**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 | 1. Eng. Mohamed Mostafa El-Sayed 2. Eng. Karim Mohamed Hafez 3. 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**

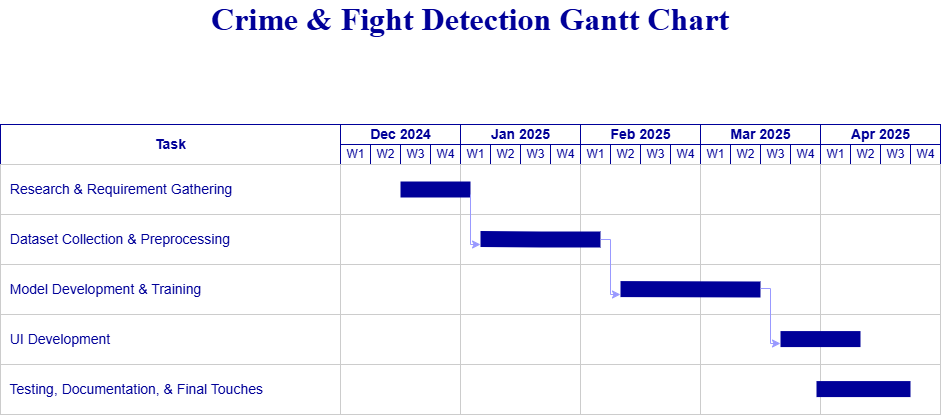
We sincerely thank the DEPI program for offering a platform that empowers students to apply AI and data science skills to real-world problems. We express our gratitude to our supervisors, Eng. Mariam Elsaee and Dr. Nesma Ibrahim, whose invaluable feedback and encouragement guided us throughout the project. 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. Project Proposal
   2. Project Plan (Timeline, Gantt Chart)
   3. Task Assignment & Roles
   4. Risk Assessment & Mitigation Plan
   5. Key Performance Indicators (KPIs)
2. **Literature Review / Related Work**
   1. Existing Systems
   2. Our Competitive Advantages
   3. Feedback & Evaluation
   4. Suggested Improvements
   5. Final Grading Criteria
3. **Requirements Gathering**
   1. Stakeholder Analysis
   2. User Stories & Use Cases
   3. Functional Requirements
   4. Non-Functional Requirements
4. **System Analysis & Design**
   1. Problem Statement & Objectives
   2. Objectives
   3. Use Case Diagram & Descriptions
   4. Software Architecture
   5. Data Flow & Behavior
   6. UI/UX Design (Wireframes, Mockups)
   7. System Deployment & Integration
5. **Implementation (Source Code & Execution)**
   1. Source Code Structure
   2. Version Control (GitHub, Branching Strategy)
   3. Deployment & Execution (README Guide)
6. **Testing & Quality Assurance**
   1. Test Cases & Test Plan
   2. Automated Testing
   3. Bug Reports & Resolutions
7. **Final Presentation & Reports**
   1. User Manual
   2. Technical Documentation
   3. Project Presentation (PPT/PDF)
8. **Conclusion & Future Work**
   1. Summary of Achievements
   2. Limitations
   3. Future Enhancements
9. **List of Abbreviations**
10. **References**
11. **Appendices**
    1. Appendix A: Screenshots of Dashboard
    2. Appendix B: Code Snippets
    3. Appendix C: Environment Setup
    4. Appendix D: Model Output Samples
    5. Appendix E: GitHub Repository Link
12. **Project Planning & Management**
    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. **Project Plan (Timeline)**



* 1. **Task Assignment & Roles**

|  |  |
| --- | --- |
| ****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, Analysis** |
| **Eng. Karim Mohamed Hafez** | **Data Collection, Diagrams** |
| **Eng. Rana Hassan Badrawi** | **Data Collection, Diagrams** |

* + 1. **Team Structure:** Each team member contributed according to theirexpertise:
    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. **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. **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.

1. **Literature Review / Related Work**
   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](https://ieeexplore.ieee.org/document/9092170) *(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](https://link.springer.com/article/10.1007/s11042-023-16252-6) *(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](https://arxiv.org/abs/2103.15691) *(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](https://www.researchgate.net/publication/343709938) *(Alaa Senjab)* | 2020 | * Simple PyQt5 interface. * Real-time inference. | * No action recognition. * Lacks behavioral context. |
| [Weapon Detection in Videos Using YOLOv5](https://ieeexplore.ieee.org/document/9586110) *(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](https://www.sciencedirect.com/science/article/pii/S1110866520300412) *(Mostafa Mohamed Moaaz)* | 2020 | * Good accuracy on standard datasets. * Uses temporal features. | * Architecture unspecified. * No real-time deployment focus. |

* 1. Our Competitive Advantages
     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.
  + 1. 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).
  + 1. 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
  + 1. 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.
  + 1. 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.
  + 1. 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.
  + 1. 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.
  + 1. 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.
  1. 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.

* 1. Suggested Improvements
* Expand dataset to include real CCTV footage.
* Incorporate audio input for scream/gunshot detection.
* Develop mobile version or lightweight edge-ready deployment.  
  1. 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

1. **Requirements Gathering**

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

* 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

* 1. **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.

* 1. **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
  1. **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)

