

AI IN INDUSTRY 4.0

NVIDIA ASSIGNMENT-Track 2

GROUP-5

Varshini Vaddepalli-SE21UARI183

Akhila Sripada-SE21UARI155

Tanmayisri Vuppala -SE21UARI167

Sparsh Rathi-SE21UARI146

Human Action Recognition using NVIDIA VLM Workflow

Introduction

This document provides a detailed explanation of the implementation of a workflow for human action recognition using NVIDIA VLM (Visual Language Model) and Gradio. The workflow allows users to upload two videos, specify an action, and compare the success rate of detecting the action in each video. The solution incorporates video processing, NVIDIA NEVA API integration, and real-time comparison of success rates through a user-friendly interface.

Libraries Used

The implementation leverages the following Python libraries:

1. moviepy: For video processing and frame extraction.
2. Pillow (PIL): For image manipulation and Base64 conversion.
3. requests: For making HTTP requests to NVIDIA NEVA API.
4. Gradio: For creating an interactive user interface.
5. docx: For creating this documentation.
6. base64: For encoding images into Base64 format.

CODE EXPLANATION

1. Frame Extraction from Videos

The `extract_frames()` function extracts evenly spaced frames from a video. This ensures that the frames adequately represent the video's content for action detection.

```
def get_video_frames(video_path, frames_count=16):  
    clip = VideoFileClip(video_path)  
    time_length = clip.duration  
    frames = [  

```

```

        clip.get_frame(i * time_length / frames_count) for i in range(frames_count)
    ]
    return [Image.fromarray(frame) for frame in frames]

```

2. Converting Frames to Base64 Format

The `frame_to_base64()` function converts image frames into Base64 format. This is required by the NVIDIA NEVA API for processing images.

```


def image_to_base64(image_frame):
    temp_buffer = BytesIO()
    image_frame.save(temp_buffer, format="PNG")
    return base64.b64encode(temp_buffer.getvalue()).decode()

```

3. NVIDIA NEVA API Integration

The `detect_action_in_frame()` function sends a Base64-encoded frame and the specified action to the NVIDIA NEVA API. The API responds with a message indicating whether the action was detected.

```

def detect_action(image_b64, activity):
    headers = {
        "Authorization": f"Bearer {KEY}",
        "Accept": "application/json",
    }
    info = {
        "messages": [
            {
                "role": "user",
                "content": f"🤖 Do you see someone performing \"{activity}\" in this image?  ',
            }
        ],
        "max_tokens": 1024,
        "temperature": 0.20,
        "top_p": 0.70,
        "seed": 0,
        "stream": False,
    }
    response = requests.post(API_LINK, headers=headers, json=info)
    result = response.json()

```

```
return "yes" in result.get("choices", [{}])[0].get("message", {}).get("content", "").lower()
```

4. Calculating Success Rate

The `calculate_success_rate()` function calculates the percentage of frames in which the specified action is successfully detected by the NVIDIA NEVA API.

```
def calculate_accuracy(video_frames, activity):
    detections = 0
    for image in video_frames:
        image_b64 = image_to_base64(image)
        if detect_action(image_b64, activity):
            detections += 1
    return (detections / len(video_frames)) * 100
```

5. Gradio Interface Setup

The Gradio interface allows users to upload videos and specify an action. It processes the videos and displays the success rates for each video.

```
app = gr.Interface(
    fn=analyze_videos,
    inputs=[
        gr.Video(label="🎥 Upload Synthetic Video"),
        gr.Video(label="🎥 Upload Real Video"),
        gr.Textbox(label="💡 Specify Action (e.g., running, jumping)"),
    ],
    outputs=[
        gr.Textbox(label="📊 Synthetic Video Analysis"),
        gr.Textbox(label="📊 Real Video Analysis"),
    ],
    title="🎬 Action Detection using NVIDIA NEVA",
    description="✨ Upload videos and specify an action to analyze. This tool calculates the success rate for each video."
)
```

Steps to Arrive at the Solution

The following steps were taken to implement the solution:

1. Understanding Requirements: Reviewed the project requirements and datasets.
2. Data Preparation: Downloaded datasets (Charades, Sims4Action) and obtained synthetic

videos.

