INTERNSHIP NOTES

Created	@October 22, 2025 11:04 PM
Class	IIITH
Event	

The core concepts and functions used in the Python code blocks, focusing on the `ultralytics` library.

Task1: Basic image segmentation and classification

TASK 1: Code

Explanation of the YOLOv8 Python Code

The core of the procedure relies on the **`ultralytics`** Python package, which provides a simple, high-level API for using the powerful YOLO models.

Core Components

The two essential lines that appear in most steps are:

1. `from ultralytics import YOLO`
2. `model = YOLO("model_file.pt")`

- **`from ultralytics import YOLO`**: This imports the main **`YOLO`** **class** from the installed library. This class is the central object you'll interact with for loading models, running predictions, and accessing results.
- **`model = YOLO("yolov8n.pt")`**: This **loads the pre-trained model**.
  - **`YOLO()`**: Creates an instance of the YOLO object.

- `"yolov8n.pt"`: This is the model file. YOLOv8 comes in different sizes (nano 'n', small 's', medium 'm', etc.). The 'n' stands for **nano**, the smallest and fastest version, which is great for quick testing. The file is downloaded automatically the first time you run the script.
  - For **detection**, you use `yolov8n.pt`.
  - For **segmentation** (Step 6), you use `yolov8n-seg.pt`, which is a segmentation-specific version.

Explanation of the `model.predict()` Function

The `model.predict()` function is the single, unified way to run the model on data (images, videos, or streams). Here is a breakdown of the arguments used:

`results = model.predict(...)`

Argument	Example Value	Purpose
`source`	`"https://..."` or `r"C:\..."`	**Mandatory**: Specifies the input to process. This can be a local **file path** (image/video), a **folder path** (to process all images), or a **URL**.
`show`	`True` or `False`	If `True`, a window will pop up to display the results in real-time as the model processes the image/video. Useful for live preview.
`save`	`True` or `False`	If `True`, the annotated image (with bounding boxes or masks) is saved to the output directory.
`project`	`r"C:\Users\...\YOLO_Outputs"`	**Optional**: The **root directory** where all prediction runs are saved. If not specified, it defaults to a folder like `runs/detect` or `runs/segment` in your current working directory.
`name`	`"Paypal_Mafia_2014"`	**Optional**: The **specific folder name** for this particular run, created inside the `project` directory. This keeps your experiments organized.
`classes`	`[0]`	**Optional/Crucial for Filtering**: This is a **list of class IDs** you want the model to process or

		display. It's used to filter results.

- YOLO is trained on the **COCO dataset**, where **class ID `0` always corresponds to a "person"**.
  * By setting `classes=[0]`, you instruct the model to only output results (bounding boxes or segmentation masks) for detected persons, ignoring all other objects (cars, bikes, animals, etc.).

The `results` Variable

The line `results = model.predict(...)` captures the output.

- The `results` variable is a list of `Results` **objects**. For a single image prediction, it usually contains one `Results` object.
- This object contains all the detailed, low-level information about the prediction, such as:
  - The coordinates of all bounding boxes.
  - The confidence score for each detection.
  - The segmentation masks (for segmentation models).
  - The detected class IDs.

The Key Difference: Detection vs. Segmentation

The main difference between Step 5 and Step 6 is which model is loaded:

Feature	Step 5: Object Detection	Step 6: Instance Segmentation
Goal	Find objects and draw a **bounding box** around them.	Find objects and generate a pixel-level **mask** for the exact shape of the object.
Model File	`yolov8n.pt`	`yolov8n-seg.pt`
Output	Rectangles/Boxes around people.	**Pixel-perfect shapes** (masks) highlighting the entire area of each person.

By using the `yolov8n-seg.pt` model and the `classes=[0]` filter, you can run an , person-only instance segmentation task on your local image.

Ultralytics YOLOv8 library to perform two distinct tasks on a **folder of images**

Task 2 Code (25-10-2025)

This code block demonstrates how to use the **Ultralytics YOLOv8 library** to perform two distinct tasks on a **folder of images**

Precision, Recall, and F1 Score are essential metrics for evaluating how well a model performs **classification** or **detection** tasks. They are all derived from the fundamental counts of correct and incorrect predictions: **True Positives (TP), False Positives (FP), and False Negatives (FN)**.