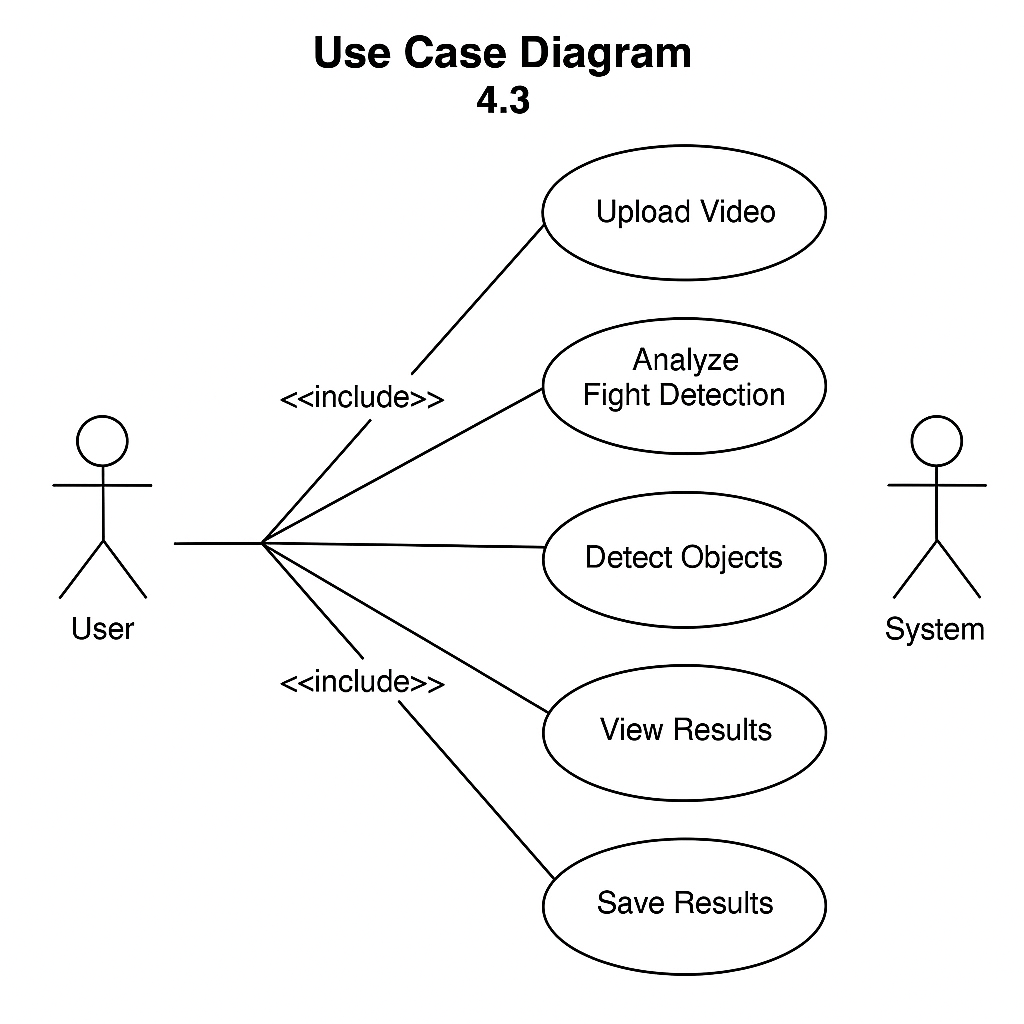
1. **System Analysis & Design**

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

* 1. **Problem Statement & Objectives**

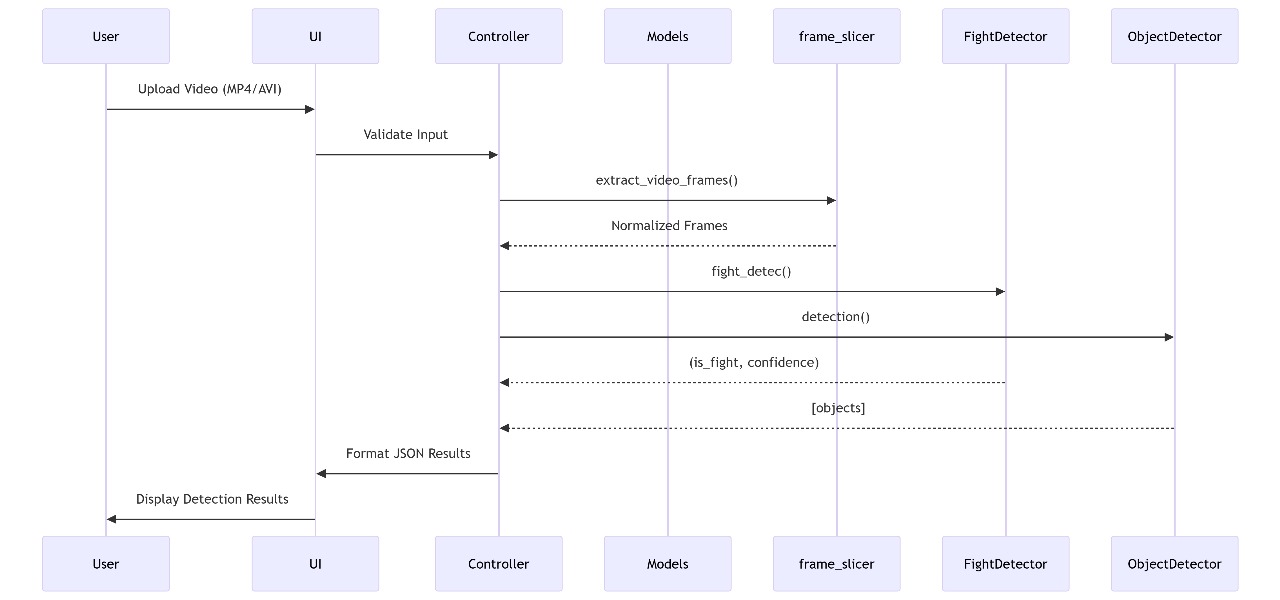
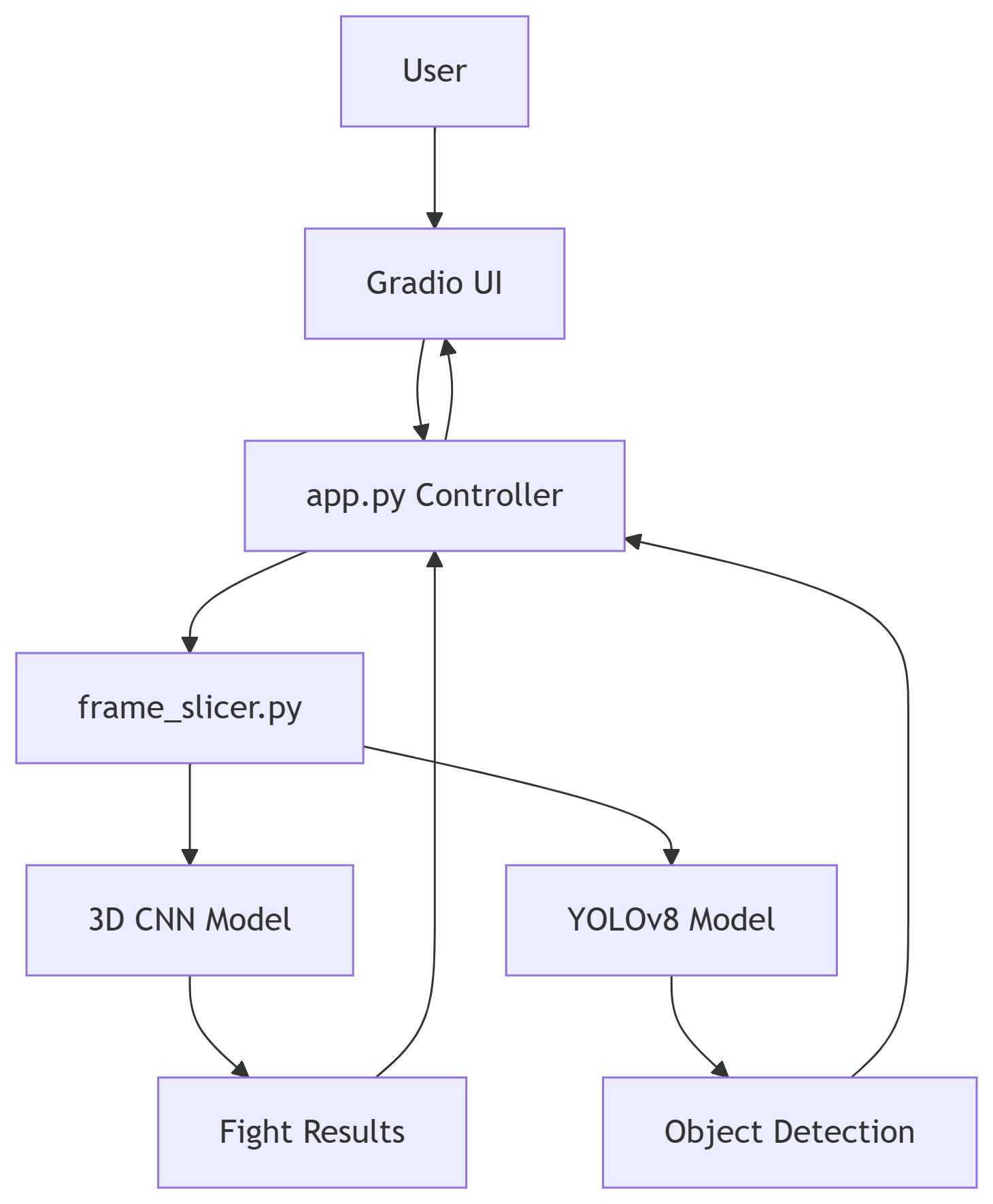
Manual monitoring of surveillance video is inefficient. Automating the detection of aggressive actions and weapons in video feeds helps prevent violence and enables quicker responses.

