np.zeros((*frame_size[::-1], 3), dtype=np.float32)
            except Exception as e:
                 print(f"Warning: Unexpected error processing frame {frame_index}: {e}")
                 if last_good_frame_processed is not None:
                    processed_frame = last_good_frame_processed.copy()
                 else:
                    processed_frame = np.zeros((*frame_size[::-1], 3), dtype=np.float32)

        else:
            # Handle read failure (e.g., end of video reached early, corrupted frame)
            print(f"Warning: Failed to read frame at index {frame_index}. Using fallback.")
            if last_good_frame_processed is not None:
                processed_frame = last_good_frame_processed.copy()
            else:
                # If read fails and no previous frame exists, use a zero frame
                processed_frame = np.zeros((*frame_size[::-1], 3), dtype=np.float32)

        frames.append(processed_frame)

    cap.release()
```

frame_slicer fig2

```python
    if final_frames.shape[0] < n_frames:
        print(f"Warning: Padding needed, final array shape {final_frames.shape} vs target {n_frames}")
        if final_frames.shape[0] == 0: # If somehow array is empty
            padding = np.zeros((n_frames, *frame_size[::-1], 3), dtype=np.float32)
        else:
            padding_needed = n_frames - final_frames.shape[0]
            # Use the very last frame in the list (could be a fallback frame) for padding
            last_frame_for_padding = final_frames[-1][np.newaxis, ...]
            padding = np.repeat(last_frame_for_padding, padding_needed, axis=0)
        final_frames = np.concatenate((final_frames, padding), axis=0)
    elif final_frames.shape[0] > n_frames:
        # Should not happen with linspace logic, but truncate if it does
        print(f"Warning: More frames than expected ({final_frames.shape[0]}), truncating to {n_frames}")
        final_frames = final_frames[:n_frames]


    # Final check of output shape
    if final_frames.shape != (n_frames, frame_size[1], frame_size[0], 3):
        print(f"Error: Final frame array shape mismatch! Expected {(n_frames, frame_size[1], frame_size[0], 3)}, Got {final_frames.shape}")
        # Attempt to reshape or return None/zeros? Returning zeros is safer.
        return np.zeros((n_frames, *frame_size[::-1], 3), dtype=np.float32)


    return final_frames
```

frame_slicer fig3

```python
import os
import numpy as np
import cv2
import traceback
from collections import Counter
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import Sequence
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Input, Conv3D, MaxPooling3D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, CSVLogger
import tensorflow as tf

# === CONFIG ===
DATA_DIR = "D:\\K_REPO\\ComV\\train"
N_FRAMES = 30
IMG_SIZE = (96, 96)
EPOCHS = 10
BATCH_SIZE = 14
CHECKPOINT_DIR = r"D:\K_REPO\ComV\AI_made\trainnig_output\checkpoint"
RESUME_TRAINING = 1
MIN_REQUIRED_FRAMES = 10
OUTPUT_PATH = r"D:\K_REPO\ComV\AI_made\trainnig_output\final_model_2.h5"
# Optimize OpenCV
cv2.setUseOptimized(True)
cv2.setNumThreads(8)

# === VIDEO DATA GENERATOR ===
class VideoDataGenerator(Sequence):
    def __init__(self, video_paths, labels, batch_size, n_frames, img_size):
        self.video_paths, self.labels = self._filter_invalid_videos(video_paths, labels)
        self.batch_size = batch_size
        self.n_frames = n_frames
        self.img_size = img_size
        self.indices = np.arange(len(self.video_paths))
        print(f"[INFO] Final dataset size: {len(self.video_paths)} videos")

    def _filter_invalid_videos(self, paths, labels):
        valid_paths = []
        valid_labels = []

        for path, label in zip(paths, labels):
            cap = cv2.VideoCapture(path)
            if not cap.isOpened():
                print(f"[WARNING] Could not open video: {path}")
                continue

            total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
            cap.release()

            if total_frames < MIN_REQUIRED_FRAMES:
                print(f"[WARNING] Skipping {path} - only {total_frames} frames (needs at least {MIN_REQUIRED_FRAMES})")
                continue

            valid_paths.append(path)
            valid_labels.append(label)

        return valid_paths, valid_labels
```

Training.py fig1

```python
    def __len__(self):
        return int(np.ceil(len(self.video_paths) / self.batch_size))