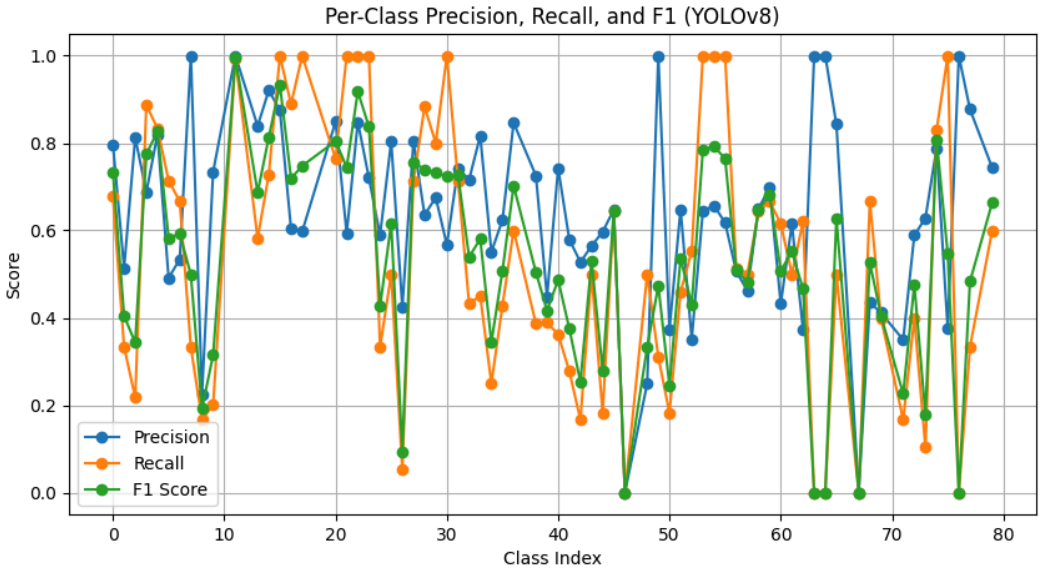
Term	Definition	Outcome
True Positive (TP)	The model correctly identifies an object.	A person is present, and the model drew a bounding box around them. ✅
False Positive (FP)	The model incorrectly identifies something as an object.	The model drew a bounding box around a tree, claiming it's a person. ❌
False Negative (FN)	The model misses an object that is present.	A person is present, but the model failed to draw a bounding box. 🚫

(Note: True Negative (TN) is typically ignored in object detection because the vast majority of an image is "negative" (empty space).) **because these metrics are designed to focus on the performance of the positive class**

`metrics = model_detect.val(data='coco128.yaml')`: The `.val()` function runs the loaded model against a **validation dataset** (here, the small COCO128 dataset is used as an example). This function calculates performance metrics.

`metrics.box.p` / `metrics.box.r` / `metrics.box.f1`: These access the **Per-Class Metrics**.
• **Precision ($\mathbf{P}$)**: Of all objects the model *detected* for a class, how

many were correct? (High P = low false positives).
• **Recall ($\mathbf{R}$)**: Of all actual objects in a class, how many did the model *detect*? (High R = low false negatives).
• **$\mathbf{F1}$ Score**: The harmonic mean of Precision and Recall.



YOLOv8 Performance Metrics (Per-Class Precision, Recall & F1 Score)

This graph shows how accurately the model detects each object class:

- **Precision (blue):** How often the detected objects are correct.
- **Recall (orange):** How well the model finds all instances of an object.
- **F1 Score (green):** The balance between Precision and Recall.

Each point represents a class (like person, car, chair, etc.), showing how YOLOv8 performs across all detected objects

