

my_notebook

August 7, 2025

```
[12]: # Import necessary libraries
import torch
import numpy as np
import os
import subprocess
import threading
import queue
import re
import json
import time

# for model
from transformers import AutoProcessor, Gemma3nForConditionalGeneration

# for video streaming
import cv2
from PIL import Image

# for audio streaming
from pydub import AudioSegment
import librosa

# for youtube links
from yt_dlp import YoutubeDL

# for visualization
import matplotlib.pyplot as plt
from IPython.display import Audio, display

# for creating app demo
import gradio as gr
```

1 Model

```
[13]: model_id = "google/gemma-3n-e4b-it"

processor = AutoProcessor.from_pretrained(model_id)
model = Gemma3nForConditionalGeneration.from_pretrained(model_id,
    ↪device_map="cpu", torch_dtype=torch.bfloat16).eval()
```

Loading checkpoint shards: 100% | 4/4 [00:00<00:00, 6.20it/s]
Some weights of the model checkpoint at google/gemma-3n-e4b-it were not used when initializing Gemma3nForConditionalGeneration:
['model.vision_tower.timm_model.conv_stem.conv.bias']
- This IS expected if you are initializing Gemma3nForConditionalGeneration from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing Gemma3nForConditionalGeneration from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).

```
[14]: yes_token_id = processor.tokenizer("Yes",
    ↪add_special_tokens=False)["input_ids"][0]
no_token_id = processor.tokenizer("No",
    ↪add_special_tokens=False)["input_ids"][0]
```

2 Method

```
[15]: # A list indications of harm, abuse, aggression in videos and audios.
visual_harm_indicators = ["Violence", "Sexual Content"]
optional_visual_harm_indicators = ["Child Smoking", "Child Unsupervised"]

audio_harm_indicators = ["Inappropriate Language", "Verbal Abuse"]
optional_audio_harm_indicators = ["Child Screaming", "Child Crying"]

system_message = {
    "role": "system",
    "content": [{ "type": "text", "text": "" "You are a harm detection assistant.
        You must answer each prompt with only one word: "Yes" or
    ↪ "No".

        Do not provide explanations, comments, or any extra text.
        Be precise and cautious in your judgment." "" }],
}

def preprocess_user_prompt(input_type, personalized_harm_indicators):
```

```

    if input_type == "VIDEO":
        indicators = visual_harm_indicators + [i for i in
personalized_harm_indicators if i in optional_visual_harm_indicators]
        PROMPT = f"""Watch the video and respond with "Yes" if you see any
of the following: {" OR ".join(indicators)}"""
    else:
        indicators = audio_harm_indicators + [i for i in
personalized_harm_indicators if i in optional_audio_harm_indicators]
        PROMPT = f"""Listen to the audio and respond with "Yes" if you hear
any of the following: {" OR ".join(indicators)}"""

    PROMPT = PROMPT + """. Otherwise Respond with "No"."""
    return PROMPT

```

```

[16]: def detect_harm(messages, threshold =0.9):

    inputs = processor.apply_chat_template(
        messages,
        add_generation_prompt=True,
        tokenize=True,
        return_dict=True,
        return_tensors="pt",
    ).to(model.device)

    with torch.inference_mode():
        output = model.generate(**inputs.to(torch.bfloat16),
                                max_new_tokens=140,
                                return_dict_in_generate=True,

                                # the models' answers should be either Yes
or No, set do_sample=False to generate more deterministic and predictable
outputs.

                                do_sample=False,

                                # List of tensors, one per generated token.
Each tensor represent the unnormalized probabilities for all vocabulary
tokens (Logits)

                                output_scores=True
                                )

        logits = output.scores
        yes_logit = logits[0][0][yes_token_id]
        no_logit = logits[0][0][no_token_id]

```

```

        softmax_probs = torch.softmax(torch.tensor([yes_logit, no_logit],
↳dtype=torch.float64), dim=0)
        yes_softmax = softmax_probs[0].item()
        no_softmax = softmax_probs[1].item()

        if yes_softmax > threshold:
            harm_flag= True
        else:
            harm_flag = False

        # input_len = inputs["input_ids"].shape[-1]
        # print("model input len", input_len)
        # generated_ids = output.sequences # The full generated token ids
↳(prompt + new tokens)
        # generated_ids = generated_ids[0][input_len:] # Only new tokens
        # output_text = processor.decode(generated_ids,
↳skip_special_tokens=True)
        # print("output_text", output_text)
        # print("softmax_probs", softmax_probs)
        print("yes probability: ", yes_softmax)

    return harm_flag, yes_softmax