* 1. **Objectives**
* Build dual detection systems
* Ensure modular codebase
* Provide easy-to-use frontend
* Enable reproducible results
  1. **Use Case Diagram & Descriptions**
* Actors: User, System
* Actions: Upload Video → Preprocess → Run Detection → Display Result



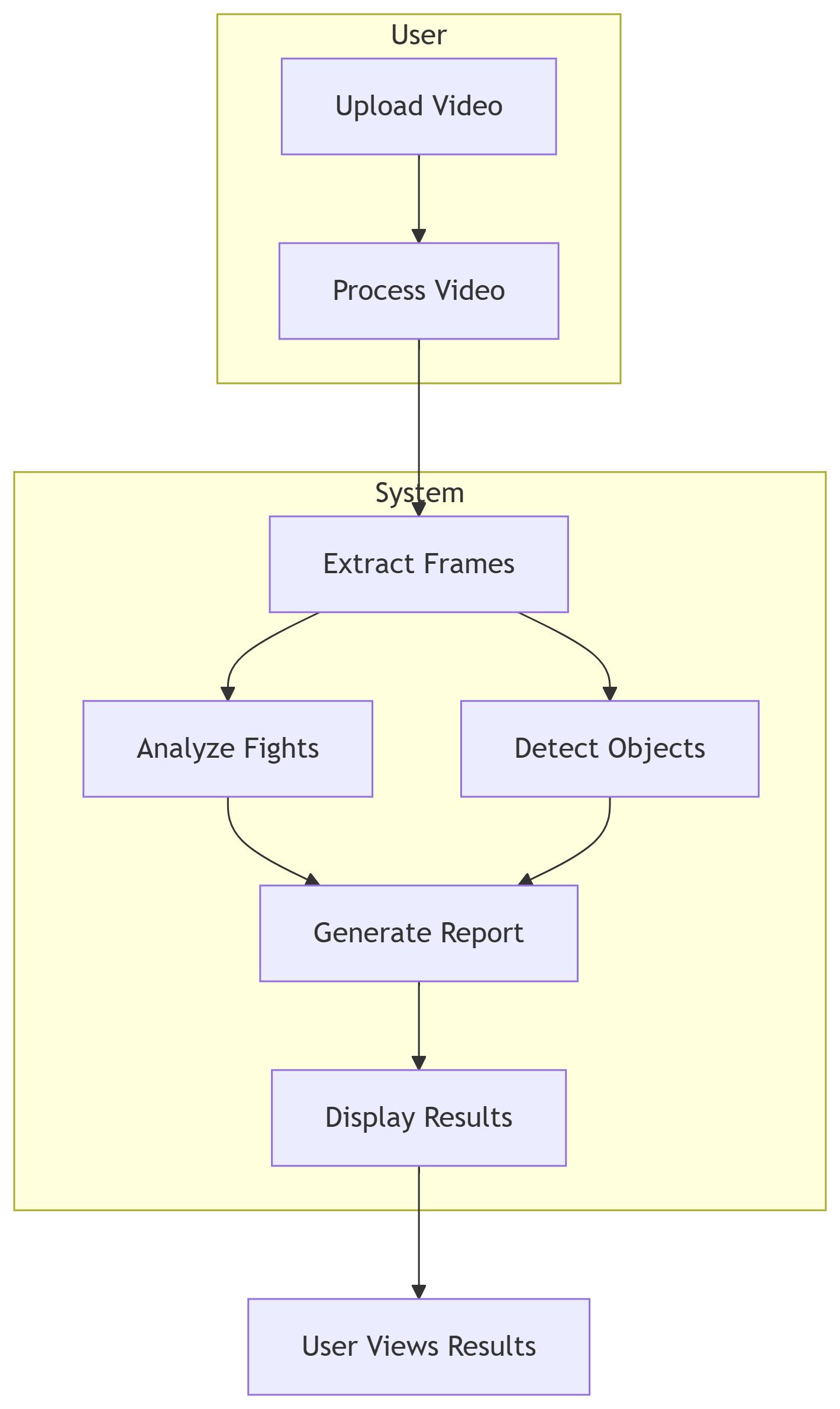
Use case fig

* 1. **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

* 1. **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

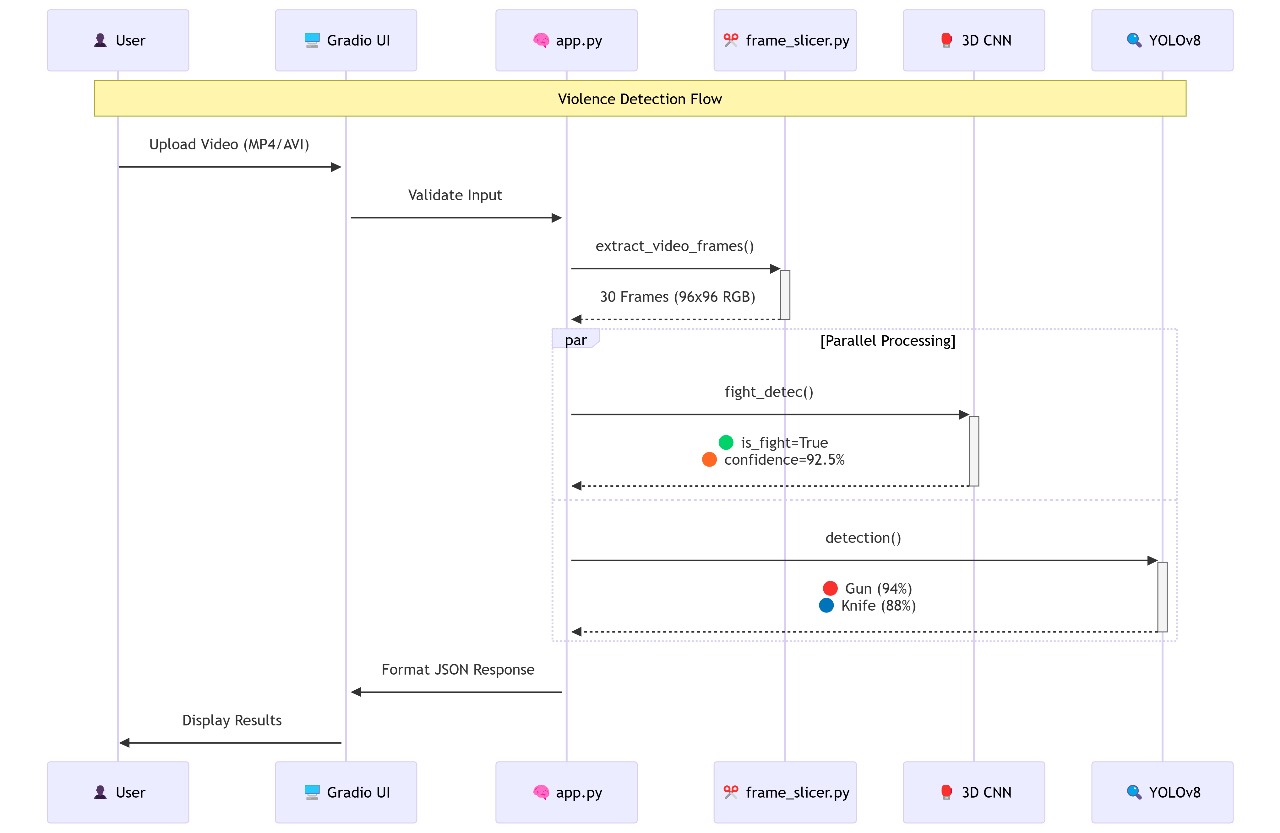


Dataflow fig

* 1. **UI/UX Design**

Gradio components:

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



UI & UX fig

* 1. **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

1. **Implementation (Source Code & Execution)**
   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)

* 1. **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.
  1. **Deployment & Execution (README Guide)**
     1. **Running Locally**
        1. Clone the repository.
        2. Install requirements: *pip install -r requirements.txt*
        3. Run the app: *python app.py*
        4. **Environment Requirements**
* Python 3.10+
* gradio>=3.0
* tensorflow>=2.10
* opencv-python>=4.6
* ultralytics>=8.0
* numpy>=1.22
* matplotlib>=3.6
  + 1. **Online Hosting**
* Deployed on Hugging Face Spaces: <https://huggingface.co/spaces/KillD00zer/fight-object_detection>