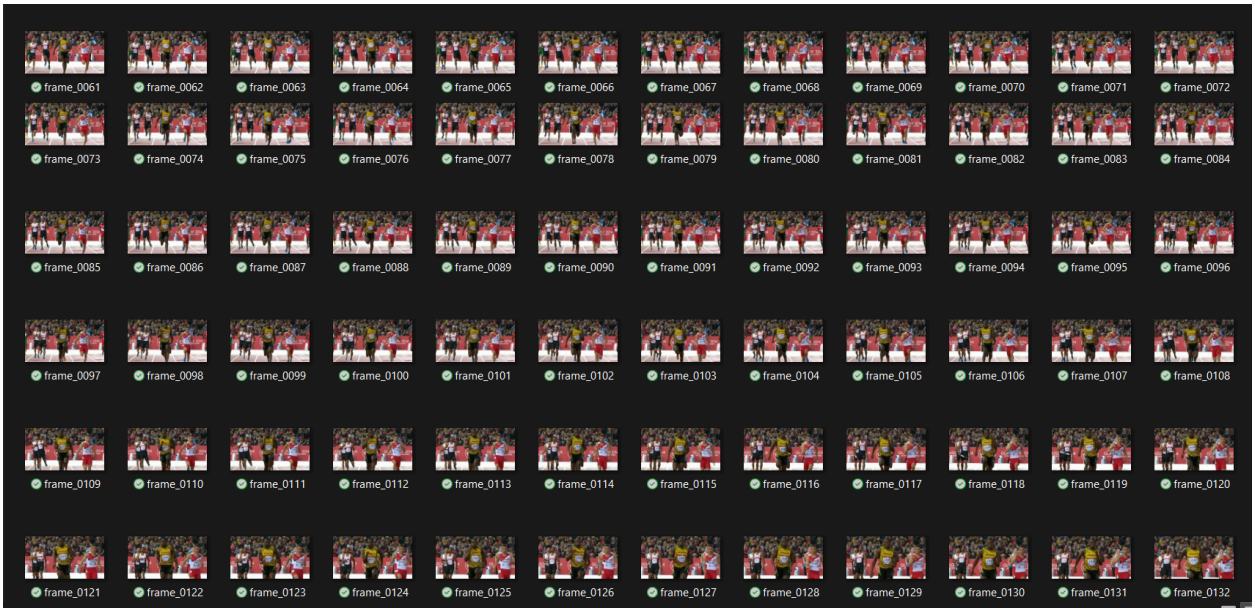
TASK 3:

```
from ultralytics import YOLO
import os
```

```
# Load YOLO model
model = YOLO('yolov8n.pt')  # You can use yolov8m.pt or yolov9.pt too

# Input/output directories
input_folder = r"C:\Users\aryan\OneDrive\Desktop\frames"
output_folder = r"C:\Users\aryan\OneDrive\Desktop\frames_detected"

os.makedirs(output_folder, exist_ok=True)

# Process all frames
results = model.predict(source=input_folder, save=True, project=output_folde
r, name='')
```

```
from ultralytics import YOLO
import os

# Load YOLO segmentation model
model_seg = YOLO('yolov8n-seg.pt')

input_folder = r"C:\Users\aryan\OneDrive\Desktop\frames"
output_folder_seg = r"C:\Users\aryan\OneDrive\Desktop\frames_segmented"

os.makedirs(output_folder_seg, exist_ok=True)

results = model_seg.predict(source=input_folder, save=True, project=output_f
older_seg, name='')
```

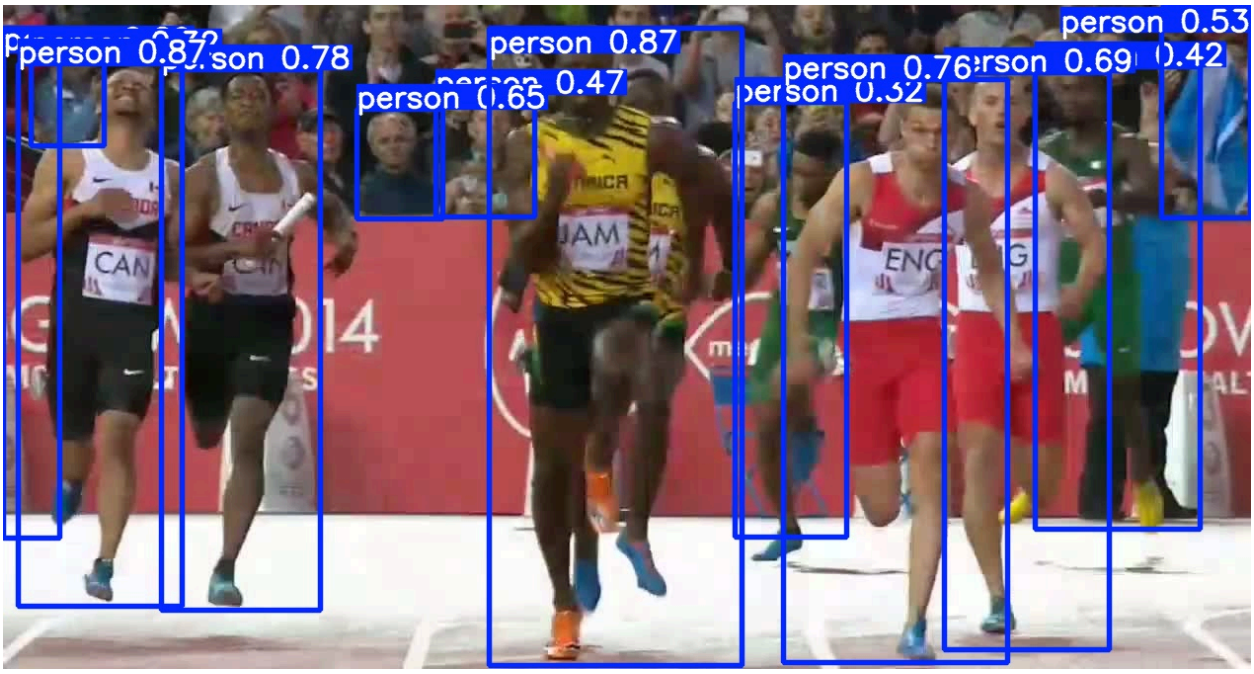
the original video:
using FFmpeg , i have divided the video into 132 Frames.