```

```

[17]: def record_visual_evidence(frames, timestamps, input_file, output_dir):
    name = input_file.split("/")[-1].split(".")[0]
    folder_path = os.path.join(output_dir, f"{name}_evidence_v_s_{timestamps[0]:
↳.2f}s_e_{timestamps[-1]:.2f}s")
    os.makedirs(folder_path, exist_ok=True)

    for idx, frame in enumerate(frames):
        frame.save(os.path.join(folder_path, f"frame_{idx}.png"))

def visualize_visual_window(frames, progress, cols=8, image_size=2):
    print(progress)
    rows = (len(frames) + cols - 1) // cols
    figsize = (cols * image_size, rows * image_size)
    plt.figure(figsize=figsize)
    for i, frame in enumerate(frames):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(frame)
        plt.axis("off")

    plt.tight_layout()
    plt.show()

```

```

def extract_youtube_video(url, output_path="outputs"):
    # with YoutubeDL({'quiet': True}) as ydl:
    #     info = ydl.extract_info(url, download=False)
    #     title = info.get("title", "video")
    #     title = title.replace(" ", "_").replace("/", "_")
    url_id = url.split("=")[-1]
    video_filename = f"example_videos/{url_id}.mp4"
    command = [
        "yt-dlp",
        "-f", "mp4",
        "-o", video_filename,
        url
    ]
    subprocess.run(command, check=True)
    return video_filename, url_id

def visual_moving_window(input_file, personalized_harm_indicators, output_dir,
    ↪target_fps, window_duration_sec, overlap_duration_sec, visualize=False):
    max_frames = int(target_fps * window_duration_sec)
    overlap_frames = int(target_fps * overlap_duration_sec)

    harm_detection_prompt = preprocess_user_prompt("VIDEO",
    ↪personalized_harm_indicators)

    cap = cv2.VideoCapture(input_file)
    if not cap.isOpened():
        raise RuntimeError(f"Failed to open video: {input_file}")

    original_fps = cap.get(cv2.CAP_PROP_FPS)
    if original_fps <= 0:
        original_fps = 30 # fallback # raise RuntimeError("FPS is zero. Can't
    ↪continue.")
    frame_interval = max(1, int(original_fps // target_fps))

    frames, timestamps = [], []
    window_idx, frame_idx = 0, 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        if frame_idx % frame_interval == 0:
            timestamp_sec = cap.get(cv2.CAP_PROP_POS_MSEC) / 1000.0
            timestamps.append(round(timestamp_sec, 3))

            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            pil_image = Image.fromarray(rgb_frame)

```

```

frames.append(pil_image)

if len(frames) == max_frames:
    print("frames_len", len(frames))
    progress = f"Processing video window {window_idx}:␣
↪start={timestamps[0]:.2f}s, end={timestamps[-1]:.2f}s"
    yield progress

    if visualize:
        visualize_visual_window(frames, progress)

    messages = [
        system_message,
        {
            "role": "user",
            "content": [
                * [{"type": "image", "image": frame} for frame in␣
↪frames],
                {"type": "text", "text": harm_detection_prompt},
            ],
        },
    ]
    is_harm, yes_prop = detect_harm(messages)
    print(is_harm)
    yield is_harm, yes_prop

    if is_harm:
        record_visual_evidence(frames, timestamps, input_file,␣
↪output_dir)

    window_idx += 1
    frames = frames[-overlap_frames:] # keep overlap frames only
    timestamps = timestamps[-overlap_frames:]

    frame_idx += 1

cap.release()

```

```

[18]: def record_audio_evidence(audio_buffer, start, end, input_file, output_dir):
    name = input_file.split("/")[-1].split(".")[0]
    filename = f"{output_dir}/{name}_evidence_a_s_{start / 1000.0:.2f}s_e_{end /
↪1000.0:.2f}s.wav"
    # Store the sliced audio file to the defined path
    audio_buffer.export(filename, format="wav")

def play_audio_window(samples_float, target_sampling_rate, progress):