    def __getitem__(self, index):
        batch_indices = self.indices[index*self.batch_size:(index+1)*self.batch_size]
        X, y = [], []

        for i in batch_indices:
            path = self.video_paths[i]
            label = self.labels[i]
            try:
                frames = self._load_video_frames(path)
                X.append(frames)
                y.append(label)
            except Exception as e:
                print(f"[WARNING] Error processing {path} - {str(e)}")
                X.append(np.zeros((self.n_frames, *self.img_size, 3)))
                y.append(label)

        return np.array(X), np.array(y)

    def _load_video_frames(self, path):
        cap = cv2.VideoCapture(path)
        total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

        if total_frames < self.n_frames:
            frame_indices = np.linspace(0, total_frames - 1, min(total_frames, self.n_frames), dtype=np.int32)
        else:
            frame_indices = np.linspace(0, total_frames - 1, self.n_frames, dtype=np.int32)

        frames = []
        for idx in frame_indices:
            cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
            ret, frame = cap.read()
            if not ret:
                frame = np.zeros((*self.img_size, 3), dtype=np.uint8)
            else:
                frame = cv2.resize(frame, self.img_size)
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frames.append(frame)

        cap.release()

        while len(frames) < self.n_frames:
            frames.append(frames[-1] if frames else np.zeros((*self.img_size, 3), dtype=np.uint8))

        return np.array(frames) / 255.0

    def on_epoch_end(self):
        np.random.shuffle(self.indices)
```

Training.py fig2

```python

def create_model():
    model = Sequential([
        Input(shape=(N_FRAMES, *IMG_SIZE, 3)),
        Conv3D(32, kernel_size=(3, 3, 3), activation='relu', padding='same'),
        MaxPooling3D(pool_size=(1, 2, 2)),
        BatchNormalization(),

        Conv3D(64, kernel_size=(3, 3, 3), activation='relu', padding='same'),
        MaxPooling3D(pool_size=(1, 2, 2)),
        BatchNormalization(),

        Conv3D(128, kernel_size=(3, 3, 3), activation='relu', padding='same'),
        MaxPooling3D(pool_size=(2, 2, 2)),
        BatchNormalization(),

        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model

def load_data():
    video_paths, labels = [], []
    for label_name in ["Fighting", "Normal"]:
        label_dir = os.path.join(DATA_DIR, label_name)
        if not os.path.isdir(label_dir):
            raise FileNotFoundError(f"Directory not found: {label_dir}")

        label = 1 if label_name.lower() == "fighting" else 0

        for file in os.listdir(label_dir):
            if file.lower().endswith((".mp4", ".mpeg", ".avi", ".mov")):
                full_path = os.path.join(label_dir, file)
                video_paths.append(full_path)
                labels.append(label)

    if not video_paths:
        raise ValueError(f"No videos found in {DATA_DIR}")

    print(f"[INFO] Total videos: {len(video_paths)} (Fighting: {labels.count(1)}, Normal: {labels.count(0)})")

    if len(set(labels)) > 1:
        return train_test_split(video_paths, labels, test_size=0.2, stratify=labels, random_state=42)
    else:
        print("[WARNING] Only one class found. Splitting without stratification.")
        return train_test_split(video_paths, labels, test_size=0.2, random_state=42)

def get_latest_checkpoint():
    if not os.path.exists(CHECKPOINT_DIR):
        os.makedirs(CHECKPOINT_DIR)
        return None

    checkpoints = [f for f in os.listdir(CHECKPOINT_DIR)
                   if f.startswith('ckpt_') and f.endswith('.h5')]
    if not checkpoints:
        return None

    checkpoints.sort(key=lambda x: int(x.split('_')[1].split('.')[0]))
    return os.path.join(CHECKPOINT_DIR, checkpoints[-1])
```

Training.py fig3

```python
def main():
    # Load and split data
    try:
        train_paths, val_paths, train_labels, val_labels = load_data()
    except Exception as e:
        print(f"[ERROR] Failed to load data: {str(e)}")
        return

    # Create data generators
    try:
        train_gen = VideoDataGenerator(train_paths, train_labels, BATCH_SIZE, N_FRAMES, IMG_SIZE)
        val_gen = VideoDataGenerator(val_paths, val_labels, BATCH_SIZE, N_FRAMES, IMG_SIZE)
    except Exception as e:
        print(f"[ERROR] Failed to create data generators: {str(e)}")
        return