1. **Testing & Quality Assurance**
   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 |

* 1. **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
  1. **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 |

1. **Final Presentation & Reports**
   1. **User Manual**

* Upload video → Wait for processing → Watch output → Click download
* No account/login needed
* Recommended video duration: < 30s for best performance
  1. **Technical Documentation**

Includes:

* Model architecture diagrams
* Annotated code
* API interface explanation
* Data schema of prediction pipeline
  1. **Project Presentation (PPT)**

Key slides:

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

1. **Conclusion & Future Work**

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

* 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
  1. **Limitations**
* Real-time video not supported
* False positives possible in low-light
* High resource usage (especially TensorFlow inference)
  1. **Future Enhancements**
* Add real-time webcam support
* Convert models to TensorRT for speed
* Add sound detection (gunshots, screams)
* Add mobile support using TensorFlow Lite

1. **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. |

1. **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

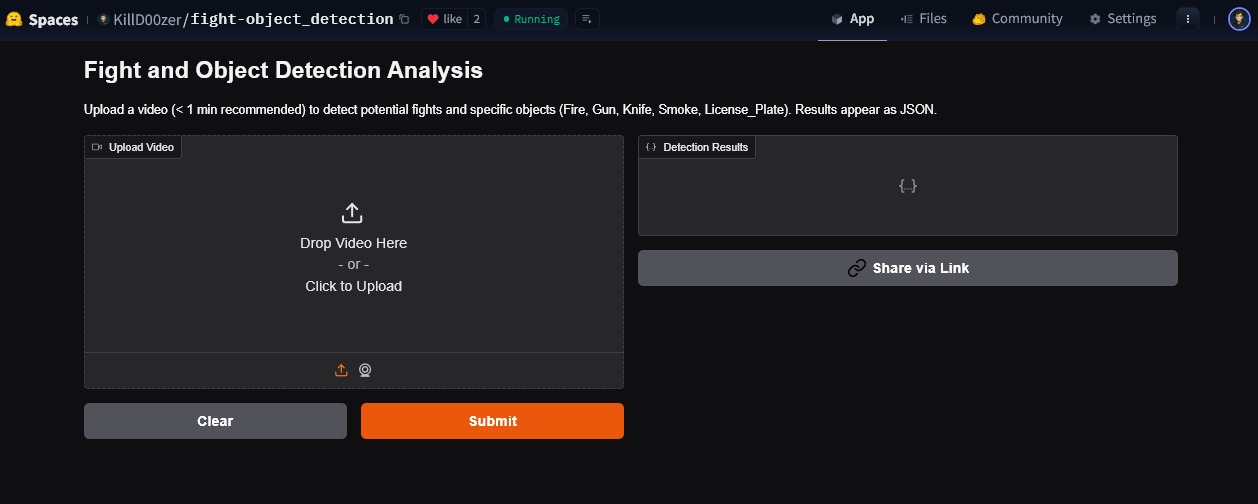
<https://ieeexplore.ieee.org/document/9586110>