```

```

print(progress)
display(Audio(samples_float, rate=target_sampling_rate))

def extract_audio_to_wav(stream_path, target_sampling_rate,
    ↪output_dir="outputs"):
    """
        Audio encoding
        When encoding audio data with your own code implementation for use with
    ↪Gemma 3n, you should follow the recommended conversion process.
        If you are working with audio files encoded in a specific format, such
    ↪as MP3 or WAV encoded data, you must first decode these to samples using a
    ↪library such as ffmpeg.
        Once the data is decoded, convert the audio into mono-channel, 16 kHz
    ↪float32 waveforms in the range [-1, 1].

        For example, if you are working with stereo signed 16-bit PCM integer
    ↪WAV files at 44.1 kHz, follow these steps:

        Resample the audio data to 16 kHz
        Downmix from stereo to mono by averaging the 2 channels
        Convert from int16 to float32, and divide by 32768.0 to scale to the
    ↪range [-1, 1]
        https://ai.google.dev/gemma/docs/capabilities/audio
    """
    name = stream_path.split("/")[-1].split(".")[0]
    output_wav_path = f"{output_dir}/{name}_audio.wav"
    command = [
        "ffmpeg",
        "-y",
        "-i", stream_path,
        "-vn",
        "-acodec", "pcm_s16le",
        "-ar", str(target_sampling_rate), #16 kHz
        "-ac", "1", #mono-channel
        output_wav_path
    ]
    subprocess.run(command, check=True, stdout=subprocess.DEVNULL,
    ↪stderr=subprocess.DEVNULL)
    return output_wav_path

def audio_moving_window(input_file, personalized_harm_indicators, output_dir,
    ↪target_sampling_rate, window_duration_seconds, overlap_duration_sec,
    ↪visualize=False):
    interval = window_duration_seconds * 1000
    overlap = overlap_duration_sec * 1000
    if overlap >= interval:

```

```

        raise ValueError("Overlap duration must be less than window duration.")

    harm_detection_prompt = preprocess_user_prompt( "AUDIO",
    ↪personalized_harm_indicators)

    audio_stream = extract_audio_to_wav(input_file, target_sampling_rate)
    audio = AudioSegment.from_wav(audio_stream)

    window_idx, start, audio_len = 0, 0, len(audio)
    while start < audio_len:
        end = min(start + interval, audio_len)
        if end - start < interval and window_idx > 0:
            break

        chunk = audio[start:end]

        progress = f"Processing audio window {window_idx}: start={start / 1000.
    ↪0:.2f}, end={end / 1000.0:.2f}"
        yield progress

        samples = chunk.get_array_of_samples()
        samples_float = librosa.util.buf_to_float(samples,n_bytes=2, dtype=np.
    ↪float32)

        if visualize:
            play_audio_window(samples_float, target_sampling_rate, progress)

    messages = [
        system_message,
        {
            "role" : "user",
            "content": [
                { "type": "audio", "audio" : samples_float },
                { "type": "text", "text" : harm_detection_prompt}
            ]
        },
    ]
    is_harm , yes_prop = detect_harm(messages)
    print("is_harm", is_harm)
    yield is_harm, yes_prop

    if is_harm:
        record_audio_evidence(chunk, start, end, input_file,output_dir)

    window_idx += 1
    if end == audio_len:

```



```

        break
    start = end - overlap

```

3 Testing

```

[20]: def save_testing_results(file_path, input_source, optional_harm_indicators,
    ↪output_dir = "outputs", visualize=False, audio_only=False):
    print("MAIN INPUT: ",file_path)
    os.makedirs(output_dir, exist_ok=True)

    if input_source == "YouTube Link":
        file_path, name = extract_youtube_video(file_path, output_dir)
        print("youtube link file path", file_path)
    else:
        name, ext = os.path.splitext(os.path.basename(file_path))

    video_results, audio_results = [], []
    if not audio_only:
        for output in visual_moving_window(
            input_file=file_path,
            personalized_harm_indicators=optional_harm_indicators,
            output_dir=output_dir,
            target_fps=1,
            window_duration_sec=30,
            overlap_duration_sec=5,
            visualize=visualize
        ):
            if isinstance(output, str):
                video_results.append(f"[VIDEO] {output}\n")
                print(f"[VIDEO] {output}")
            else:
                icon = f" Harm Detected with confidence score {output[1]:.2f}.
    ↪Saved Recorded Evidence.\n" if output[0] else " No Harm Detected\n"
                video_results.append(icon)
                print(f"{output} {icon}")

        else:
            video_results.append("Visual track disabled. Skipping frames analysis.")

    # Check if the video contains audio
    probe = ["ffprobe", "-i", file_path, "-show_streams", "-select_streams",
    ↪"a", "-loglevel", "error"]
    has_audio = subprocess.run(probe, capture_output=True, text=True).stdout.
    ↪strip() != ""