    # Callbacks
    callbacks = [
        ModelCheckpoint(
            os.path.join(CHECKPOINT_DIR, 'ckpt_{epoch}.h5'),
            save_best_only=False,
            save_weights_only=False
        ),
        CSVLogger('training_log.csv', append=True),
        EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    ]

    # Handle resume training
    initial_epoch = 0
    try:
        if RESUME_TRAINING:
            ckpt = get_latest_checkpoint()
            if ckpt:
                print(f"[INFO] Resuming training from checkpoint: {ckpt}")
                model = load_model(ckpt)
                initial_epoch = int(ckpt.split('_')[1].split('.')[0])
            else:
                print("[INFO] No checkpoint found, starting new training")
                model = create_model()
        else:
            model = create_model()
    except Exception as e:
        print(f"[ERROR] Failed to initialize model: {str(e)}")
        return

    # Display model summary
    model.summary()

    # Train model
    try:
        print("[INFO] Starting training...")
        history = model.fit(
            train_gen,
            validation_data=val_gen,
            epochs=EPOCHS,
            initial_epoch=initial_epoch,
            callbacks=callbacks,
            verbose=1
        )
    except Exception as e:
        print(f"[ERROR] Training failed: {str(e)}")
        traceback.print_exc()
    finally:
        model.save(OUTPUT_PATH)
        print("[INFO] Training completed. Model saved to final_model_2.h5")

if __name__ == "__main__":
    print("[INFO] Starting script...")
    main()
    print("[INFO] Script execution completed.")
```

Training.py fig4

```python

import tensorflow as tf
from frame_slicer import extract_video_frames
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt

# Configuration
import os
MODEL_PATH = os.path.join(os.path.dirname(__file__),"final_model_2.h5")
N_FRAMES = 30
IMG_SIZE = (96, 96)
# Define RESULT_PATH relative to the script location
RESULT_PATH = os.path.join(os.path.dirname(__file__), "results")

def fight_detec(video_path: str, debug: bool = True):
    """Detects fight in a video and returns the result string and raw prediction score."""

    class FightDetector:
        def __init__(self):
            self.model = self._load_model()

        def _load_model(self):
            # Ensure the model path exists before loading
            if not os.path.exists(MODEL_PATH):
                print(f"Error: Model file not found at {MODEL_PATH}")
                return None
            try:
                # Load model with compile=False if optimizer state isn't needed for inference
                model = tf.keras.models.load_model(MODEL_PATH, compile=False)
                if debug:
                    print("\nModel loaded successfully. Input shape:", model.input_shape)
                return model
            except Exception as e:
                print(f"Model loading failed: {e}")
                return None

        def _extract_frames(self, video_path):
            frames = extract_video_frames(video_path, N_FRAMES, IMG_SIZE)
            if frames is None:
                print(f"Frame extraction returned None for {video_path}")
                return None

            if debug:
                blank_frames = np.all(frames == 0, axis=(1, 2, 3)).sum()
                if blank_frames > 0:
                    print(f"Warning: {blank_frames} blank frames detected")
                # Save a sample frame for debugging only if debug is True
                if frames.shape[0] > 0 and not np.all(frames[0] == 0): # Avoid saving blank frame
                    sample_frame = (frames[0] * 255).astype(np.uint8)
                    try:
                        os.makedirs(RESULT_PATH, exist_ok=True) # Ensure result path exists
                        debug_frame_path = os.path.join(RESULT_PATH, 'debug_frame.jpg')
                        cv2.imwrite(debug_frame_path, cv2.cvtColor(sample_frame, cv2.COLOR_RGB2BGR))
                        print(f"Debug frame saved to {debug_frame_path}")
                    except Exception as e:
                        print(f"Failed to save debug frame: {e}")
                else:
                     print("Skipping debug frame save (first frame blank or no frames).")