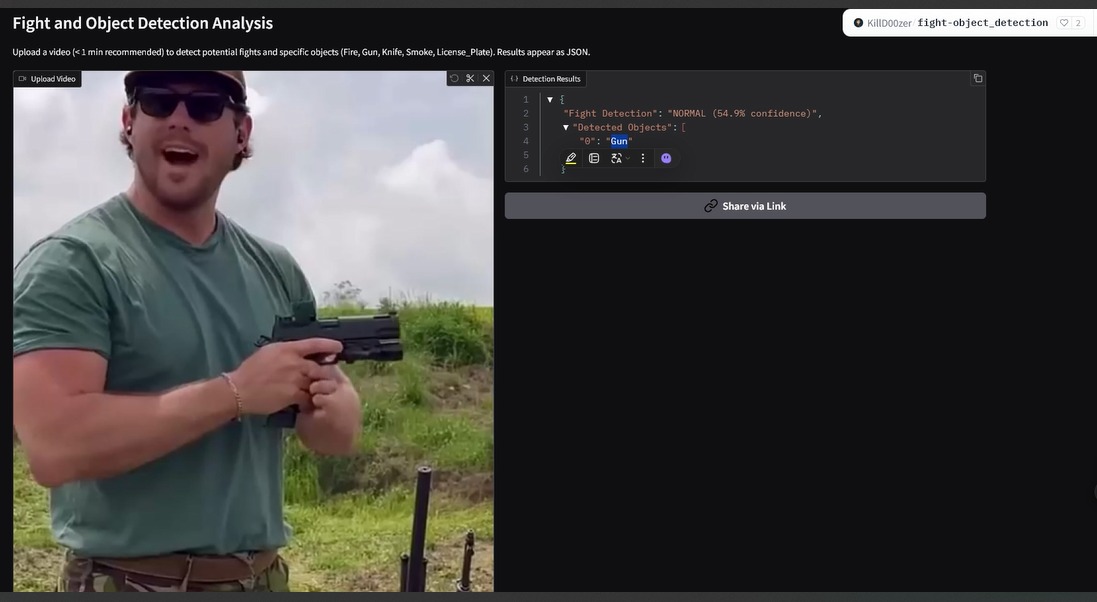
* Violence Detection in Surveillance Videos Using Deep Learning

<https://www.sciencedirect.com/science/article/pii/S1110866520300412>

1. **Appendices**

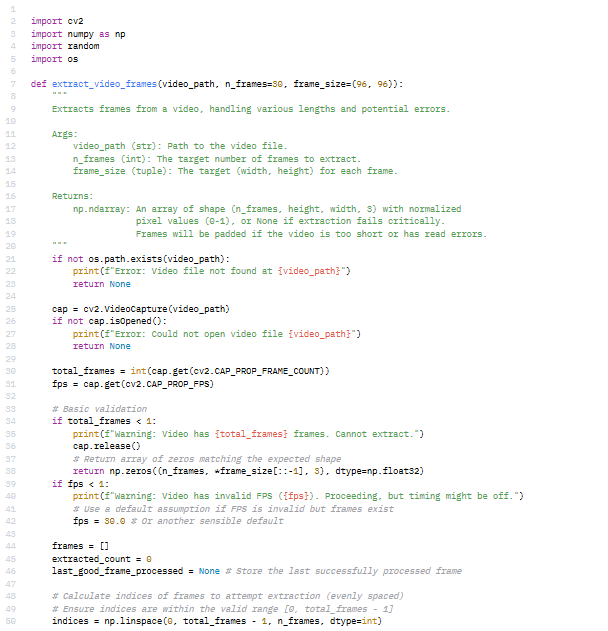
This section includes **supplementary materials** that support the main documentation.