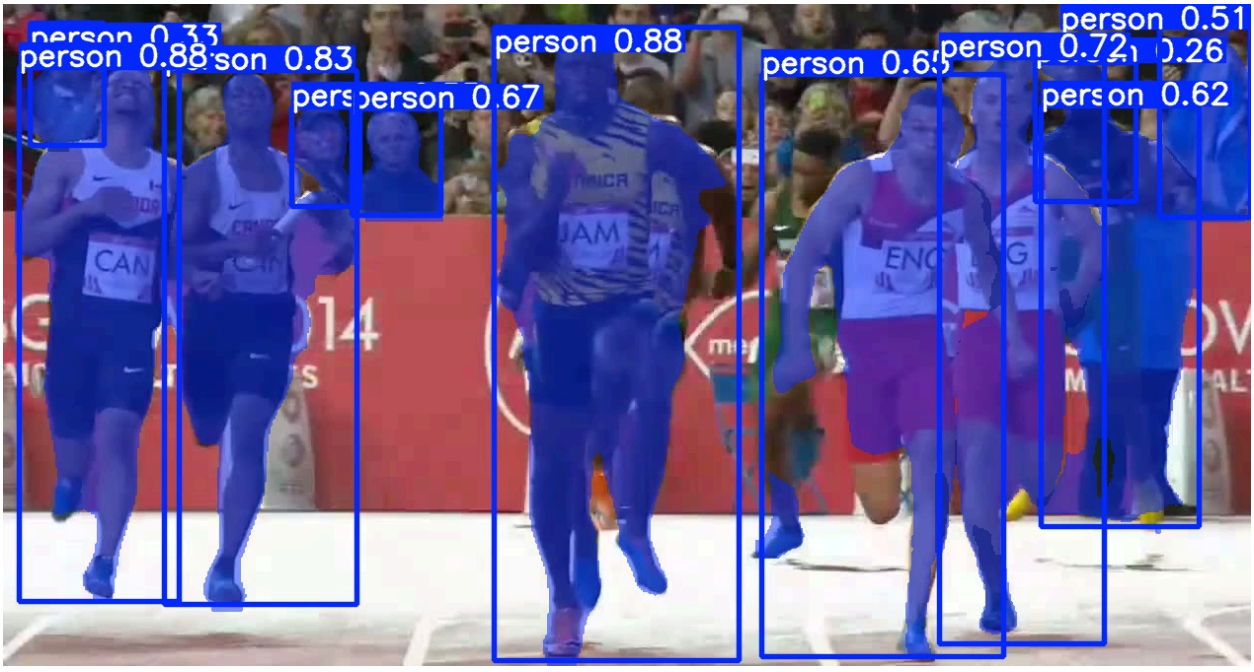


Saved it under Frames folder

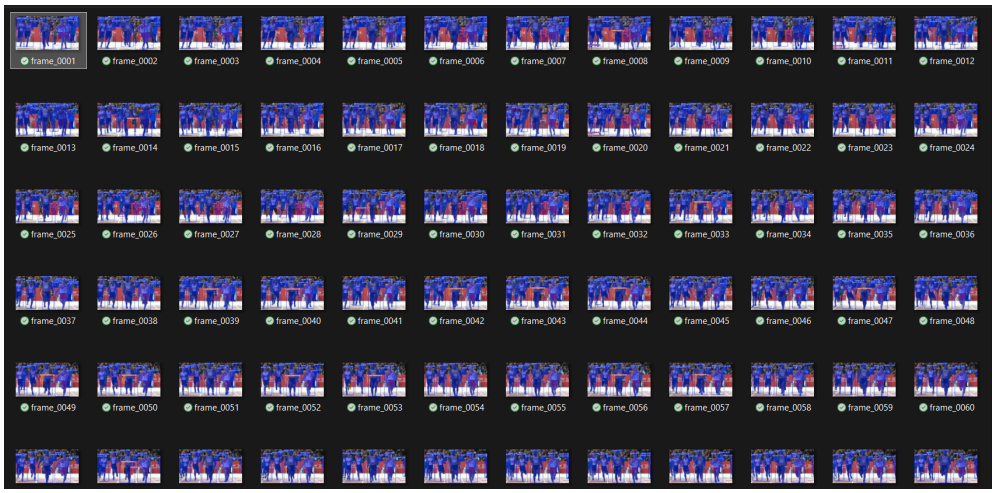
then, i have applied Multiple image Object Detection and Segmentation, as learned in TASK 2.