3. API Research: Studied NVIDIA NEVA API documentation to understand integration.

4. Framework Selection: Gradio for building an interactive interface.

5. Implementation: Developed functions for frame extraction, Base64 conversion, API calls, and success rate computation.

6. Testing: Validated the workflow with sample videos to ensure accuracy.

NVIDIA NEVA API Key and Its Role in the Code

The **NVIDIA NEVA API key** is a unique identifier provided by NVIDIA to authenticate access to its cloud-based Visual Language Models (VLM), including **Neva-22b**. This model leverages advanced AI capabilities for tasks such as visual and textual analysis.

Why is it Important?

The API key is critical because:

1. **Authentication:** It ensures that only authorized users can access the NEVA services.
2. **Usage Tracking:** It helps NVIDIA monitor usage to enforce rate limits and provide personalized support.
3. **Integration:** Allows seamless communication between the user's application and NVIDIA's cloud infrastructure.

How It Works in the Code

In the provided workflow:

1. The API key is included in the request headers for all API calls:

```
headers = {  
    "Authorization": f"Bearer {KEY}",  
    "Accept": "application/json",  
}
```

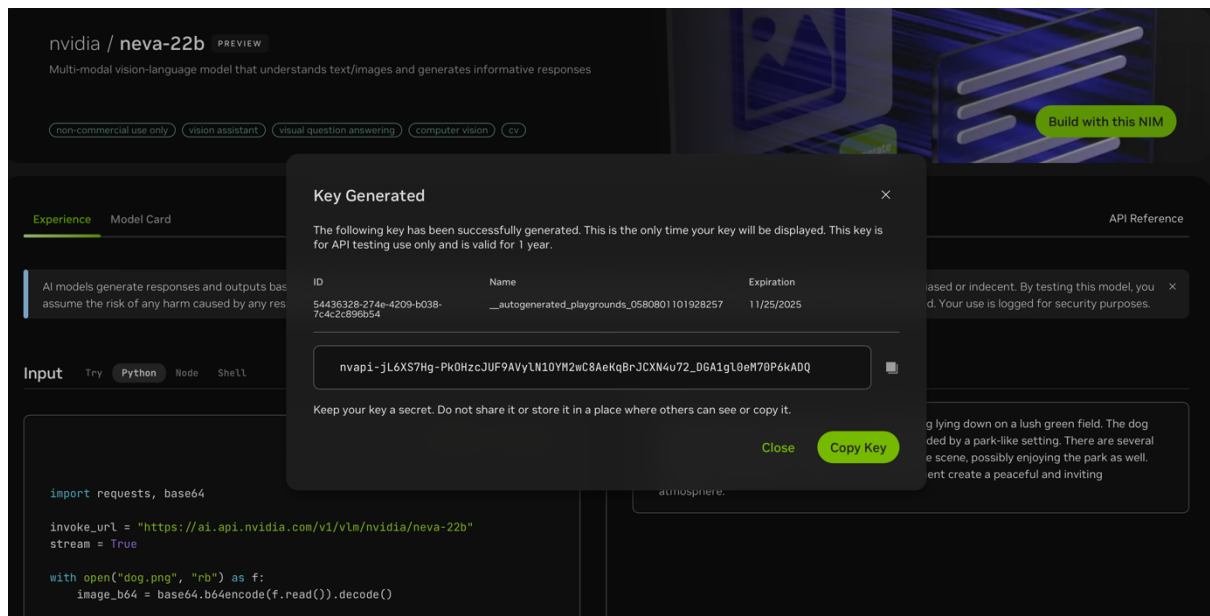
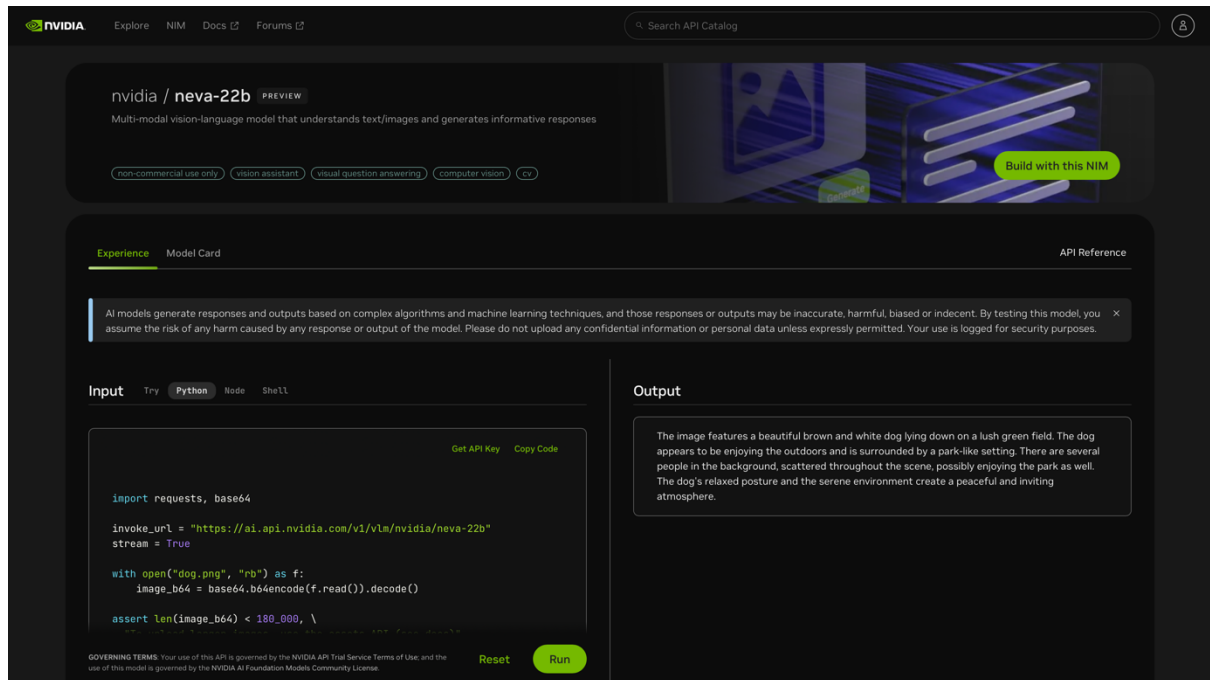
2. The NEVA API processes the input (e.g., Base64-encoded video frames) and returns a response indicating whether the specified action (e.g., "sitting") is detected in the frame.

