No, **you do not need GPUs** for extracting pose coordinates from a **5MB video** using **MediaPipe**. MediaPipe is optimized for real-time inference on **CPUs**, and for small to medium-sized videos, CPU performance is generally sufficient.

**🏆 When Do You Need a GPU for MediaPipe?**

❌ **Not Needed (CPU is Enough)**

* Processing small or medium-sized videos (e.g., a 5MB file).
* Running pose estimation on a single video at a time.
* When real-time processing speed is not a constraint.

✅ **GPU is Beneficial When:**

* Processing large, high-resolution videos (e.g., 4K or long-duration files).
* Running multiple pose extraction tasks in parallel.
* Need ultra-fast real-time processing (e.g., live-stream pose detection).

**⚡ Recommended Setup for Your Case**

* Since your **5MB video** is relatively small, you can run **MediaPipe on CPU** for efficient extraction.

**✅ Force MediaPipe to Use CPU**

To ensure that your server does not use the GPU unnecessarily, modify the script:

import os

os.environ["CUDA\_VISIBLE\_DEVICES"] = "-1" # Disable GPU usage for TensorFlow if installed

Place this at the top of your extract\_pose.py script.

**🚀 Optimized MediaPipe Pose Extraction for CPU**

Here’s the full CPU-optimized script:

import os

import cv2

import mediapipe as mp

import numpy as np

# Force CPU usage

os.environ["CUDA\_VISIBLE\_DEVICES"] = "-1"

# Initialize MediaPipe Pose Model

mp\_pose = mp.solutions.pose

pose = mp\_pose.Pose()

# Load Video

video\_path = "input\_video.mp4" # Replace with your file

cap = cv2.VideoCapture(video\_path)

if not cap.isOpened():

print("Error: Cannot open video.")

exit()

coordinates = []

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

# Convert to RGB

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Run Pose Detection

results = pose.process(frame\_rgb)

if results.pose\_landmarks:

landmarks = results.pose\_landmarks.landmark

frame\_coords = [

[landmarks[mp\_pose.PoseLandmark.LEFT\_SHOULDER.value].x,

landmarks[mp\_pose.PoseLandmark.LEFT\_SHOULDER.value].y,

landmarks[mp\_pose.PoseLandmark.LEFT\_SHOULDER.value].z],

[landmarks[mp\_pose.PoseLandmark.LEFT\_ELBOW.value].x,

landmarks[mp\_pose.PoseLandmark.LEFT\_ELBOW.value].y,

landmarks[mp\_pose.PoseLandmark.LEFT\_ELBOW.value].z],

[landmarks[mp\_pose.PoseLandmark.LEFT\_WRIST.value].x,

landmarks[mp\_pose.PoseLandmark.LEFT\_WRIST.value].y,

landmarks[mp\_pose.PoseLandmark.LEFT\_WRIST.value].z]

]

coordinates.append(frame\_coords)

cap.release()

# Save extracted coordinates

coordinates\_np = np.array(coordinates)

np.save("pose\_data.npy", coordinates\_np)

print(f"Pose extraction complete. Data saved as 'pose\_data.npy'. Shape: {coordinates\_np.shape}")

**🎯 Key Takeaways**

* **For a 5MB video, GPU is not required**—MediaPipe works efficiently on CPU.
* **Disabling GPU (os.environ["CUDA\_VISIBLE\_DEVICES"] = "-1")** ensures CPU-only processing.
* **Processing speed:** On a modern CPU, pose extraction should take just a few seconds for a short video.

**Random arm swing video from uTube**

short video clips specifically capturing arm swings in teenagers with cerebral palsy. While such specific content is limited, here are some resources that may be helpful:

1. **Children with Bilateral Cerebral Palsy | Figueiredo | DMCN**

* *Description:* This video discusses hand–arm bimanual intensive therapy and daily functioning of children with bilateral cerebral palsy, which may include demonstrations of arm movements.
* [Watch on YouTube](https://www.youtube.com/watch?v=Gq2gdjUm4jw)

2. **Brothers in Arms: Beating Cerebral Palsy | Australia**

* *Description:* This video features two inspiring brothers who share a special bond and showcases their experiences living with cerebral palsy, including their physical activities.
* [Watch on YouTube](https://www.youtube.com/watch?v=q7X-_1iPPTQ)

Additionally, the **Gait Patterns in Cerebral Palsy Fact Sheet** offers insights into various movement patterns associated with cerebral palsy, which might provide context for the arm movements you’re interested in.