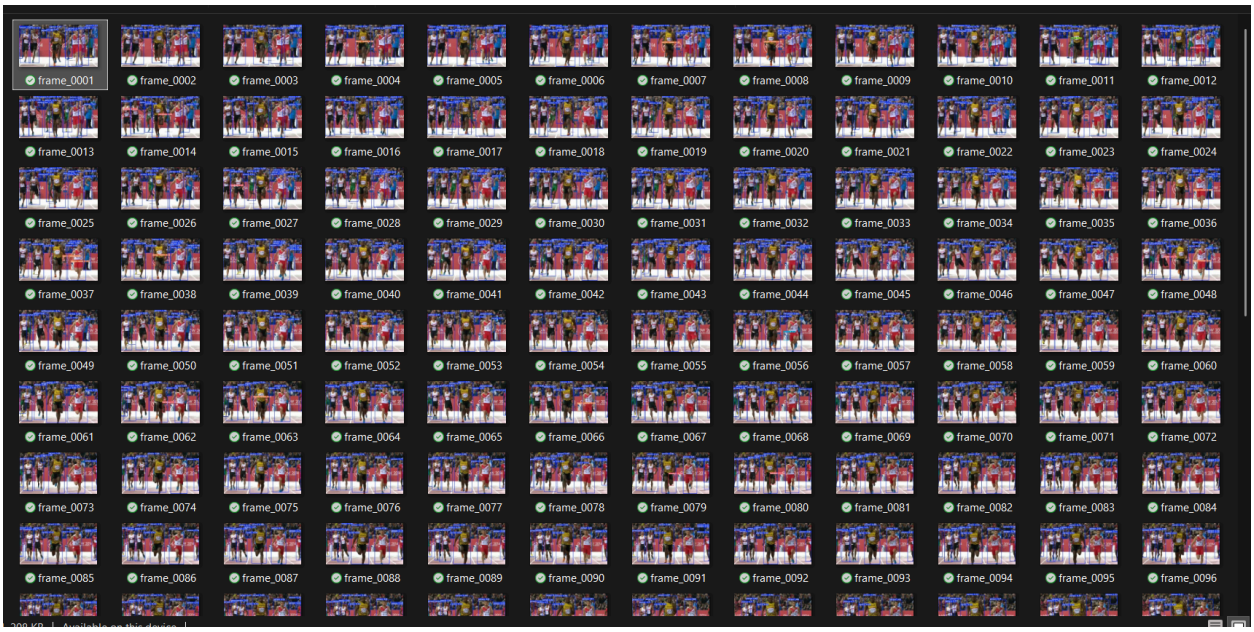
These are the Sample Detected and segmented Images ,





Similarly the segmentation and detection is applied to all the 132 frames of the video





Rebuilding Videos from Frames.

