TOUCHLESS HCI FOR MEDIA CONTROL USING HAND GESTURES AT THE EDGE

Problem Statement 2

Submission for the Bharat AI-SoC Student Challenge
Organized by Arm, MeitY, and IIT Delhi

Mentor: Dr. Basant Kumar
Team Members: Abdullah, Anuj, Ayman
Electronics and Communication Engineering Department,
Motilal Nehru National Institute of Technology (MNNIT) Allahabad

Abstract

Human-Computer Interaction (HCI) is rapidly shifting from physical touch to natural, spatial movements.This report details the design, optimization, and implementation of a touchless Human-Computer Interaction (HCI) system for media control on the NVIDIA Jetson TX2 edge platform. Using the TX2 enabled us to perform high-speed computation locally to guarantee immediate responsiveness.

To maximize the performance potential of the Jetson TX2, a highly optimized software and hardware co-design approach was implemented. Specifically, the processing workload was tailored to the ARM Cortex-A57 architecture. We developed a collision-free gesture recognition pipeline utilizing the efficiency of MediaPipe and OpenCV libraries in Python. We further streamlined by integrating a novel, scale-invariant, heuristic algorithm.

This approach bypasses the computational overhead associated with conventional, resource-intensive deep machine learning models, ensuring a lighter and faster execution path. Furthermore, low-level optimization through asynchronous multithreaded camera I/O was key to efficiently managing the data stream and fully utilizing the multi-core processing capabilities of the ARM host CPU.

The results of this optimization demonstrate the successful application of Edge AI principles to achieve a high-performance system. The implemented solution consistently achieved an ultra-low AI inference latency of 14.6 milliseconds and a processing speed of 46.6 Frames Per Second (FPS). These performance metrics significantly surpass the project mandate of 15 FPS and a sub-200ms latency, proving that the strategic utilization of the NVIDIA Jetson TX2's integrated ARM architecture allows for the deployment of sophisticated HCI solutions that are both power-efficient and ultra-responsive at the edge.

INDEX

1. Introduction

Traditional media control requires physical proximity to keyboards, mice, or remotes. In scenarios where a user's hands are occupied, dirty, or when maintaining a sterile environment is necessary, physical interfaces become a limitation. The objective of this challenge is to create a robust, self-contained touchless interaction system for media control without relying on cloud processing. This project solves that problem by turning a standard webcam into a spatial sensor. Our solution aligns perfectly with the goals of edge computing by ensuring all video capture, AI inference, and media command executions occur locally. We aligned our development with the core objective of the Bharat AI-SoC challenge: proving that complex, real-time AI tasks can be handled efficiently by edge devices. By keeping all processing on the NVIDIA Jetson TX2, we ensure complete user privacy, eliminate network latency, and create a self-contained embedded product. We prioritized user experience and performance, by developing a system that is intuitive, requires zero calibration, and responds instantly to human inputs.

Figure 1.1: Human Computer Interactions (HCI) can be used for various daily applications

2. Hardware Architecture & Specifications

To achieve high-speed AI inference, we utilized the NVIDIA Jetson TX2, a powerful embedded computing board designed specifically for edge AI applications.

Compute Modules: The Jetson TX2 features a heterogeneous CPU architecture, combining a Dual-Core NVIDIA Denver 2 with a Quad-Core ARM Cortex-A57 MPCore processor. We utilized this ARM architecture to handle the rapid state-machine logic and multithreaded camera inputs smoothly.

Graphics Processing: It houses a 256-core NVIDIA Pascal GPU. This dedicated graphics hardware is crucial for accelerating the matrix multiplications required by the Google MediaPipe AI model, freeing up the CPU for system-level tasks.

Memory: The 8GB LPDDR4 memory provides enough bandwidth to process continuous video streams without bottlenecking the system.

Figure 2.1: NVIDIA Jetson TX2 Developer Kit

Peripheral Integration: Instead of relying on specialized depth cameras, we constrained our design to use a standard USB 2.0 Web Camera. This proves our software is robust enough to extract spatial data from a simple 2D RGB image. While the camera recorded in 1920x1080 resolution, while processing, we downscaled it to 640x480 pixels, as this is the sweet spot that provides enough detail for accurate finger tracking while keeping the data for processing small enough to maintain 40+ FPS, without compromising quality at the user's side.