For more specific footage, consider reaching out to medical institutions, therapy centers, or organizations specializing in cerebral palsy, as they may have more targeted resources or be able to provide guidance on where to find such materials.

Here are some resources that provide visual demonstrations of arm movements:

1. **Understanding Arm Movements in Walking**

* *Description:* This video delves into the mechanics of arm movements during walking, offering insights into natural arm swings.
* [Watch on YouTube](https://www.youtube.com/watch?v=fgOrVJqYus8)

2. **Basic Arm Swing Patterns**

* *Description:* An introduction to rotational arm swing patterns, highlighting how arms contribute to movement dynamics.
* [Watch on YouTube](https://www.youtube.com/watch?v=EQPtIjJq1kw)

3. **Arm Flow | Home Practice with Stopgap Dance Company**

* *Description:* This session explores arm movements using human anatomy, imagery, and exploration, suitable for all levels.
* [Watch on YouTube](https://www.youtube.com/watch?v=NkstTdp0s5g)

**Record a portion of the YouTube video** while watching it using **built-in screen recording tools** or third-party software. Here are the best methods depending on your device:

**🎥 Mac (Built-in Screen Recording)**

✅ **No Extra Software Needed**

1. Open the YouTube video.
2. Press **Cmd + Shift + 5** to open the screen recording tool.
3. Select **Record Selected Portion** (to record only the YouTube player).
4. Click **Record** and play the part of the video you need.
5. Click the **stop button** in the menu bar or press **Cmd + Shift + 5** again to stop.
6. The recording will be saved in your Movies or Desktop folder.

**🎥 iPhone/iPad (Built-in Screen Recording)**

✅ **Perfect for Mobile Users**

1. Open **Settings > Control Center > Add Screen Recording** (if not already enabled).
2. Open YouTube and play the video.
3. Swipe down (or up on older iPhones) and tap **Screen Recording**.
4. Stop recording by tapping the red bar or Control Center button.
5. The video will be saved in the **Photos app**.

**🎥 OBS Studio (Advanced, Windows/Mac/Linux)**

✅ **Best for High-Quality or Customized Recording**

1. Download [**OBS Studio**](https://obsproject.com/) (free software).
2. Add a **Display Capture** source to record your screen.
3. Adjust settings (resolution, frame rate, audio, etc.).
4. Click **Start Recording** and play the video.
5. Click **Stop Recording** to save the file.

**🔥 Best Choice for You?**

* **For Mobile Users**: Use built-in screen recording on **iPhone/Android**.
* **For Advanced Control**: **OBS Studio**.

Step-by-step to extract **arm coordinates** using **MediaPipe** on your **Mac terminal**. 🚀

**📌 Step 1: Install & Set Up Python Virtual Environment**

1️⃣ **Update Homebrew (if you haven’t recently)**

brew update

2️⃣ **Install Python3 & Required Dependencies**

brew install python3 ffmpeg

3️⃣ **Create a Virtual Environment (Recommended)**

python3 -m venv mediapipe\_env

source mediapipe\_env/bin/activate

You should now see (mediapipe\_env) at the beginning of your terminal prompt. This means the virtual environment is activated.

4️⃣ **Upgrade Pip & Install MediaPipe + Dependencies**

pip install --upgrade pip

pip install mediapipe opencv-python numpy

**📌 Step 2: Verify Your Video File**

1️⃣ **Check if your video is in the correct directory**

ls

You should see your **video file** (e.g., input\_video.mp4). If it’s in another folder, move it to the current directory:

mv /path/to/your/video.mp4 ./

2️⃣ **Check Video Properties**

ffmpeg -i input\_video.mp4

This will display details like **resolution, FPS, and duration**.