```

```

if has_audio:
    for output in audio_moving_window(
        input_file=file_path,
        personalized_harm_indicators=optional_harm_indicators,
        output_dir=output_dir,
        target_sampling_rate=16000,
        window_duration_seconds=30,
        overlap_duration_sec=5,
        visualize=visualize
    ):
        if isinstance(output, str):
            audio_results.append(f"[AUDIO] {output}\n")
            print(f"[AUDIO] {output}\n")
        else:
            icon = f" Harm Detected with confidence score {output[1]:.2f}.␣
↪Saved Recorded Evidence.\n" if output[0] else " No Harm Detected\n"
            audio_results.append(icon)
            print(f"{output} {icon}")
        else:
            audio_results.append("No audio track found. Skipping audio analysis.")

# Store results and logs
log_data = {
    "video_logs": video_results,
    "audio_logs": audio_results
}

log_output_path = f"{output_dir}/{name}_processed_results.json"
with open(log_output_path, "w", encoding="utf-8") as f:
    json.dump(log_data, f, ensure_ascii=False, indent=2)

print(f"MODEL RESULTS SAVED IN {log_output_path}")

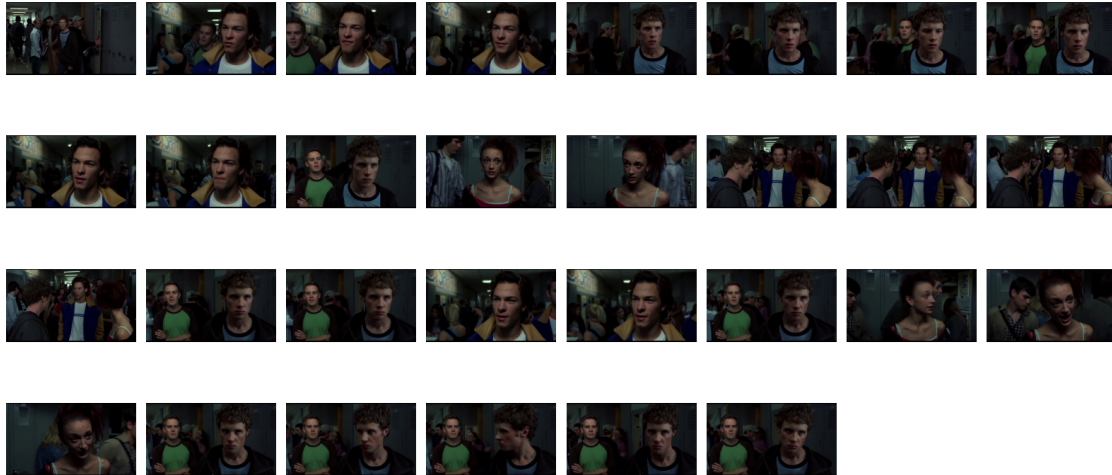
```

```

[9]: save_testing_results(file_path = "example_videos/verbal_abuse_movie.mp4",
    input_source= "Local Video",
    optional_harm_indicators= optional_audio_harm_indicators +␣
↪optional_visual_harm_indicators,
    output_dir = "outputs",
    visualize=True)

```

MAIN INPUT: example_videos/verbal_abuse_movie.mp4
 frames_len 30
 [VIDEO] Processing video window 0: start=0.00s, end=29.00s
 Processing video window 0: start=0.00s, end=29.00s



The following generation flags are not valid and may be ignored: ['top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.8596637505099167

False

(False, 0.8596637505099167) No Harm Detected

[AUDIO] Processing audio window 0: start=0.00, end=30.00

Processing audio window 0: start=0.00, end=30.00

<IPython.lib.display.Audio object>

The following generation flags are not valid and may be ignored: ['top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.9993736658418905

is_harm True

(True, 0.9993736658418905) Harm Detected with confidence score 1.00. Saved Recorded Evidence.