* 1. **Appendix A: Screenshots of Dashboard**

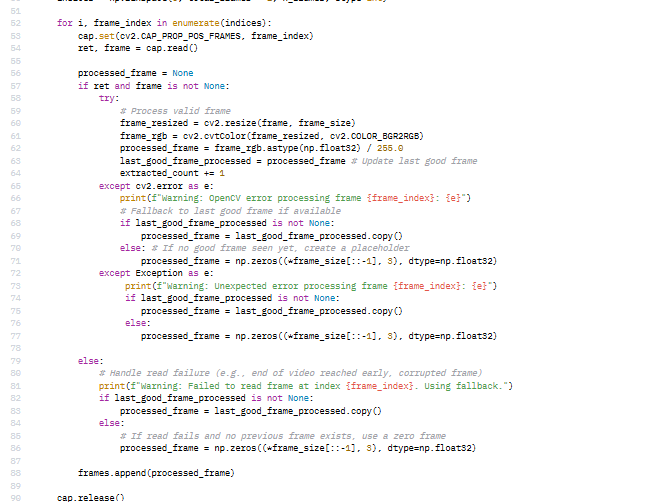
****huggingface fig 1huggingface fig 2

* 1. **Appendix B: Code Snippets**

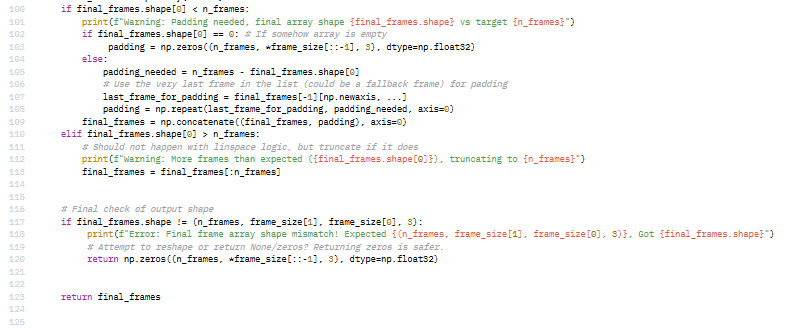
app.py fig

****

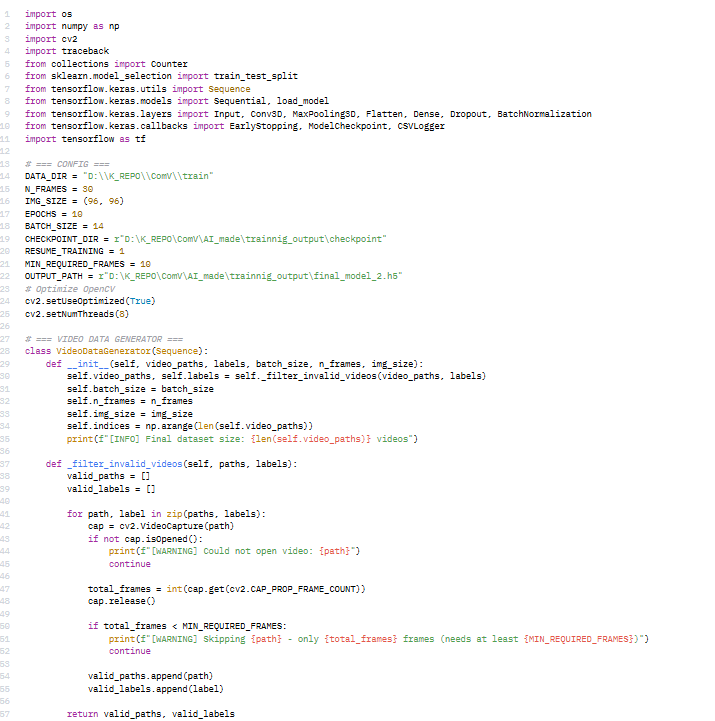
frame\_slicer fig1

****

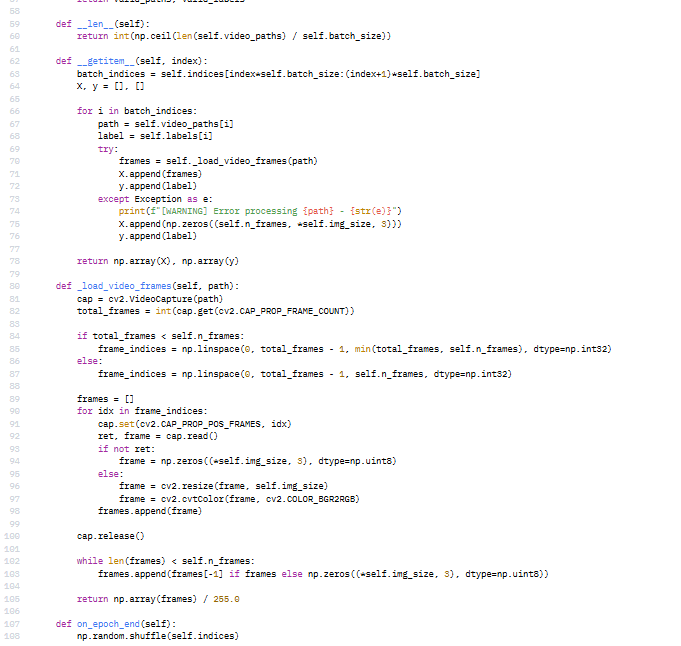
frame\_slicer fig2



frame\_slicer fig3



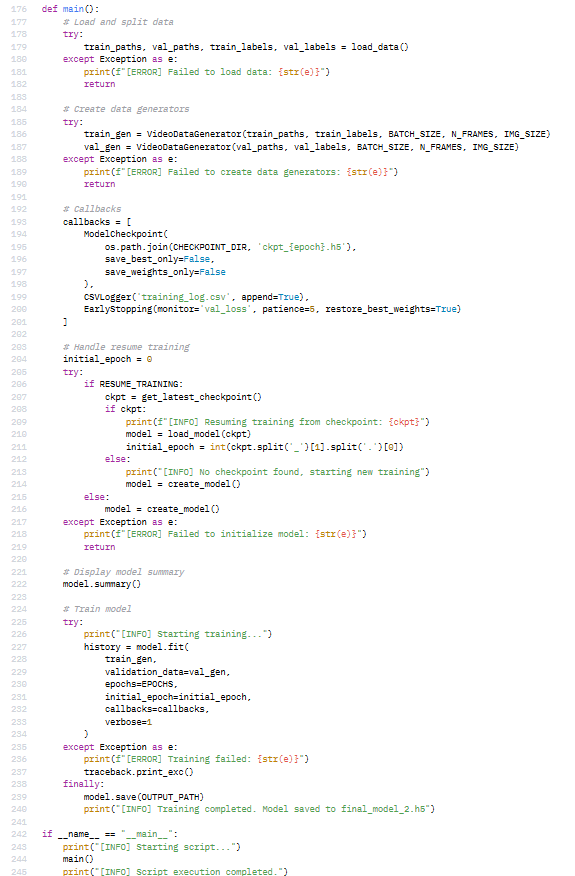
Training.py fig1



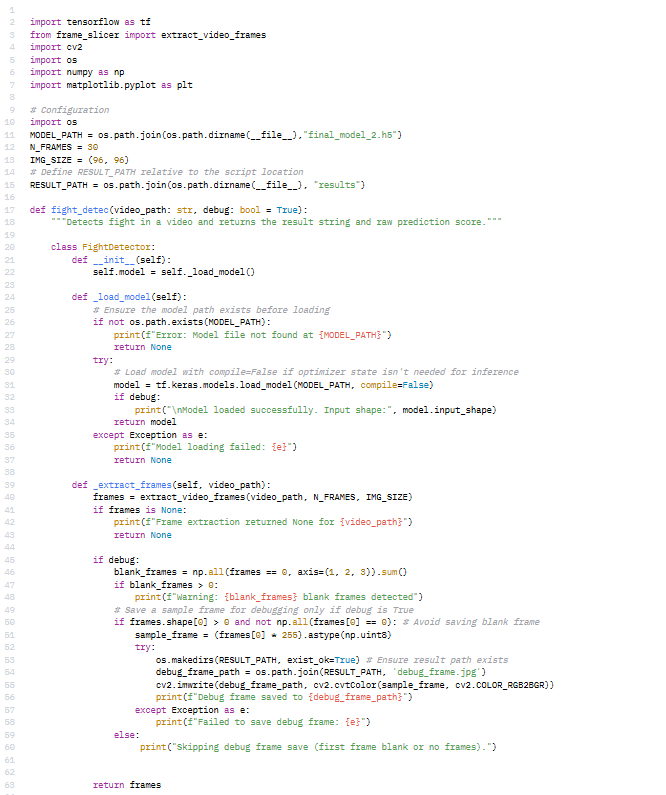