Benefits in Our Application

- **Action Recognition:** NEVA-22b combines visual and textual understanding to detect complex human actions in videos.
- **Accuracy:** It delivers robust performance on real-world and synthetic data by leveraging NVIDIA's pre-trained AI models.

- **Scalability:** The cloud-based nature of NEVA allows scaling to process multiple videos simultaneously without hardware limitations.

The integration of NVIDIA NEVA-22b through its API key transforms the workflow into a powerful tool for action detection and evaluation, enabling real-time analysis and comparative insights.



CODE

```
[2] !pip install gradio

Collecting gradio
  Downloading gradio-5.6.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<24.0,>=22.0 (from gradio)
  Downloading aiofiles-23.2.1-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.7.1)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.5-py3-none-any.whl.metadata (27 kB)
Collecting ffmpeg (from gradio)
  Downloading ffmpeg-0.4.0-py3-none-any.whl.metadata (2.9 kB)
Collecting gradio-client==1.4.3 (from gradio)
  Downloading gradio_client-1.4.3-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: httpx<0.24.1 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.27.2)
Requirement already satisfied: huggingface-hub<0.25.1 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.26.2)
Requirement already satisfied: Jinja2<4.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.1.4)
Collecting MarkupSafe<2.0 (from gradio)
  Downloading MarkupSafe-2.1.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.0 kB)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (1.26.4)
Requirement already satisfied: orjson<3.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.10.11)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (11.0.0)
Requirement already satisfied: pydantic<2.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (2.9.2)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart==0.0.12 (from gradio)
  Downloading python_multipart-0.0.12-py3-none-any.whl.metadata (1.9 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (6.0.2)
Collecting ruff<0.2.2 (from gradio)
  Downloading ruff-0.8.0-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<1.0,>=0.1.1 (from gradio)
  Downloading safehttpx-0.1.1-py3-none-any.whl.metadata (4.1 kB)
Collecting semantic-version<2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.41.3-py3-none-any.whl.metadata (6.0 kB)
Collecting tomlkit==0.12.0 (from gradio)
  Downloading tomlkit-0.12.0-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.13.0)
Requirement already satisfied: typing-extensions<4.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (4.12.2)
Collecting uvicorn<0.14.0 (from gradio)
  Downloading uvicorn-0.32.1-py3-none-any.whl.metadata (6.6 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from gradio-client==1.4.3->gradio) (2024.10.0)
Collecting websockets<13.0,>=10.0 (from gradio-client==1.4.3->gradio)
  Downloading websockets-12.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux2014_x86_64.whl.metadata (6.6 kB)
Requirement already satisfied: idna<2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
15s completed at 10:00 PM
```

```
Group5_AI4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 3:01 PM

+ Code + Text
import requests
import base64
from moviepy.video.io.VideoFileClip import VideoFileClip
from PIL import Image
from io import BytesIO
import gradio as gr

API_LINK = "https://ai.api.nvidia.com/v1/vlm/nvidia/eva-22b"
KEY = "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"

def get_video_frames(video_path, frames_count=16):
    clip = VideoFileClip(video_path)
    time_length = clip.duration
    frames = []
    clip.get_frame(i * time_length / frames_count) for i in range(frames_count)
    return [Image.fromarray(frame) for frame in frames]

def image_to_base64(image_frame):
    temp_buffer = BytesIO()
    image_frame.save(temp_buffer, format="PNG")
    return base64.b64encode(temp_buffer.getvalue()).decode()

def detect_action(image_b64, activity):
    headers = {
        "Authorization": f"Bearer {KEY}",
        "Accept": "application/json",
    }
    info = {
        "messages": [
            {
                "role": "user",
                "content": f"👉 Do you see someone performing \"{activity}\" in this image? <img src='data:image/png;base64,{image_b64}' />",
            }
        ],
        "max_tokens": 1024,
        "temperature": 0.20,
        "top_p": 0.70,
        "seed": 0,
        "stream": False,
    }
    response = requests.post(API_LINK, headers=headers, json=info)
```