**📌 Step 3: Create Python Script for Pose Extraction**

1️⃣ **Open a text editor and create the script**

nano extract\_pose.py

2️⃣ **Paste this MediaPipe Pose Extraction Code**

import cv2

import mediapipe as mp

import numpy as np

import os

# Disable GPU for CPU-based processing

os.environ["CUDA\_VISIBLE\_DEVICES"] = "-1"

# Initialize MediaPipe Pose Model

mp\_pose = mp.solutions.pose

pose = mp\_pose.Pose()

# Load Video

video\_path = "input\_video.mp4" # Ensure your video file is in the same folder

cap = cv2.VideoCapture(video\_path)

if not cap.isOpened():

print("Error: Cannot open video.")

exit()

coordinates = []

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break # Exit loop if video ends

# Convert frame to RGB (MediaPipe requires RGB input)

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Run Pose Detection

results = pose.process(frame\_rgb)

if results.pose\_landmarks:

landmarks = results.pose\_landmarks.landmark

# Extract Shoulder, Elbow, and Wrist coordinates

frame\_coords = [

[landmarks[mp\_pose.PoseLandmark.LEFT\_SHOULDER.value].x,

landmarks[mp\_pose.PoseLandmark.LEFT\_SHOULDER.value].y,

landmarks[mp\_pose.PoseLandmark.LEFT\_SHOULDER.value].z],

[landmarks[mp\_pose.PoseLandmark.LEFT\_ELBOW.value].x,

landmarks[mp\_pose.PoseLandmark.LEFT\_ELBOW.value].y,

landmarks[mp\_pose.PoseLandmark.LEFT\_ELBOW.value].z],

[landmarks[mp\_pose.PoseLandmark.LEFT\_WRIST.value].x,

landmarks[mp\_pose.PoseLandmark.LEFT\_WRIST.value].y,

landmarks[mp\_pose.PoseLandmark.LEFT\_WRIST.value].z]

]

coordinates.append(frame\_coords)

cap.release()

# Convert to NumPy array and save results

coordinates\_np = np.array(coordinates)

np.save("pose\_data.npy", coordinates\_np)

print(f"✅ Pose extraction complete. Data saved as 'pose\_data.npy'. Shape: {coordinates\_np.shape}")

3️⃣ **Save and Exit** (in nano editor)

* + Press Ctrl + X
  + Press Y to confirm saving
  + Press Enter

**📌 Step 4: Run the Script**

Now, execute the script to extract arm coordinates:

python extract\_pose.py

✅ **Expected Output:**

* It should process each video frame and save **arm joint coordinates** in pose\_data.npy.

**📌 Step 5: Check Extracted Pose Data**

To verify the extracted data:

python -c "import numpy as np; data = np.load('pose\_data.npy'); print(data.shape)"

This should output something like:

(300, 3, 3) # Example: 300 frames, 3 joints (shoulder, elbow, wrist), 3D coordinates (x, y, z)

**📌 Step 6 (Optional): Convert Extracted Data to CSV for Easy Viewing**

If you want to save the extracted coordinates into a CSV file for easier analysis:

import numpy as np

import pandas as pd

# Load the extracted pose data

pose\_data = np.load("pose\_data.npy")

# Flatten into a DataFrame

df = pd.DataFrame(pose\_data.reshape(pose\_data.shape[0], -1),

columns=["shoulder\_x", "shoulder\_y", "shoulder\_z",

"elbow\_x", "elbow\_y", "elbow\_z",

"wrist\_x", "wrist\_y", "wrist\_z"])

# Save to CSV

df.to\_csv("pose\_data.csv", index=False)

print("✅ Pose data saved as 'pose\_data.csv'")

Run this script:

python convert\_pose\_to\_csv.py

Now, pose\_data.csv can be opened in Excel or Google Sheets for easier visualization.

**🎯 Next Steps**

* ✅ Extracted arm pose coordinates from video 📽️
* ✅ Verified data with .npy and .csv formats

**MediaPipe supports .mov videos**, but to ensure compatibility, it’s a good idea to convert the .mov file to .mp4 using **FFmpeg**, which is already installed from our setup.

**🔄 Step 1: Convert .mov to .mp4**

Run the following command in your **Mac terminal**:

ffmpeg -i input\_video.mov -q:v 1 -q:a 1 output\_video.mp4

• **input\_video.mov** → Your original file.

• **output\_video.mp4** → The converted file.

• **-q:v 1 -q:a 1** → Ensures high-quality video and audio.

✅ **Check if the conversion was successful**:

ls -lh output\_video.mp4

This should list the file details.

**🔧 Step 2: Run the Pose Extraction Script with the .mp4 File**

Now, update your **extract\_pose.py** script to use the .mp4 file:

video\_path = "output\_video.mp4" # Updated filename

Run the script:

python extract\_pose.py

**📌 Alternative: Directly Use .mov in MediaPipe**

If you **don’t** want to convert the file, you can modify OpenCV’s VideoCapture to read .mov files:

cap = cv2.VideoCapture("input\_video.mov", cv2.CAP\_FFMPEG)

However, some MacOS versions **may not support .mov directly**, so converting to .mp4 is the best approach.