            return frames
```

Fight_detec_func.py fig1

```python
    def predict(self, video_path):
        if not os.path.exists(video_path):
            print(f"Error: Video not found at {video_path}")
            return "Error: Video not found", None

        try:
            frames = self._extract_frames(video_path)
            if frames is None:
                return "Error: Frame extraction failed", None

            if frames.shape[0] != N_FRAMES:
                # Pad with last frame or zeros if not enough frames were extracted
                print(f"Warning: Expected {N_FRAMES} frames, got {frames.shape[0]}. Padding...")
                if frames.shape[0] == 0: # No frames at all
                     frames = np.zeros((N_FRAMES, *IMG_SIZE, 3), dtype=np.float32)
                else: # Pad with the last available frame
                    padding_needed = N_FRAMES - frames.shape[0]
                    last_frame = frames[-1][np.newaxis, ...]
                    padding = np.repeat(last_frame, padding_needed, axis=0)
                    frames = np.concatenate((frames, padding), axis=0)
                print(f"Frames padded to shape: {frames.shape}")


            if np.all(frames == 0):
                # Check if all frames are actually blank (can happen with padding)
                print("Error: All frames are blank after processing/padding.")
                return "Error: All frames are blank", None

            # Perform prediction
            prediction = self.model.predict(frames[np.newaxis, ...], verbose=0)[0][0]
            # Determine result based on threshold
            threshold = 0.61 # Example threshold
            is_fight = prediction >= threshold
            result = "FIGHT" if is_fight else "NORMAL"

            # Calculate confidence (simple distance from threshold, scaled)
            # Adjust scaling factor (e.g., 150) and base (e.g., 50) as needed
            # Ensure confidence reflects certainty (higher for values far from threshold)
            if is_fight:
                confidence = min(max((prediction - threshold) * 150 + 50, 0), 100)
            else:
                confidence = min(max((threshold - prediction) * 150 + 50, 0), 100)


            result_string = f"{result} ({confidence:.1f}% confidence)"

            if debug:
                print(f"Raw Prediction Score: {prediction:.4f}")
                self._debug_visualization(frames, prediction, result_string, video_path)

            return result_string, float(prediction) # Return string and raw score

        except Exception as e:
            print(f"Prediction error: {str(e)}")
            # Consider logging the full traceback here in a real application
            # import traceback
            # print(traceback.format_exc())
            return f"Prediction error: {str(e)}", None
```

Fight_detec_func.py fig2

```python
        def _debug_visualization(self, frames, score, result, video_path):
            # This function will only run if debug=True is passed to fight_detec
            print(f"\n--- Debug Visualization ---")
            print(f"Prediction Score: {score:.4f}")
            print(f"Decision: {result}")

            # Avoid plotting if matplotlib is not available or causes issues in deployment
            try:
                import matplotlib.pyplot as plt
                plt.figure(figsize=(15, 5))
                num_frames_to_show = min(10, len(frames))
                for i in range(num_frames_to_show):
                    plt.subplot(2, 5, i+1)
                    # Ensure frame values are valid for imshow (0-1 or 0-255)
                    img_display = frames[i]
                    if np.max(img_display) <= 1.0: # Assuming normalized float [0,1]
                         img_display = (img_display * 255).astype(np.uint8)
                    else: # Assuming it might already be uint8 [0,255]
                         img_display = img_display.astype(np.uint8)

                    plt.imshow(img_display)
                    plt.title(f"Frame {i}\nMean: {frames[i].mean():.2f}") # Use original frame for mean
                    plt.axis('off')
                plt.suptitle(f"Video: {os.path.basename(video_path)}\nPrediction: {result} (Raw Score: {score:.4f})")
                plt.tight_layout(rect=[0, 0.03, 1, 0.95]) # Adjust layout

                # Save the visualization
                os.makedirs(RESULT_PATH, exist_ok=True) # Ensure result path exists again
                base_name = os.path.splitext(os.path.basename(video_path))[0]
                save_path = os.path.join(RESULT_PATH, f"{base_name}_prediction_result.png")
                plt.savefig(save_path)
                plt.close() # Close the plot to free memory
                print(f"Debug visualization saved to: {save_path}")
            except ImportError:
                print("Matplotlib not found. Skipping debug visualization plot.")
            except Exception as e:
                print(f"Error during debug visualization: {e}")
            print("--- End Debug Visualization ---")


    # --- Main function logic ---
    detector = FightDetector()
    if detector.model is None:
        # Model loading failed, return error
        return "Error: Model loading failed", None

    # Call the predict method
    result_str, prediction_score = detector.predict(video_path)
    return result_str, prediction_score
```

Fight_detec_func.py fig3

```python

import cv2
import numpy as np
import os
from ultralytics import YOLO
import time
from typing import Tuple, Set, List

def detection(path: str) -> Tuple[Set[str], str]:
    """
    Detects and tracks objects in a video using YOLOv8 model, saving an annotated output video.