MODEL RESULTS SAVED IN outputs/verbal_abuse_movie_processed_results.json

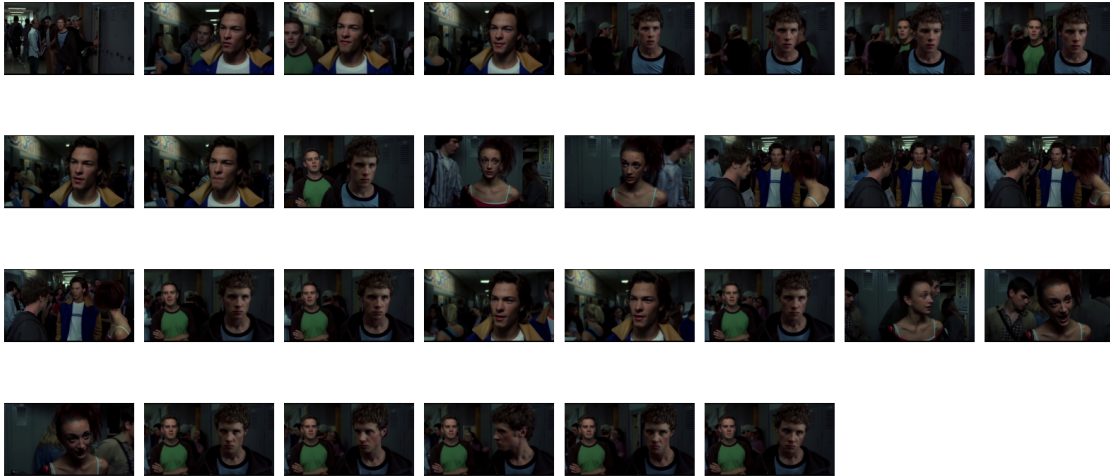
```
[10]: save_testing_results(file_path = "example_videos/violance_movie.mp4",
                           input_source= "Local Video",
                           optional_harm_indicators= optional_audio_harm_indicators +
                           ↪optional_visual_harm_indicators,
                           output_dir = "outputs",
                           visualize=True)
```

MAIN INPUT: example_videos/violance_movie.mp4

frames_len 30

[VIDEO] Processing video window 0: start=0.00s, end=29.00s

Processing video window 0: start=0.00s, end=29.00s



The following generation flags are not valid and may be ignored: ['top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.8596637505099167

False

(False, 0.8596637505099167) No Harm Detected

[AUDIO] Processing audio window 0: start=0.00, end=30.00

Processing audio window 0: start=0.00, end=30.00

<IPython.lib.display.Audio object>

The following generation flags are not valid and may be ignored: ['top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.9993736658418905

is_harm True

(True, 0.9993736658418905) Harm Detected with confidence score 1.00. Saved Recorded Evidence.

MODEL RESULTS SAVED IN outputs/violance_movie_processed_results.json

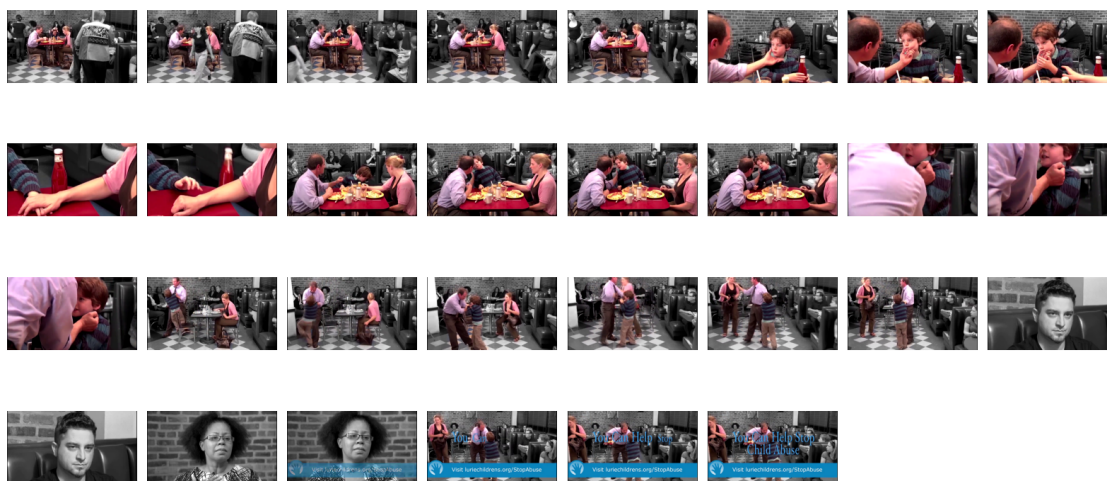
```
[11]: save_testing_results(file_path = "https://www.youtube.com/watch?v=F85XP1qoqjg",
                           input_source= "YouTube Link",
                           optional_harm_indicators=optional_audio_harm_indicators +
                           ↪optional_visual_harm_indicators,
                           output_dir = "outputs",
                           visualize=True)
```