**🎯 Summary**

* 1️⃣ Convert .mov to .mp4 using **FFmpeg**.
* 2️⃣ Update extract\_pose.py to use output\_video.mp4.
* 3️⃣ Run **pose extraction** as usual.

**Pandas installation** inside your virtual environment (mediapipe\_env).

**✅ Step 1: Install Pandas in Your Virtual Environment**

Since you’re already in the mediapipe\_env, run:

pip install pandas

**✅ Step 2: Verify the Installation**

Once installed, confirm that pandas is available:

python -c "import pandas as pd; print(pd.\_\_version\_\_)"

This should return the installed version (e.g., 1.5.3).

**✅ Step 3: Retry Your Script**

Now, re-run your script:

python convert\_pose\_to\_csv.py

**🎯 Summary**

* 1️⃣ Activate the virtual environment (if not already active):

source mediapipe\_env/bin/activate

* 2️⃣ Install Pandas:

pip install pandas

* 3️⃣ Verify installation:

python -c "import pandas as pd; print(pd.\_\_version\_\_)"

* 4️⃣ Run your script again:

python convert\_pose\_to\_csv.py

Visualize the **arm movement trends** using **Matplotlib and Pandas**. Here’s how you can generate **plots** and analyze movement:

**🎯 Step 1: Install Matplotlib**

If you haven’t already, install Matplotlib inside your **virtual environment**:

pip install matplotlib pandas

**📊 Step 2: Load and Plot the Data**

Now, create a **Python script** (visualize\_pose.py) or run these commands in a Python shell:

import pandas as pd

import matplotlib.pyplot as plt

# Load pose data

df = pd.read\_csv("pose\_data.csv")

# Extract coordinate columns

frames = range(len(df))

shoulder\_x, shoulder\_y = df["shoulder\_x"], df["shoulder\_y"]

elbow\_x, elbow\_y = df["elbow\_x"], df["elbow\_y"]

wrist\_x, wrist\_y = df["wrist\_x"], df["wrist\_y"]

# Plot movements over frames

plt.figure(figsize=(10, 5))

plt.plot(frames, shoulder\_y, label="Shoulder Y", linestyle="dashed")

plt.plot(frames, elbow\_y, label="Elbow Y", linestyle="solid")

plt.plot(frames, wrist\_y, label="Wrist Y", linestyle="dotted")

plt.xlabel("Frame Number")

plt.ylabel("Y-Coordinate")

plt.title("Arm Movement Over Time")

plt.legend()

plt.show()

**📌 Explanation of the Graph**

* The **X-axis** represents **frame numbers** (time in video).
* The **Y-axis** represents the **vertical (Y) movement** of the **shoulder, elbow, and wrist**.
* Different **line styles** show different joint movements.
* This helps analyze **motion trends**, like how much the wrist moves compared to the elbow.

**🎥 Optional: Animate the Movement**

If you want to **animate the movement**, use Matplotlib’s **animation module**:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.animation as animation

# Load pose data

df = pd.read\_csv("pose\_data.csv")

fig, ax = plt.subplots()

ax.set\_xlim(min(df["shoulder\_x"]), max(df["wrist\_x"]))

ax.set\_ylim(min(df["shoulder\_y"]), max(df["shoulder\_y"]))

# Initialize scatter plot

point, = ax.plot([], [], 'ro', markersize=10) # Red dots for joints

def update(frame):

x\_values = [df["shoulder\_x"][frame], df["elbow\_x"][frame], df["wrist\_x"][frame]]

y\_values = [df["shoulder\_y"][frame], df["elbow\_y"][frame], df["wrist\_y"][frame]]

point.set\_data(x\_values, y\_values)

return point,

ani = animation.FuncAnimation(fig, update, frames=len(df), interval=50)

plt.show()

🔹 **This will animate the arm motion across frames!** 🎥✨

**🎯 Next Steps**

* ✅ **Basic Y-coordinate movement visualization** (done).
* ✅ **Optional: Animate movement** using scatter plot.