Connected to Python 3 Google Compute Engine backend

```
Group5_AI4.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes

Files
-
sample_data

Code
15s
response = requests.post(API_LINK, headers=headers, json=info)
result = response.json()
return "yes" in result.get("choices", []) and result.get("message", {}).get("content", "").lower()

def calculate_accuracy(video_frames, activity):
    detections = 0
    for image in video_frames:
        image_b64 = image_to_base64(image)
        if detect_action(image_b64, activity):
            detections += 1
    return (detections / len(video_frames)) * 100

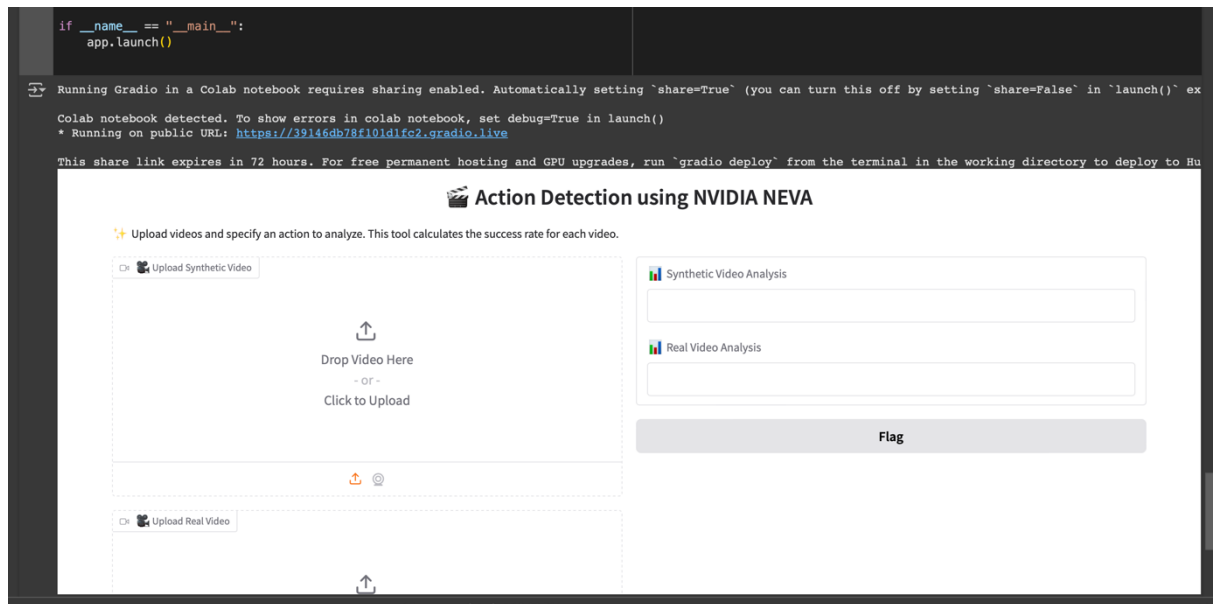
def analyze_videos(video1, video2, activity):
    try:
        frames_video1 = get_video_frames(video1)
        frames_video2 = get_video_frames(video2)
        result1 = calculate_accuracy(frames_video1, activity)
        result2 = calculate_accuracy(frames_video2, activity)

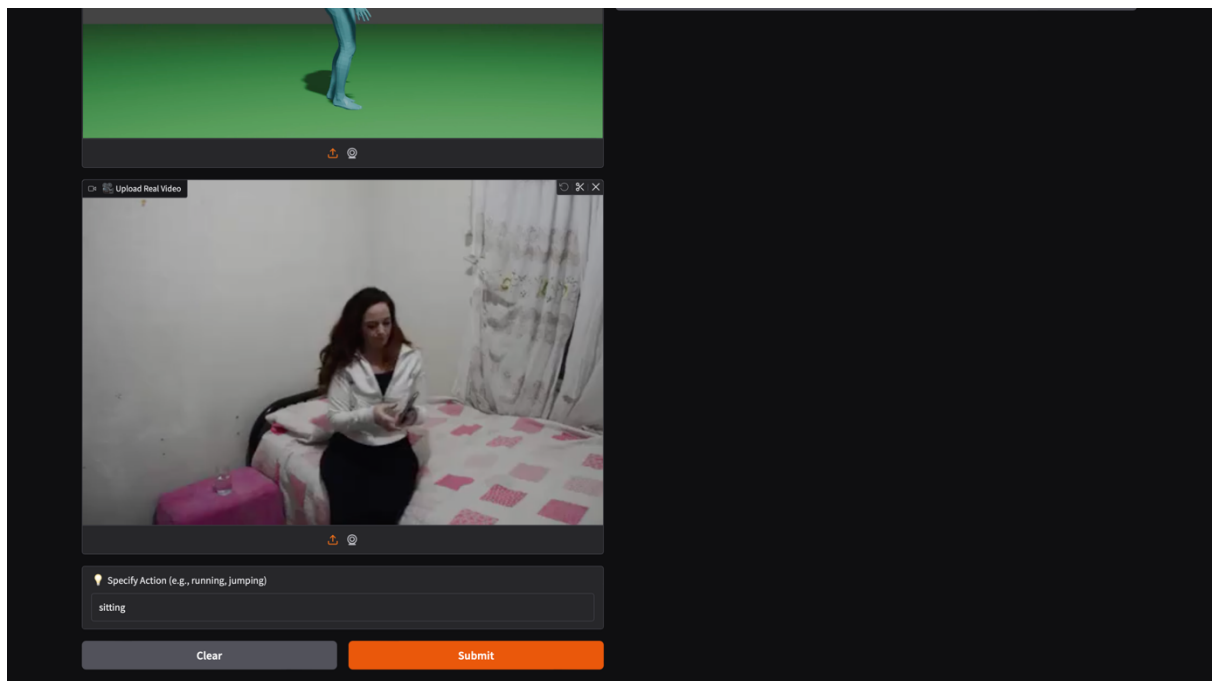
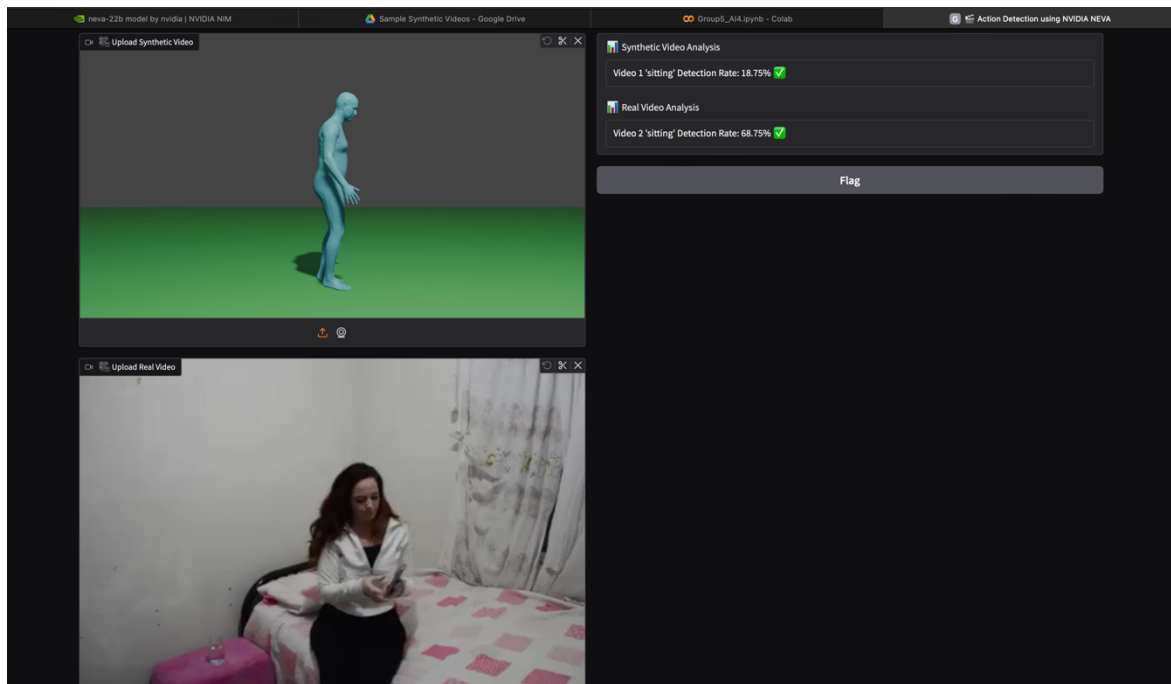
        return (
            f"Video 1 '{activity}' Detection Rate: {result1:.2f}% ✓",
            f"Video 2 '{activity}' Detection Rate: {result2:.2f}% ✓",
        )
    except Exception as error:
        return f"Oops! Something went wrong: {str(error)}", None

app = gr.Interface(
    fn=analyze_videos,
    inputs=[
        gr.Video(label="Upload Synthetic Video"),
        gr.Video(label="Upload Real Video"),
        gr.Textbox(label="Specify Action (e.g., running, jumping)"),
    ],
    outputs=[
        gr.Textbox(label="Synthetic Video Analysis"),
        gr.Textbox(label="Real Video Analysis"),
    ],
    title="Action Detection using NVIDIA NEVA",
    description="Upload videos and specify an action to analyze. This tool calculates the success rate for each video."
)

if __name__ == "__main__":
    app.launch()
```

OUTPUT





The output interface allows users to upload two videos (synthetic and real) and specify an action to detect. It processes the videos by extracting 16 evenly spaced frames from each, sends them to NVIDIA NEVA for analysis, and calculates the success rate of detecting the specified action in each video. The results are displayed as follows:

- Synthetic Video Analysis: Shows the percentage of frames in the synthetic video where the specified action (e.g., sitting) was detected.
- Real Video Analysis: Shows the percentage of frames in the real video where the specified action was detected.

For the given input, the tool detected 'sitting' with a success rate of 18.75% in the synthetic video and 68.75% in the real video. This indicates that the model performs significantly better on real-world data compared to synthetic content.

CONCLUSION

The implemented solution meets all assignment requirements, providing a robust and user-friendly workflow for human action recognition. The modular design ensures flexibility and scalability for future applications.