MAIN INPUT: <https://www.youtube.com/watch?v=F85XP1qoqjg>

```

[youtube] Extracting URL: https://www.youtube.com/watch?v=F85XP1qoqjg
[youtube] F85XP1qoqjg: Downloading webpage
[youtube] F85XP1qoqjg: Downloading tv client config
[youtube] F85XP1qoqjg: Downloading tv player API JSON
[youtube] F85XP1qoqjg: Downloading ios player API JSON
[youtube] F85XP1qoqjg: Downloading m3u8 information
[info] F85XP1qoqjg: Downloading 1 format(s): 18
[download] example_videos/F85XP1qoqjg.mp4 has already been downloaded
[download] 100% of 2.58MiB
youtube link file path example_videos/F85XP1qoqjg.mp4
frames_len 30
[VIDEO] Processing video window 0: start=0.00s, end=28.06s
Processing video window 0: start=0.00s, end=28.06s

```



The following generation flags are not valid and may be ignored: ['top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.9992445943672705

True

(True, 0.9992445943672705) Harm Detected with confidence score 1.00. Saved Recorded Evidence.

[AUDIO] Processing audio window 0: start=0.00, end=30.00

Processing audio window 0: start=0.00, end=30.00

<IPython.lib.display.Audio object>

The following generation flags are not valid and may be ignored: ['top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.9999227780816594

is_harm True

(True, 0.9999227780816594) Harm Detected with confidence score 1.00. Saved Recorded Evidence.

MODEL RESULTS SAVED IN outputs/F85XP1qoqjg_processed_results.json

```
[12]: save_testing_results(file_path = "https://www.youtube.com/watch?v=klJMNosyLPk",
                           input_source= "YouTube Link",
                           optional_harm_indicators=optional_audio_harm_indicators +
                           optional_visual_harm_indicators,
                           output_dir = "outputs",
                           audio_only=True,
                           visualize=True)
```

MAIN INPUT: https://www.youtube.com/watch?v=klJMNosyLPk

[youtube] Extracting URL: https://www.youtube.com/watch?v=klJMNosyLPk

[youtube] klJMNosyLPk: Downloading webpage

[youtube] klJMNosyLPk: Downloading tv client config

[youtube] klJMNosyLPk: Downloading player 010fbc8d-main

[youtube] klJMNosyLPk: Downloading tv player API JSON

[youtube] klJMNosyLPk: Downloading ios player API JSON

[youtube] klJMNosyLPk: Downloading m3u8 information

[info] klJMNosyLPk: Downloading 1 format(s): 18

[download] example_videos/klJMNosyLPk.mp4 has already been downloaded

[download] 100% of 1.13MiB

youtube link file path example_videos/klJMNosyLPk.mp4

[AUDIO] Processing audio window 0: start=0.00, end=30.00

Processing audio window 0: start=0.00, end=30.00

<IPython.lib.display.Audio object>

The following generation flags are not valid and may be ignored: ['top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.9966233876182606

is_harm True

(True, 0.9966233876182606) Harm Detected with confidence score 1.00. Saved Recorded Evidence.

[AUDIO] Processing audio window 1: start=25.00, end=55.00

Processing audio window 1: start=25.00, end=55.00

<IPython.lib.display.Audio object>

The following generation flags are not valid and may be ignored: ['top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.8670357598021706

is_harm False

(False, 0.8670357598021706) No Harm Detected

MODEL RESULTS SAVED IN outputs/klJMNosyLPk_processed_results.json

```
[22]: testing_examples = ["example_videos/verbal_abuse_movie.mp4", "example_videos/
      ↪violence_movie.mp4"]
youtube_testing_examples = ["https://www.youtube.com/watch?
      ↪v=F85XP1qoqjg", "https://www.youtube.com/watch?v=klJMNosyLPk"]

testing_examples_ids = [os.path.basename(i)[:4] for i in testing_examples ]
youtube_testing_examples_ids = [i.split("=")[-1] for i in
      ↪youtube_testing_examples]

def retrieve_testing_results(file_path, input_source, output_dir = "outputs"):

    print("input preprocessed example:", file_path)

    if input_source == "YouTube Link":
        name = file_path.split("=")[-1]

    else:
        name = os.path.basename(file_path)[:4]

    if name in testing_examples_ids or name in youtube_testing_examples_ids:

        log_output_path = f"{output_dir}/{name}_processed_results.json"
        print("retrieve testing results from: ", log_output_path)

        with open(log_output_path, "r", encoding="utf-8") as f:
            log_data = json.load(f)
            video_logs_list, audio_logs_list = log_data["video_logs"],
            ↪log_data["audio_logs"]

            return video_logs_list, audio_logs_list

    else:
        return None, None
```