    Args:
        path (str): Path to the input video file. Supports common video formats (mp4, avi, etc.)

    Returns:
        Tuple[Set[str], str]:
            - Set of unique detected object labels (e.g., {'Gun', 'Knife'})
            - Path to the output annotated video with detection boxes and tracking IDs

    Raises:
        FileNotFoundError: If input video doesn't exist
        ValueError: If video cannot be opened/processed or output dir cannot be created
    """

    # Validate input file exists
    if not os.path.exists(path):
        raise FileNotFoundError(f"Video file not found: {path}")

    # --- Model Loading ---
    # Construct path relative to this script file
    model_path = os.path.join(os.path.dirname(__file__), "best.pt")
    if not os.path.exists(model_path):
         raise FileNotFoundError(f"YOLO model file not found at: {model_path}")
    try:
        model = YOLO(model_path)
        class_names = model.names  # Get class label mappings
        print(f"[INFO] YOLO model loaded from {model_path}. Class names: {class_names}")
    except Exception as e:
        raise ValueError(f"Failed to load YOLO model: {e}")


    # --- Output Path Setup ---
    input_video_name = os.path.basename(path)
    base_name = os.path.splitext(input_video_name)[0]
    # Sanitize basename to prevent issues with weird characters in filenames
    safe_base_name = "".join(c if c.isalnum() or c in ('-', '_') else '_' for c in base_name)

    # Define output directory relative to this script
    # In HF Spaces, this will be inside the container's file system
    output_dir = os.path.join(os.path.dirname(__file__), "results")
    temp_output_name = f"{safe_base_name}_output_temp.mp4"

    try:
        os.makedirs(output_dir, exist_ok=True)  # Create output dir if needed
        if not os.path.isdir(output_dir):
             raise ValueError(f"Path exists but is not a directory: {output_dir}")
    except OSError as e:
        raise ValueError(f"Failed to create or access output directory '{output_dir}': {e}")

    temp_output_path = os.path.join(output_dir, temp_output_name)
    print(f"[INFO] Temporary output will be saved to: {temp_output_path}")

```

object_detect_yolo.py fig1

```python
# --- Video Processing Setup ---
cap = cv2.VideoCapture(path)
if not cap.isOpened():
    raise ValueError(f"Failed to open video file: {path}")

# Get video properties for output writer
# Use source FPS if available and reasonable, otherwise default to 30
source_fps = cap.get(cv2.CAP_PROP_FPS)
output_fps = source_fps if 10 <= source_fps <= 60 else 30.0

# Process at a fixed resolution for consistency or use source resolution
# Using fixed 640x640 as potentially used during training/fine-tuning
frame_width, frame_height = 640, 640
# OR use source resolution (might require adjusting YOLO parameters if model expects specific size)
# frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
# frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

try:
    out = cv2.VideoWriter(
        temp_output_path,
        cv2.VideoWriter_fourcc(*'mp4v'),  # Use MP4 codec
        output_fps,
        (frame_width, frame_height)
    )
    if not out.isOpened():
        # Attempt alternative codec if mp4v fails (less common)
        print("[WARNING] mp4v codec failed, trying avc1...")
        out = cv2.VideoWriter(
            temp_output_path,
            cv2.VideoWriter_fourcc(*'avc1'),
            output_fps,
            (frame_width, frame_height)
        )
        if not out.isOpened():
             raise ValueError("Failed to initialize VideoWriter with mp4v or avc1 codec.")

except Exception as e:
    cap.release() # Release capture device before raising
    raise ValueError(f"Failed to create VideoWriter: {e}")


# --- Main Processing Loop ---
detected_classes: List[str] = []  # Track detected object class names
start = time.time()
frame_count = 0
print(f"[INFO] Video processing started...")