🎞️ Stitching Images into a Video (Using FFmpeg)

After performing object detection or segmentation on each frame, you generated hundreds of processed images like:

```
frame_0001.jpg
frame_0002.jpg
frame_0003.jpg
...
```

These were saved inside:

- For **Detection:**
  `C:\Users\aryan\OneDrive\Desktop\frames_detected\predict`
- For **Segmentation:**
  `C:\Users\aryan\OneDrive\Desktop\frames_segmented\predict`

⚙️ Step 1: Convert Processed Frames → Video

You used **FFmpeg** (installed manually) to combine those frames back into a video.

✅ Command for Detection Video:

```
& "C:\Users\aryan\Downloads\ffmpeg-2025-10-27-git-68152978b5-full_build\ffmpeg-2025-10-27-git-68152978b5-full_build\bin\ffmpeg.exe" -framerate 10 -i "C:\Users\aryan\OneDrive\Desktop\frames_detected\predict\frame_%04d.jpg" -c:v libx264 -pix_fmt yuv420p "C:\Users\aryan\OneDrive\Desktop\detected_video.mp4"
```

✅ Command for Segmentation Video:

```
& "C:\Users\aryan\Downloads\ffmpeg-2025-10-27-git-68152978b5-full_build\ffmpeg-2025-10-27-git-68152978b5-full_build\bin\ffmpeg.exe" -framerate 10 -i "C:\Users\aryan\OneDrive\Desktop\frames_segmented\predict\frame_%04d.jpg" -c:v libx264 -pix_fmt yuv420p "C:\Users\aryan\OneDrive\Desktop\segmented_video.mp4"
```

🔍 Explanation of the Command

Parameter	Meaning
`-framerate 10`	Sets frame rate (10 images per second). You can use 24 for smoother video.
`-i "frame_%04d.jpg"`	Reads images named as frame_0001.jpg, frame_0002.jpg, etc.
`-c:v libx264`	Uses H.264 codec — standard for MP4 videos.
`-pix_fmt yuv420p`	Ensures compatibility with all video players.
`"output.mp4"`	Name and path of the generated video file.

🧩 Step 2: Merge Multiple Videos Side-by-Side

Once both videos (detection + segmentation) were created, you merged them horizontally for comparison.

✅ Command for Merging Detection + Segmentation:

```
& "C:\Users\aryan\Downloads\ffmpeg-2025-10-27-git-68152978b5-full_build\ffmpeg-2025-10-27-git-68152978b5-full_build\bin\ffmpeg.exe" -i "C:\Users\aryan\OneDrive\Desktop\detected_video.mp4" -i "C:\Users\aryan\OneDrive\Desktop\segmented_video.mp4" -filter_complex "hstack=inputs=2" "C:\Users\aryan\OneDrive\Desktop\final_combined.mp4"
```

Or if you also want to include the **original video** side by side:

```
& "C:\Users\aryan\Downloads\ffmpeg-2025-10-27-git-68152978b5-full_build\ffmpeg-2025-10-27-git-68152978b5-full_build\bin\ffmpeg.exe" -i "C:\Users\aryan\OneDrive\Desktop\input.mp4" -i "C:\Users\aryan\OneDrive\Desktop\detected_video.mp4" -i "C:\Users\aryan\OneDrive\Desktop\segmented_video.mp4" -filter_complex "hstack=inputs=3" "C:\Users\aryan\OneDrive\Desktop\final_combined.mp4"
```

🔍 Explanation of Merge Command

Part	Description
`-i "file1.mp4" -i "file2.mp4"`	Input videos to combine.
`-filter_complex "hstack=inputs=2"`	Combines videos **horizontally** (use `vstack` for vertical).
`"final_combined.mp4"`	Output merged video file.

📁 Folder structure summary:

```
Desktop/
│
├── input.mp4
├── frames/
│   └── frame_0001.jpg ...
│
├── frames_detected/
│   └── predict/
│       └── frame_0001.jpg ...
│
├── frames_segmented/
│   └── predict/
│       └── frame_0001.jpg ...
│
├── detected_video.mp4
├── segmented_video.mp4
└── final_combined.mp4
```

🎯 Final Output:

- 🎬 `detected_video.mp4` → bounding boxes drawn
- 🎨 `segmented_video.mp4` → colored object masks
- 🧩 `final_combined.mp4` → side-by-side comparison video