4 Live Demo

```
[23]: # SAME LOGIC AS save_testing_results BUT WITH YIELD FOR STREAMING RESULTS.
def run_audio(file_path, input_source, optional_harm_indicators, selected,
      ↪yt_url, output_dir = "outputs"):

    audio_logs = ""
    # RETRIEVE RESULTS FROM PREPROCESSED FROM TESTING EXAMPLES SHOWN ABOVE
    if input_source == "Preprocessed Example":
```



```

        file_path = selected
    if input_source == "YouTube Link":
        file_path = yt_url

    _, preprocessed_logs = retrieve_testing_results(file_path, input_source)
    if preprocessed_logs:
        for log in preprocessed_logs:
            audio_logs+=log
            yield audio_logs
            time.sleep(8)
    else:
        if input_source == "YouTube Link":
            file_path, name = extract_youtube_video(file_path, output_dir)

            # Check if the video contains audio
            probe = ["ffprobe", "-i", file_path, "-show_streams", "\n",
↪"-select_streams", "a", "-loglevel", "error"]
            has_audio = subprocess.run(probe, capture_output=True, text=True).
↪stdout.strip() != ""

            if has_audio:
                for output in audio_moving_window(
                    input_file=file_path,
                    personalized_harm_indicators=optional_harm_indicators,
                    output_dir=output_dir,
                    target_sampling_rate=16000,
                    window_duration_seconds=30,
                    overlap_duration_sec=5,
                ):
                    if isinstance(output, str):
                        audio_logs += f"[AUDIO] {output} \n"
                    else:
                        audio_logs += f" Harm Detected with confidence score_
↪{output[1]:.2f}. Saved Recorded Evidence.\n" if output[0] else " No Harm_
↪Detected\n"
                        print("video logs", audio_logs)
                        yield audio_logs
            else:
                audio_logs+= "No audio track found. Skipping audio analysis."

def run_video(file_path, input_source, optional_harm_indicators,
↪selected,yt_url, output_dir = "outputs"):
    video_logs = ""
    # RETRIEVE RESULTS FROM PREPROCESSED TESTING EXAMPLES SHOWN ABOVE
    if input_source == "Preprocessed Example":
        file_path = selected
    if input_source == "YouTube Link":

```



```

        file_path = yt_url
    preprocessed_logs, _ = retrieve_testing_results(file_path, input_source)
    if preprocessed_logs:
        for log in preprocessed_logs:
            video_logs+=log
            yield video_logs
            time.sleep(10)
    else:
        for output in visual_moving_window(
            input_file=file_path,
            personalized_harm_indicators=optional_harm_indicators,
            output_dir=output_dir,
            target_fps=1,
            window_duration_sec=10,
            overlap_duration_sec=1,
        ):
            if isinstance(output, str):
                video_logs+=f"[VIDEO] {output}\n"
            else:
                video_logs+= f" Harm Detected with confidence score {output[1]:
↪.2f}. Saved Recorded Evidence.\n" if output[0] else " No Harm Detected\n"
            print("video logs", video_logs)
            yield video_logs

example_paths = ["example_videos/verbal_abuse_movie.mp4", "example_videos/
↪violence_movie.mp4","example_videos/F85XP1qoqjg.mp4","example_videos/
↪klJMNosyLPk.mp4"]

# Gradio App Layout
with gr.Blocks() as demo:
    gr.Markdown("## Harm Detection for Child Protection")
    gr.Markdown("This app analyzes streaming or local videos with audio to
↪detect any indication of harm for the children")

    with gr.Row():
        harm_selector = gr.CheckboxGroup(
            label=" Select additional harm indicators you want to detect:",
            choices= visual_harm_indicators + audio_harm_indicators +
↪list(set(optional_visual_harm_indicators + optional_audio_harm_indicators)),
            value=visual_harm_indicators + audio_harm_indicators
        )

    with gr.Row():
        with gr.Column():
            input_source = gr.Radio(
                label="Select Input Source:",