while True:
    ret, frame = cap.read()
    if not ret:  # End of video or read error
        break

    frame_count += 1
    # Resize frame BEFORE passing to model
    resized_frame = cv2.resize(frame, (frame_width, frame_height))

    try:
        # Run YOLOv8 detection and tracking on the resized frame
        results = model.track(
            source=resized_frame, # Use resized frame
            conf=0.7,             # Confidence threshold
            persist=True,         # Maintain track IDs across frames
            verbose=False         # Suppress Ultralytics console output per frame
        )
```

object_detect_yolo.py fig2

```python
            # Check if results are valid and contain boxes
            if results and results[0] and results[0].boxes:
                # Annotate the RESIZED frame with bounding boxes and track IDs
                annotated_frame = results[0].plot() # plot() draws on the source image

                # Record detected class names for this frame
                for box in results[0].boxes:
                    if box.cls is not None: # Check if class ID is present
                        cls_id = int(box.cls[0]) # Get class index
                        if 0 <= cls_id < len(class_names):
                            detected_classes.append(class_names[cls_id])
                        else:
                            print(f"[WARNING] Detected unknown class ID: {cls_id}")
            else:
                 # If no detections, use the original resized frame for the output video
                 annotated_frame = resized_frame

            # Write the (potentially annotated) frame to the output video
            out.write(annotated_frame)

        except Exception as e:
             print(f"[ERROR] Error processing frame {frame_count}: {e}")
             # Write the unannotated frame to keep video timing consistent
             out.write(resized_frame)


    # --- Clean Up ---
    end = time.time()
    print(f"[INFO] Video processing finished. Processed {frame_count} frames.")
    print(f"[INFO] Total processing time: {end - start:.2f} seconds")
    cap.release()
    out.release()
    cv2.destroyAllWindows() # Close any OpenCV windows if they were opened


    # --- Final Output Renaming ---
    unique_detected_labels = set(detected_classes)
    # Create a short string from labels for the filename
    labels_str = "_".join(sorted(list(unique_detected_labels))).replace(" ", "_")
    # Limit length to avoid overly long filenames
    max_label_len = 50
    if len(labels_str) > max_label_len:
        labels_str = labels_str[:max_label_len] + "_etc"
    if not labels_str: # Handle case where nothing was detected
        labels_str = "no_detections"

    final_output_name = f"{safe_base_name}_{labels_str}_output.mp4"
    final_output_path = os.path.join(output_dir, final_output_name)

    # Ensure final path doesn't already exist (rename might fail otherwise)
    if os.path.exists(final_output_path):
        os.remove(final_output_path)

    try:
        # Rename the temporary file to the final name
        os.rename(temp_output_path, final_output_path)
        print(f"[INFO] Detected object labels: {unique_detected_labels}")
        print(f"[INFO] Annotated video saved successfully at: {final_output_path}")
    except OSError as e:
        print(f"[ERROR] Failed to rename {temp_output_path} to {final_output_path}: {e}")
        # Fallback: return the temp path if rename fails but file exists
        if os.path.exists(temp_output_path):
             print(f"[WARNING] Returning path to temporary file: {temp_output_path}")
             return unique_detected_labels, temp_output_path
        else:
             raise ValueError(f"Output video generation failed. No output file found.")


    return unique_detected_labels, final_output_path
```

object_detect_yolo.py fig3

```python
import cv2
import numpy as np
import os
from ultralytics import YOLO
import time
import tensorflow as tf
from frame_slicer import extract_video_frames
import matplotlib.pyplot as plt

from Fight_detec_func import fight_detec
from objec_detect_yolo import detection



# Entry point
path0 = input("Enter the local path : ")
path = path0.strip('"')  # Remove extra quotes if copied from Windows
print(f"[INFO] Loading video: {path}")