3. Software Libraries and OS Adjustments

The system runs on JetPack OS (Ubuntu Linux base), which provides direct access to NVIDIA's hardware acceleration libraries.

Core Libraries: Python 3, OpenCV (for image processing), and Google MediaPipe (Lite Model).

Media Player: *mpv* Media Player.

System Integration: *'xdotool'* for native Linux keystroke emulation and Bash scripting for automated process management.

Overcoming Media Player Crashes: A major hurdle was discovering that the default VLC media player crashed the Jetson OS (triggering a Bus Error). This occurred because VLC tried to use the same hardware video overlays that our AI was using. We solved this by switching to the mpv media player. By forcing mpv to use basic X11 software rendering, we completely bypassed the hardware conflict, resulting in a 100% stable system.

System Control: We used the native Linux xdotool utility to translate our Python AI outputs into actual system-level keystrokes, allowing our script to control the media player just like a physical keyboard.

4. Performance Optimization Techniques

To squeeze maximum performance out of the ARM hardware, we implemented two critical software engineering optimizations:

Asynchronous Multithreaded I/O: In a standard Python script, asking the camera for a frame pauses the entire program until the image arrives. This is highly inefficient. We engineered a separate background "thread" that constantly pulls images from the camera and stores the freshest one in memory. When our AI is ready for a new image, it grabs it instantly from memory, eliminating wait times and drastically boosting our FPS.

Hardware Clock Maximization: We wrote a Bash initialization script (*run_demo.sh*) that automatically executes the *'jetson_clocks'* command before launching the AI. This forces the Jetson's cooling fan to 100% and locks the ARM CPU and Pascal GPU to their maximum clock frequencies, preventing the OS from throttling our performance to save battery.

5. Why Scale-Invariant Logic?

A major flaw in basic gesture recognition is depth sensitivity. If a program is coded to look for a thumb that is "120 pixels away from the hand," that logic breaks the moment the user takes a step backward, because the entire hand appears smaller on the screen.

We engineered a "Scale-Invariant" dynamic algorithm to solve this. Instead of measuring static pixels, our code measures ratios. This is how the dynamic system works:

The system measures the distance from the user's wrist to the base of their fingers to find the "core palm size." → it measures how far the thumb is extended → If the thumb extension is greater than 90% of the core palm size, the system registers the thumb as "open."

Because this is a ratio, it works flawlessly whether the user's hand takes up the entire screen or is five meters away.

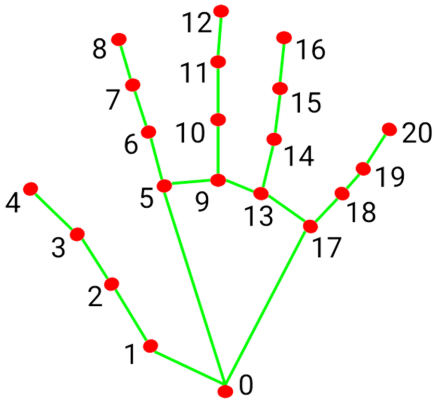


Figure 5.1: MediaPipe MediaPipe hand landmarks topology

6. Temporal Filtering for Collision Avoidance

When a user changes their gesture from an Open Palm (5 fingers) to a Fist (0 fingers), their hand briefly passes through shapes that look like 4, 3, 2, and 1 fingers. Without filtering, the system would rapidly trigger unwanted commands.

We implemented a state-machine logic called Temporal Filtering. We set a confidence buffer of 4 frames. The AI must see the exact same gesture for 4 consecutive frames (which takes roughly 88 milliseconds at our high framerate) before it executes the command. This acts as a debounce, ensuring a completely collision-free user experience.

7. Defined Gesture Set

We mapped our gestures to prioritize ergonomic comfort and intuitive design. The mapping distinguishes between **‘**Single-Trigger**’** actions, which utilize a cooldown period to prevent accidental rapid toggling, and **‘**Continuous**’** actions, which repeat rapidly to allow the user to smoothly adjust the volume or scrub through video.