**Overlay the extracted pose coordinates onto the original .mp4 video**, we will use **OpenCV** to draw the animated pose on each frame and save it as a new video. 🎥✨

**✅ Step 1: Install Dependencies**

Make sure you have **OpenCV** installed in your virtual environment:

pip install opencv-python opencv-python-headless pandas numpy

**✅ Step 2: Modify animate\_pose.py to Overlay Pose on Video**

Create a new script or modify your existing animate\_pose.py to **read frames from the video** and **draw pose points** on each frame.

**📜 Save this as overlay\_pose\_on\_video.py**

import cv2

import pandas as pd

import numpy as np

# Load pose data

df = pd.read\_csv("pose\_data.csv")

# Open the original video

video\_path = "output\_video.mp4" # Ensure this matches your actual video file

cap = cv2.VideoCapture(video\_path)

# Get video properties

frame\_width = int(cap.get(3))

frame\_height = int(cap.get(4))

fps = int(cap.get(cv2.CAP\_PROP\_FPS))

# Define output video writer

output\_path = "pose\_overlay.mp4"

fourcc = cv2.VideoWriter\_fourcc(\*'mp4v') # Use 'XVID' if 'mp4v' doesn't work

out = cv2.VideoWriter(output\_path, fourcc, fps, (frame\_width, frame\_height))

# Frame counter

frame\_idx = 0

while cap.isOpened():

ret, frame = cap.read()

if not ret or frame\_idx >= len(df):

break # Exit loop when video ends or no more pose data

# Extract pose data for this frame

shoulder = (int(df.iloc[frame\_idx]["shoulder\_x"] \* frame\_width), int(df.iloc[frame\_idx]["shoulder\_y"] \* frame\_height))

elbow = (int(df.iloc[frame\_idx]["elbow\_x"] \* frame\_width), int(df.iloc[frame\_idx]["elbow\_y"] \* frame\_height))

wrist = (int(df.iloc[frame\_idx]["wrist\_x"] \* frame\_width), int(df.iloc[frame\_idx]["wrist\_y"] \* frame\_height))

# Draw circles for joints

cv2.circle(frame, shoulder, 5, (0, 255, 0), -1) # Green for shoulder

cv2.circle(frame, elbow, 5, (0, 0, 255), -1) # Red for elbow

cv2.circle(frame, wrist, 5, (255, 0, 0), -1) # Blue for wrist

# Draw lines connecting the joints

cv2.line(frame, shoulder, elbow, (0, 255, 255), 2)

cv2.line(frame, elbow, wrist, (255, 0, 255), 2)

# Write the modified frame to output video

out.write(frame)

frame\_idx += 1 # Move to next frame

# Release resources

cap.release()

out.release()

cv2.destroyAllWindows()

print(f"✅ Pose overlay completed. Video saved as {output\_path}")

**✅ Step 3: Run the Script**

Once the script is saved, execute it:

python overlay\_pose\_on\_video.py

**📌 Expected Output**

* The script will read each video frame, overlay the **pose coordinates**, and save a new video called **pose\_overlay.mp4**.
* The **joints (shoulder, elbow, wrist)** will be color-coded and connected by lines.

**🎥 Next Steps**

* ✅ Overlay pose coordinates onto the video.
* ✅ Save the result as **pose\_overlay.mp4**.

**✅ How to Play the .mp4 Video in Your Terminal Directory on Mac**

**🔹 Option 1: Open Video Using QuickTime (GUI)**

Run the following command in your terminal:

open pose\_overlay.mp4

This will open the video in **QuickTime Player** (Mac’s default media player).

**🔹 Option 2: Play Video in Terminal Using mpv (Recommended)**

1️⃣ Install mpv (if not installed):

brew install mpv

2️⃣ Play the video:

mpv pose\_overlay.mp4

**🔹 Option 3: Play Video Using ffplay (From FFmpeg)**

1️⃣ If ffmpeg is installed, you can use ffplay:

ffplay pose\_overlay.mp4

**🔹 Option 4: Open in VLC Player**

1️⃣ If VLC is installed:

open -a VLC pose\_overlay.mp4

**🎯 Best Option?**

* **QuickTime (open pose\_overlay.mp4)** → Easiest for GUI playback.
* **mpv pose\_overlay.mp4** → Best for in-terminal playback.
* **ffplay pose\_overlay.mp4** → Lightweight and works on most systems.

To **capture both right and left arms**, you need to modify your **pose extraction script** to save **both sides** (shoulder, elbow, wrist). Then, visualize them separately or together in the overlay.