Training.py fig2

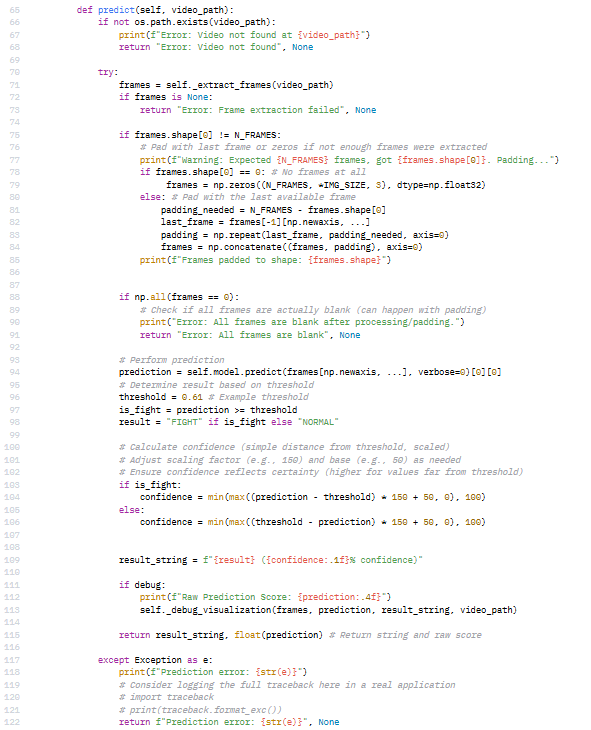


Training.py fig3



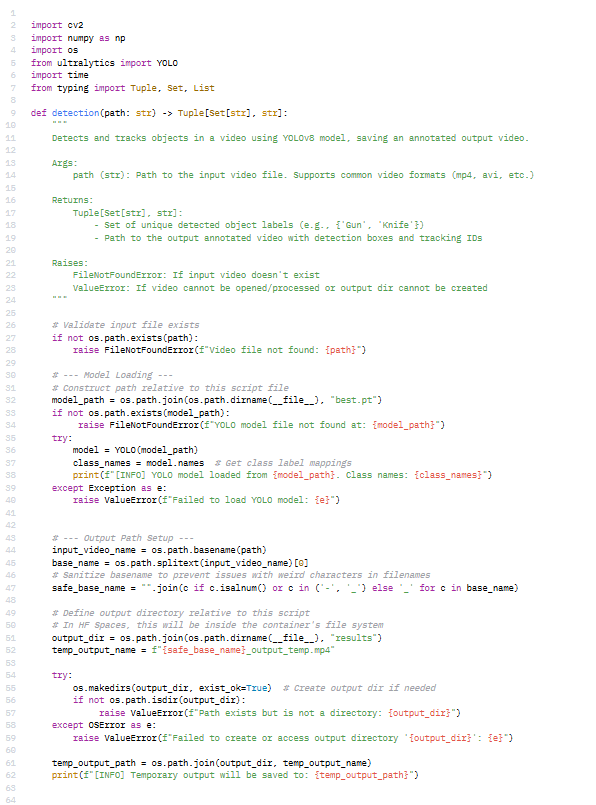
Training.py fig4

Fight\_detec\_func.py fig1

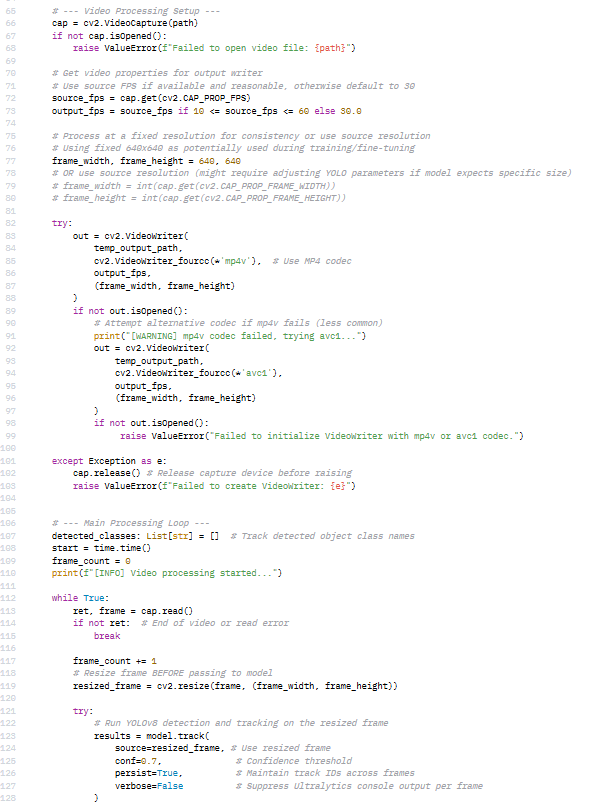
Fight\_detec\_func.py fig2



Fight\_detec\_func.py fig3



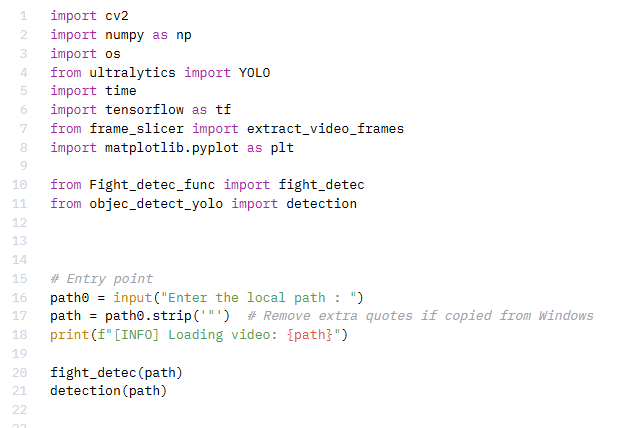
object\_detect\_yolo.py fig1



object\_detect\_yolo.py fig2

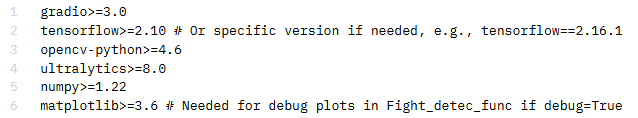


object\_detect\_yolo.py fig3