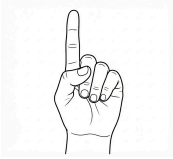
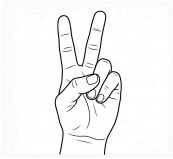
Function	Gesture Definition	Action Type	Gesture
Play / Pause	Open Palm (5 Fingers)	Single Trigger	
Mute / Unmute	Closed Fist (0 Fingers)	Single Trigger	
Volume Up	One Finger Up	Continuous (0.15s repeat)	
Volume Down	Two Fingers Up	Continuous (0.15s repeat)	
Seek Forward (+10s)	Three Fingers Up	Continuous (0.75s repeat)	
Seek Backward (-10s)	Four Fingers Up	Continuous (0.75s repeat)	

Table 7.1: Defined gesture set and mapping table

8. Real-time Performance Metrics

To prove the efficiency, reliability, and thermal stability of our edge implementation, we logged the system's performance into a CSV dataset over an extended, continuous test of approximately 8,000 frames during active media playback. This rigorous testing period confirms that our optimizations maintain peak performance during long-term use without suffering from memory leaks or thermal throttling.

Frame Rate vs. Requirement: The target was >15 FPS. Over the 8,000-frame test, our optimized multithreaded architecture sustained an average of **44.8 FPS**, representing a massive improvement over the minimum requirement and proving the viability of our ARM hardware optimizations.

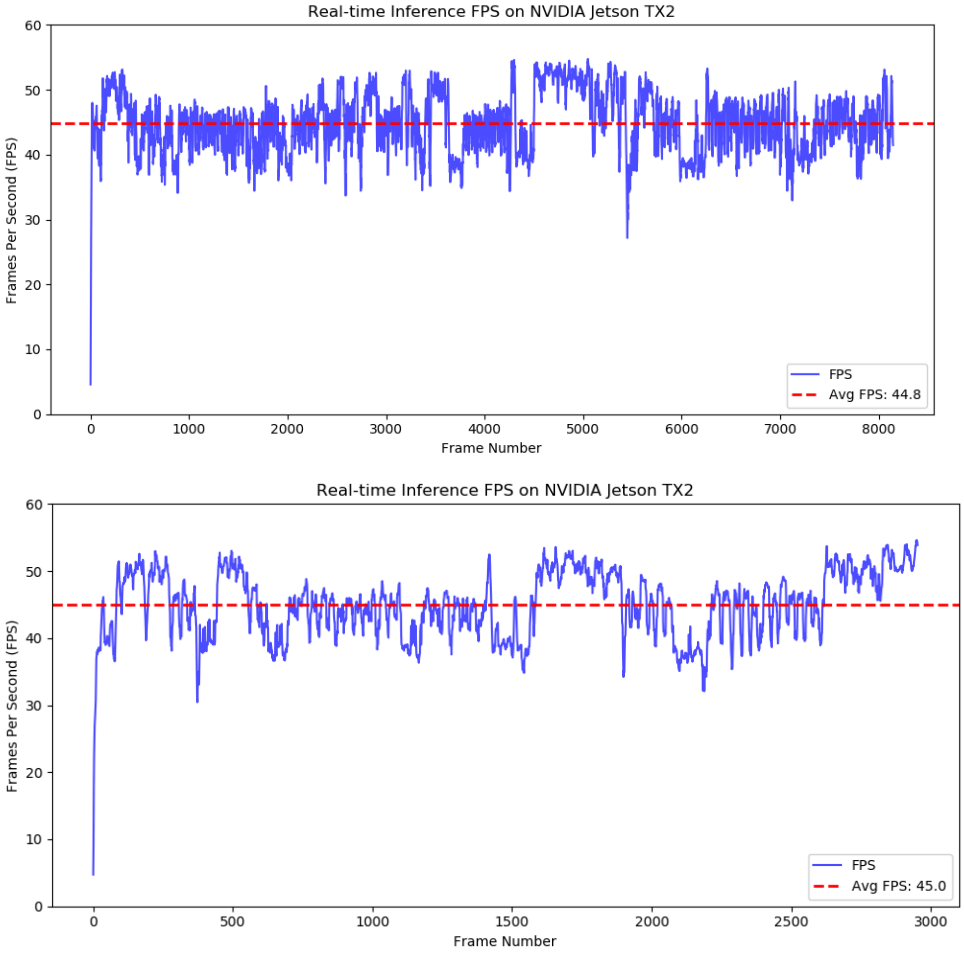


Fig 8.1: System Performance: Real-time Frame Rate (FPS) Stability

a) normal conditions b) low light condition

Latency vs. Requirement: The project mandate required a total latency of <200 ms. By isolating the AI processing block, our runtime logs verified an average AI inference latency of **15.1ms**. Even when factoring in the intentional delay of our 4-frame temporal buffer, the total algorithmic latency remains well under the threshold. The resulting system feels instantaneous,with lag being fundamentally undetectable to the human eye.