fight_detec(path)
detection(path)

```

full_project.py fig1

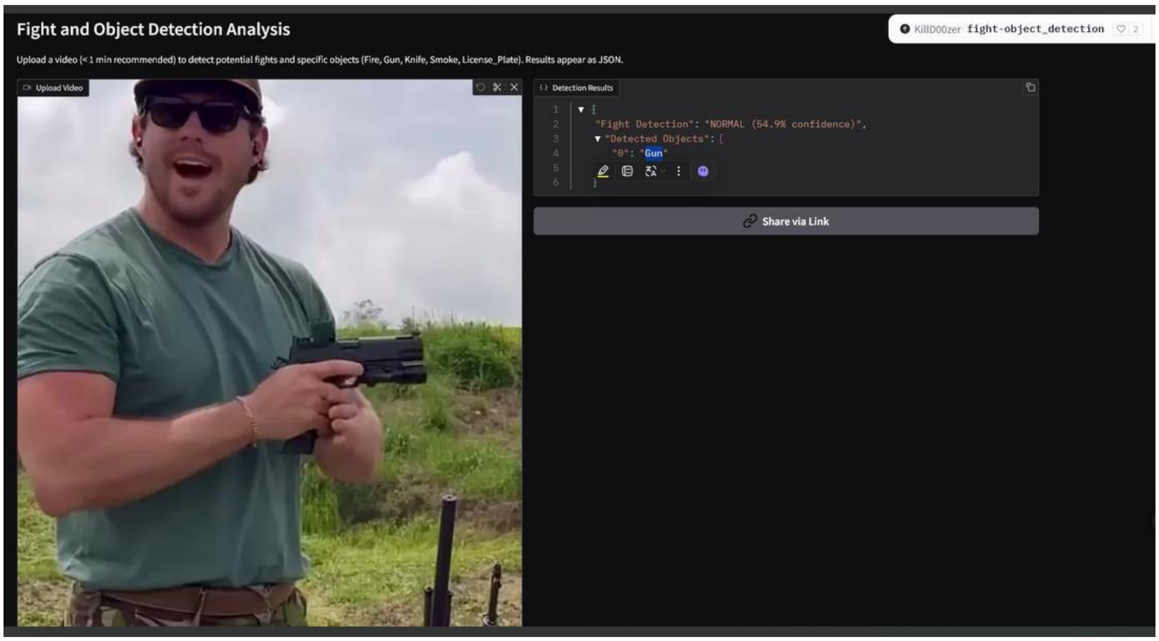
11.3. Appendix C: Environment Setup

- *pip install -r requirements.txt*
- *python app.py*

```
gradio>=3.0
tensorflow>=2.10 # Or specific version if needed, e.g., tensorflow==2.16.1
opencv-python>=4.6
ultralytics>=8.0
numpy>=1.22
matplotlib>=3.6 # Needed for debug plots in Fight_detec_func if debug=True
```

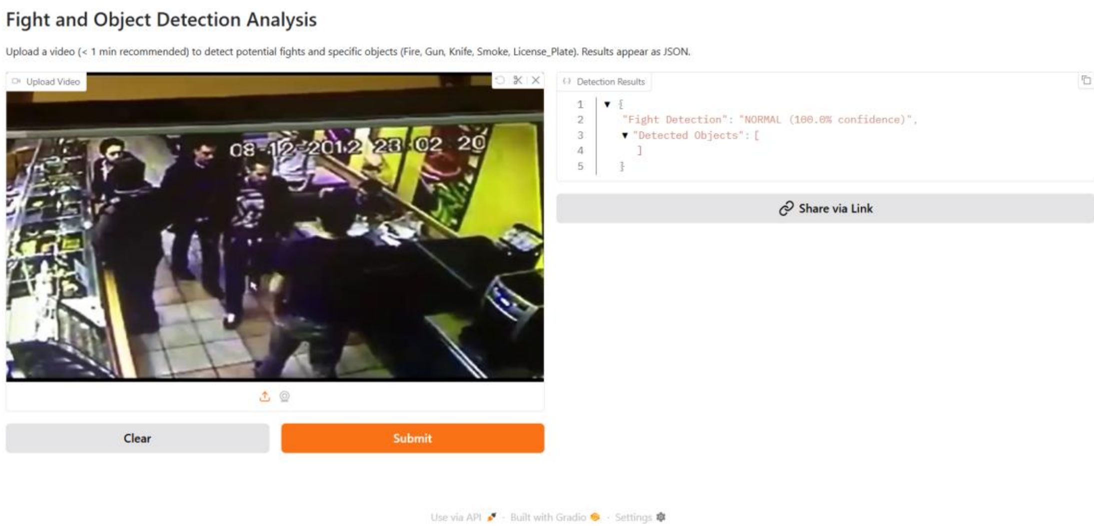
Requirements installation fig1

11.4. Appendix D: Model Output Samples



Model output fig1

Model output fig2



Model output fig3

11.5. Appendix E: GitHub/Hugging Face Link:

https://huggingface.co/spaces/KillD00zer/fight-object_detection