**✅ Step 1: Update Pose Extraction to Capture Both Arms**

Modify your **extract\_pose.py** script to include **both right and left arm joints**.

**📜 Updated extract\_pose.py**

import cv2

import mediapipe as mp

import numpy as np

import os

# Disable GPU (for CPU processing)

os.environ["CUDA\_VISIBLE\_DEVICES"] = "-1"

# Initialize MediaPipe Pose Model

mp\_pose = mp.solutions.pose

pose = mp\_pose.Pose()

# Load Video

video\_path = "output\_video.mp4"

cap = cv2.VideoCapture(video\_path)

if not cap.isOpened():

print("Error: Cannot open video.")

exit()

coordinates = []

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break # Exit loop when video ends

# Convert frame to RGB (MediaPipe requires RGB input)

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Run Pose Detection

results = pose.process(frame\_rgb)

if results.pose\_landmarks:

landmarks = results.pose\_landmarks.landmark

# Extract \*\*Right Arm\*\* coordinates

right\_shoulder = [landmarks[mp\_pose.PoseLandmark.RIGHT\_SHOULDER.value].x,

landmarks[mp\_pose.PoseLandmark.RIGHT\_SHOULDER.value].y,

landmarks[mp\_pose.PoseLandmark.RIGHT\_SHOULDER.value].z]

right\_elbow = [landmarks[mp\_pose.PoseLandmark.RIGHT\_ELBOW.value].x,

landmarks[mp\_pose.PoseLandmark.RIGHT\_ELBOW.value].y,

landmarks[mp\_pose.PoseLandmark.RIGHT\_ELBOW.value].z]

right\_wrist = [landmarks[mp\_pose.PoseLandmark.RIGHT\_WRIST.value].x,

landmarks[mp\_pose.PoseLandmark.RIGHT\_WRIST.value].y,

landmarks[mp\_pose.PoseLandmark.RIGHT\_WRIST.value].z]

# Extract \*\*Left Arm\*\* coordinates

left\_shoulder = [landmarks[mp\_pose.PoseLandmark.LEFT\_SHOULDER.value].x,

landmarks[mp\_pose.PoseLandmark.LEFT\_SHOULDER.value].y,

landmarks[mp\_pose.PoseLandmark.LEFT\_SHOULDER.value].z]

left\_elbow = [landmarks[mp\_pose.PoseLandmark.LEFT\_ELBOW.value].x,

landmarks[mp\_pose.PoseLandmark.LEFT\_ELBOW.value].y,

landmarks[mp\_pose.PoseLandmark.LEFT\_ELBOW.value].z]

left\_wrist = [landmarks[mp\_pose.PoseLandmark.LEFT\_WRIST.value].x,

landmarks[mp\_pose.PoseLandmark.LEFT\_WRIST.value].y,

landmarks[mp\_pose.PoseLandmark.LEFT\_WRIST.value].z]

frame\_coords = [right\_shoulder, right\_elbow, right\_wrist,

left\_shoulder, left\_elbow, left\_wrist]

coordinates.append(frame\_coords)

cap.release()

# Convert to NumPy array and save results

coordinates\_np = np.array(coordinates)

np.save("both\_arms\_pose.npy", coordinates\_np)

print(f"✅ Both arms pose extraction complete. Data saved as 'both\_arms\_pose.npy'. Shape: {coordinates\_np.shape}")

**✅ Step 2: Convert Extracted Data to CSV**

To store the **right and left arm movements** in a CSV file:

**📜 Save this as convert\_both\_arms\_to\_csv.py**

import numpy as np

import pandas as pd

# Load the extracted pose data

pose\_data = np.load("both\_arms\_pose.npy")

# Convert to Pandas DataFrame

df = pd.DataFrame(

pose\_data.reshape(pose\_data.shape[0], -1),

columns=["right\_shoulder\_x", "right\_shoulder\_y", "right\_shoulder\_z",

"right\_elbow\_x", "right\_elbow\_y", "right\_elbow\_z",

"right\_wrist\_x", "right\_wrist\_y", "right\_wrist\_z",

"left\_shoulder\_x", "left\_shoulder\_y", "left\_shoulder\_z",

"left\_elbow\_x", "left\_elbow\_y", "left\_elbow\_z",

"left\_wrist\_x", "left\_wrist\_y", "left\_wrist\_z"]

)

# Save to CSV

df.to\_csv("both\_arms\_pose.csv", index=False)

print("✅ Both arms pose data saved as 'both\_arms\_pose.csv'")

**🔧 Run the conversion**

python convert\_both\_arms\_to\_csv.py

**✅ Step 3: Update Overlay Script to Display Both Arms**

Modify your **overlay script (overlay\_pose\_on\_video.py)** to **draw both arms on the video**.