```

```

        choices= ["Preprocessed Example", "Local Video", "YouTube_
↪Link", "IP Camera Streaming URL"],
        value="Preprocessed Example"
    )

    ip_url = gr.Textbox(label=" RTSP URL", placeholder="rtsp://
↪username:password@192.168.1.64:554/stream1", visible=False)
    yt_url = gr.Textbox(label=" YouTube Link", placeholder="https://
↪youtube.com/watch?v=...", visible=False)
    yt_video = gr.Video(label="Displayed Video", visible=False)
    video_upload = gr.Video(label=" Upload Video File", visible=False)
    video_example = gr.Video(label=" Select Preprocessed Example",
↪interactive=False, visible=True)
    example_selector = gr.Examples(
        examples=example_paths,
        inputs=[video_example],
        label="Example Videos"
    )
    run_btn = gr.Button(" Run Detection")

    with gr.Column():
        audio_output = gr.Textbox(label="Audio Logs", lines=11,
↪interactive=False)
        video_output = gr.Textbox(label="Video Logs", lines=11,
↪interactive=False)

    def toggle_inputs(source):
        return {
            ip_url: gr.update(visible=source == "IP Camera Streaming URL"),
            yt_url: gr.update(visible=source == "YouTube Link"),
            yt_video: gr.update(visible=source == "YouTube Link"),
            video_upload: gr.update(visible=source == "Local Video"),
            video_example: gr.update(visible=source == "Preprocessed Example")
        }

    input_source.change(
        fn=toggle_inputs,
        inputs=input_source,
        outputs=[ip_url, yt_url, video_upload, yt_video, video_example]
    )

    def display_yt_video(yt_path):
        print("ins display", yt_path)
        url_id = yt_path.split("=")[-1]
        video_filename = f"example_videos/{url_id}.mp4"
        print(video_filename)

```

```

        return video_filename

def get_example_name(video_path):
    selected_filename = os.path.basename(video_path)
    return selected_filename

selected= gr.State()
video_example.change(fn=get_example_name,inputs=video_example,
↳outputs=selected)
yt_url.change(fn = display_yt_video, inputs = yt_url, outputs=yt_video)
run_btn.click(
    fn=run_audio,
    inputs=[video_upload, input_source, harm_selector, selected, yt_url],
    outputs=audio_output
)

run_btn.click(
    fn=run_video,
    inputs=[video_upload, input_source, harm_selector, selected, yt_url],
    outputs=video_output
)

demo.launch(share=True)

```

* Running on local URL: <http://127.0.0.1:7882>
 * Running on public URL: <https://c881ed2902d7513196.gradio.live>

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (<https://huggingface.co/spaces>)

<IPython.core.display.HTML object>

[23]:

```

input preprocessed example: violence_movie.mp4
retrieve testing results from: outputs/violance_movie_processed_results.json
input preprocessed example: violence_movie.mp4
retrieve testing results from: outputs/violance_movie_processed_results.json
input preprocessed example: verbal_abuse_movie.mp4
retrieve testing results from:
outputs/verbal_abuse_movie_processed_results.json
input preprocessed example: verbal_abuse_movie.mp4
retrieve testing results from:
outputs/verbal_abuse_movie_processed_results.json
input preprocessed example: /tmp/gradio/12ae7400245ce6b60621e8548fb2a46237fbd639
07f21cf75c8719445ceecbc0/x6CP2Nrs8gE.mp4

```

input preprocessed example: /tmp/gradio/12ae7400245ce6b60621e8548fb2a46237fbd639
07f21cf75c8719445ceecbc0/x6CP2Nrs8gE.mp4
frames_len 10

The following generation flags are not valid and may be ignored: ['top_p',
'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

video logs [AUDIO] Processing audio window 0: start=0.00, end=30.00

The following generation flags are not valid and may be ignored: ['top_p',
'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.9998557401355151

is_harm True

video logs [AUDIO] Processing audio window 0: start=0.00, end=30.00

Harm Detected with confidence score 1.00. Saved Recorded Evidence.

The following generation flags are not valid and may be ignored: ['top_p',
'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.8670357598021706

False

frames_len 10

The following generation flags are not valid and may be ignored: ['top_p',
'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

yes probability: 0.8872045937171068

False

frames_len 10