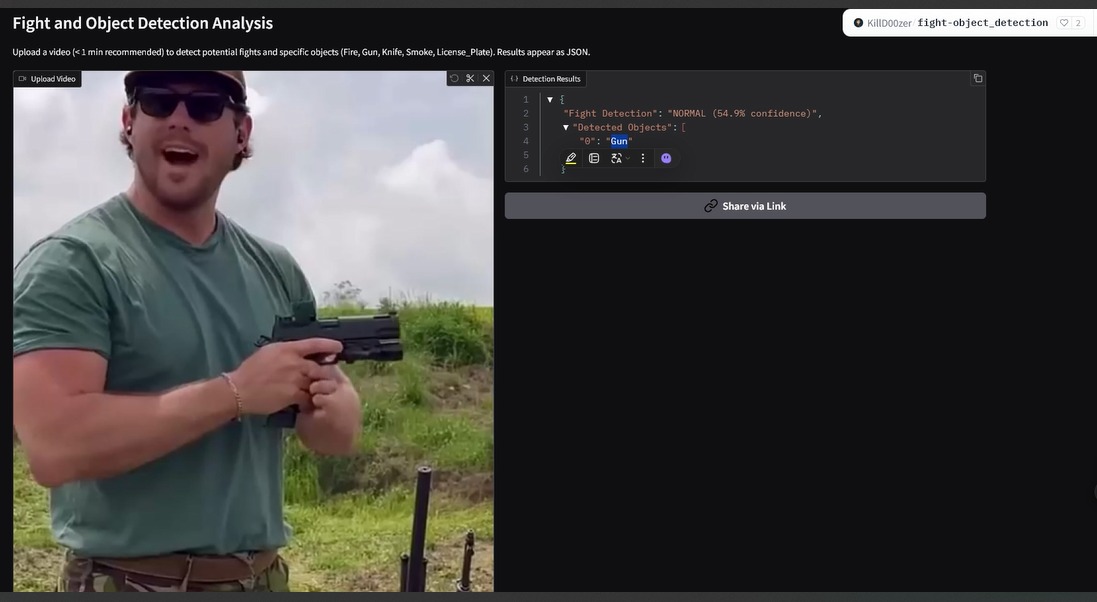
full\_project.py fig1

* 1. **Appendix C: Environment Setup**
* *pip install -r requirements.txt*
* *python app.py*

**

Requirements installation fig1

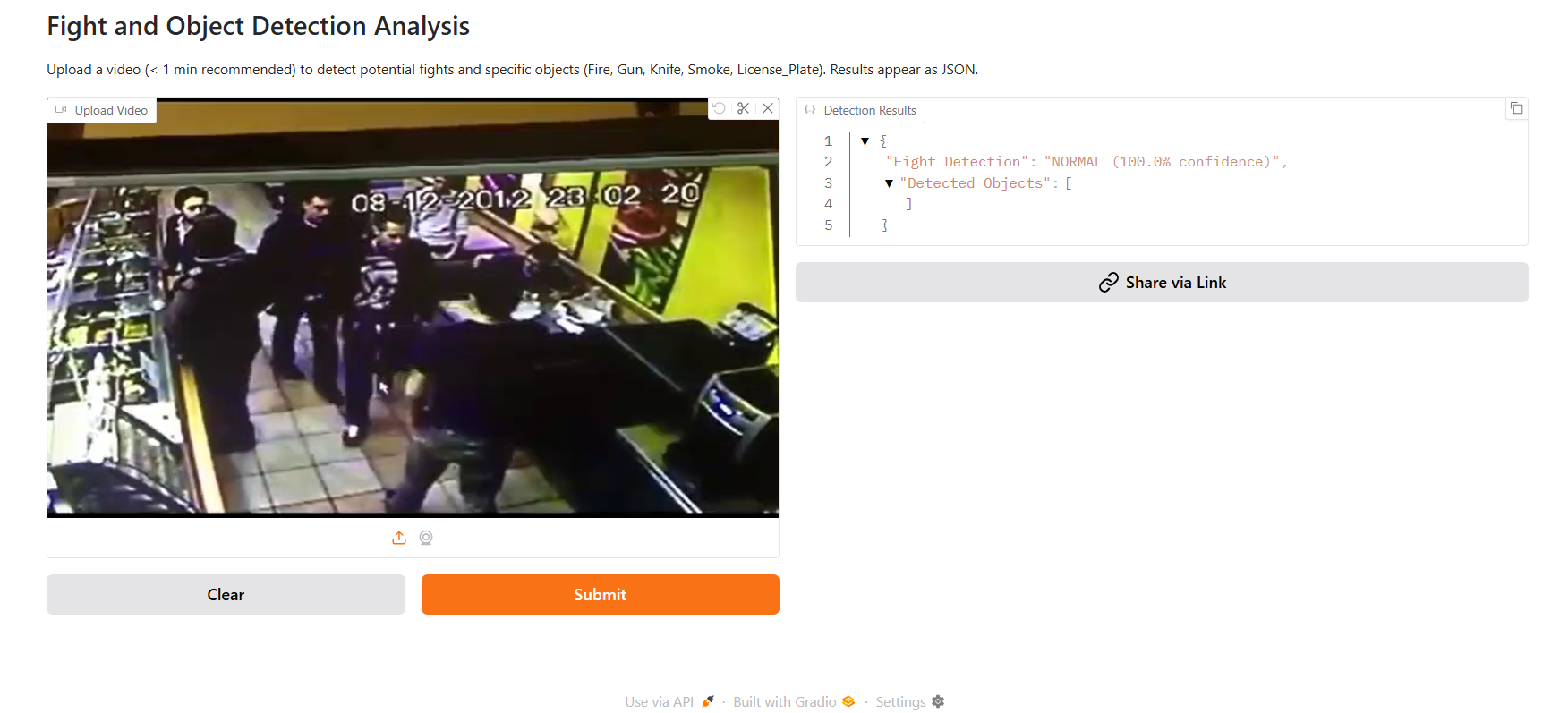
* 1. **Appendix D: Model Output Samples**

****

Model output fig1

****

Model output fig2

****

Model output fig3

* 1. **Appendix E: GitHub/Hugging Face Link:** <https://huggingface.co/spaces/KillD00zer/fight-object_detection>