**📜 Updated overlay\_pose\_on\_video.py**

import cv2

import pandas as pd

import numpy as np

# Load both arms pose data

df = pd.read\_csv("both\_arms\_pose.csv")

# Open the original video

video\_path = "output\_video.mp4"

cap = cv2.VideoCapture(video\_path)

# Get video properties

frame\_width = int(cap.get(3))

frame\_height = int(cap.get(4))

fps = int(cap.get(cv2.CAP\_PROP\_FPS))

# Define output video writer

output\_path = "both\_arms\_pose\_overlay.mp4"

fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')

out = cv2.VideoWriter(output\_path, fourcc, fps, (frame\_width, frame\_height))

# Frame counter

frame\_idx = 0

while cap.isOpened():

ret, frame = cap.read()

if not ret or frame\_idx >= len(df):

break # Exit loop when video ends or no more pose data

# Extract right arm pose data

right\_shoulder = (int(df.iloc[frame\_idx]["right\_shoulder\_x"] \* frame\_width), int(df.iloc[frame\_idx]["right\_shoulder\_y"] \* frame\_height))

right\_elbow = (int(df.iloc[frame\_idx]["right\_elbow\_x"] \* frame\_width), int(df.iloc[frame\_idx]["right\_elbow\_y"] \* frame\_height))

right\_wrist = (int(df.iloc[frame\_idx]["right\_wrist\_x"] \* frame\_width), int(df.iloc[frame\_idx]["right\_wrist\_y"] \* frame\_height))

# Extract left arm pose data

left\_shoulder = (int(df.iloc[frame\_idx]["left\_shoulder\_x"] \* frame\_width), int(df.iloc[frame\_idx]["left\_shoulder\_y"] \* frame\_height))

left\_elbow = (int(df.iloc[frame\_idx]["left\_elbow\_x"] \* frame\_width), int(df.iloc[frame\_idx]["left\_elbow\_y"] \* frame\_height))

left\_wrist = (int(df.iloc[frame\_idx]["left\_wrist\_x"] \* frame\_width), int(df.iloc[frame\_idx]["left\_wrist\_y"] \* frame\_height))

# Draw circles for right arm joints

cv2.circle(frame, right\_shoulder, 5, (0, 255, 0), -1) # Green for shoulder

cv2.circle(frame, right\_elbow, 5, (0, 0, 255), -1) # Red for elbow

cv2.circle(frame, right\_wrist, 5, (255, 0, 0), -1) # Blue for wrist

# Draw circles for left arm joints

cv2.circle(frame, left\_shoulder, 5, (0, 255, 0), -1) # Green for shoulder

cv2.circle(frame, left\_elbow, 5, (0, 0, 255), -1) # Red for elbow

cv2.circle(frame, left\_wrist, 5, (255, 0, 0), -1) # Blue for wrist

# Draw lines connecting the joints

cv2.line(frame, right\_shoulder, right\_elbow, (0, 255, 255), 2)

cv2.line(frame, right\_elbow, right\_wrist, (255, 0, 255), 2)

cv2.line(frame, left\_shoulder, left\_elbow, (0, 255, 255), 2)

cv2.line(frame, left\_elbow, left\_wrist, (255, 0, 255), 2)

# Write the modified frame to output video

out.write(frame)

frame\_idx += 1 # Move to next frame

# Release resources

cap.release()

out.release()

cv2.destroyAllWindows()

print(f"✅ Both arms pose overlay completed. Video saved as {output\_path}")

**✅ Step 4: Run the Overlay Script**

python overlay\_pose\_on\_video.py

🎯 **Final Output:**

• A new video **both\_arms\_pose\_overlay.mp4** will show **both arms tracked in real-time**.

**🎯 Final Summary**

* ✅ Extracted **both arms’ movements** from video
* ✅ Saved coordinates in **both\_arms\_pose.csv**
* ✅ Overlaid pose on **both arms** in **both\_arms\_pose\_overlay.mp4**