# **SAM-2 Video Processing Script Documentation**

This Python script is specifically designed for automating video content segmentation and background removal using the SAM2 model, a powerful tool for segmentation tasks. The script guides you through setting up the necessary environment, pre-processing video data, and applying deep learning models for segmentation.

Link to Notebook: <https://github.com/Aadharsh1/ML-Deep-Learning/blob/main/Pose_Estimation_Experiments/SAM2_Segmentation/sam_2_NEW.ipynb>

## **Initial Setup**

This segment involves mounting Google Drive to access stored video files, unzipping video content into a designated processing directory, and setting up the workspace environment.

from google.colab import drive

drive.mount('/content/drive', force\_remount=True)

!unzip /content/drive/MyDrive/weds\_vids.zip -d /content/videos

import os

HOME = os.getcwd()

print("HOME:", HOME)

## **Repository and Dependency Setup**

This section involves cloning the necessary GitHub repository for segmentation, installing dependencies, and preparing model checkpoints for use.

!git clone https://github.com/facebookresearch/segment-anything-2.git

!pip install -e . -q

!python setup.py build\_ext --inplace

!pip install -q supervision[assets] jupyter\_bbox\_widget

!mkdir -p {HOME}/checkpoints

!wget -q https://dl.fbaipublicfiles.com/segment\_anything\_2/072824/sam2\_hiera\_tiny.pt -P {HOME}/checkpoints

!wget -q https://dl.fbaipublicfiles.com/segment\_anything\_2/072824/sam2\_hiera\_small.pt -P {HOME}/checkpoints

!wget -q https://dl.fbaipublicfiles.com/segment\_anything\_2/072824/sam2\_hiera\_base\_plus.pt -P {HOME}/checkpoints

!wget -q https://dl.fbaipublicfiles.com/segment\_anything\_2/072824/sam2\_hiera\_large.pt -P {HOME}/checkpoints

## **Mixed-Precision Setup**

Enables faster computation using mixed precision, which combines different data types to speed up model training and inference without significant loss of accuracy.

import torch

torch.autocast(device\_type="cuda", dtype=torch.bfloat16).\_\_enter\_\_()

if torch.cuda.get\_device\_properties(0).major >= 8:

torch.backends.cuda.matmul.allow\_tf32 = True

torch.backends.cudnn.allow\_tf32 = True

**Explanation:**

* torch.autocast: Enables automatic mixed precision for operations, reducing the computational demand.
* Checks CUDA device properties to optimize matrix operations and convolutional operations based on the capabilities of the available GPU.

## **Model Loading and Initialization**

Loads the segmentation model with a specified configuration and prepares it for video processing.

cd /content/segment-anything-2

DEVICE = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

CHECKPOINT = f"{HOME}/checkpoints/sam2\_hiera\_large.pt"

CONFIG = "sam2\_hiera\_l.yaml"

predictor = build\_sam2\_video\_predictor(CONFIG, CHECKPOINT)

## **Video Preprocessing**

Handles frame extraction from the video file, preparing the data for segmentation.

import cv2

import supervision as sv

from pathlib import Path

vid\_name = 'm.mp4'

SOURCE\_VIDEO = f'/content/preprocessed\_videos/{vid\_name}'

total\_frames = count\_frames(SOURCE\_VIDEO)

SCALE\_FACTOR = 1

START\_IDX = 0

END\_IDX = total\_frames

SOURCE\_FRAMES = Path(HOME) / Path(SOURCE\_VIDEO).stem

SOURCE\_FRAMES.mkdir(parents=True, exist\_ok=True)

frames\_generator = sv.get\_video\_frames\_generator(SOURCE\_VIDEO, start=START\_IDX, end=END\_IDX)

images\_sink = sv.ImageSink(

target\_dir\_path=SOURCE\_FRAMES.as\_posix(),

overwrite=True,

image\_name\_pattern="{:05d}.jpeg"

)

with images\_sink:

for frame in frames\_generator:

frame = sv.scale\_image(frame, SCALE\_FACTOR)

images\_sink.save\_image(frame)

**Explanation:**

* Calculates the total number of frames in the video to manage processing.
* SCALE\_FACTOR adjusts the size of each frame for processing consistency.
* Frames are saved individually to allow detailed processing.

## **Selecting Coordinates for Object Masking**

This crucial step involves inputting the coordinates for the object you wish to mask, initializing the segmentation process in specific areas of interest within the video frames.

ann\_frame\_idx = 0 # Frame index to interact with

ann\_obj\_id = 1 # Unique object identifier

points = np.array([[850,400]], dtype=np.float32) # Coordinates of the object to mask

labels = np.array([1], dtype=np.int32) # Label for the object

\_, out\_obj\_ids, out\_mask\_logits = predictor.add\_new\_points(

inference\_state=inference\_state,

frame\_idx=ann\_frame\_idx,

obj\_id=ann\_obj\_id,

points=points,

labels=labels,

)

**Explanation:**

* Specifies the frame and object IDs to initialize interaction.
* points and labels define the exact location and identification for the initial mask, guiding the model on where to focus its segmentation efforts.

## **Segmentation and Output**

Processes each frame, applies the segmentation mask, and compiles the results into a final video output.

import matplotlib.pyplot as plt

from PIL import Image

output\_video\_path = f"/content/output\_videos/{vid\_name.split('.')[0]}.mp4"

!ffmpeg -framerate 30 -i '{output\_dir}/%05d.jpg' -c:v libx264 -profile:v high -crf 20 -pix\_fmt yuv420p '{output\_video\_path}'

**Explanation:**

* Combines the processed frames back into a video file using ffmpeg, setting the frame rate and compression settings for optimal quality.

## **Cleanup**

Removes all temporary files and directories to clear up space after processing is complete.

import shutil

shutil.rmtree('/content/segmented\_frames')

shutil.rmtree(video\_dir)

os.remove(SOURCE\_VIDEO)

**Explanation:**

* Cleans up the environment by deleting temporary files and directories created during the video processing.