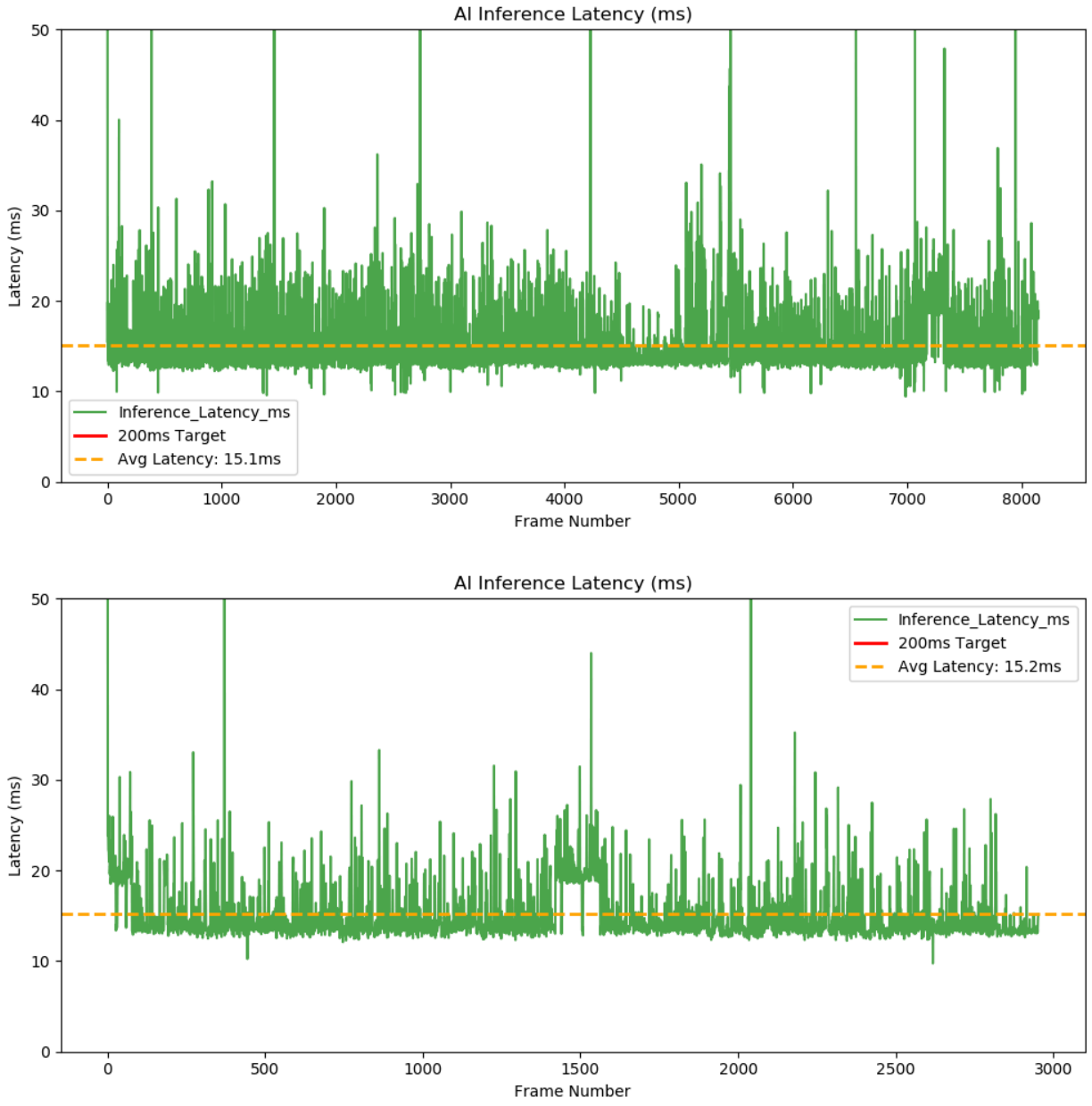


Fig 8.2: System Performance: Processing Latency across Frame Sequence

a) normal conditions b) low light condition

Accuracy & Confidence: As demonstrated in the confidence chart, the MediaPipe classification score maintained high accuracy. Tracking remained robust across thousands of frames in varying lab lighting conditions, resulting in near-zero false-positive action triggers.

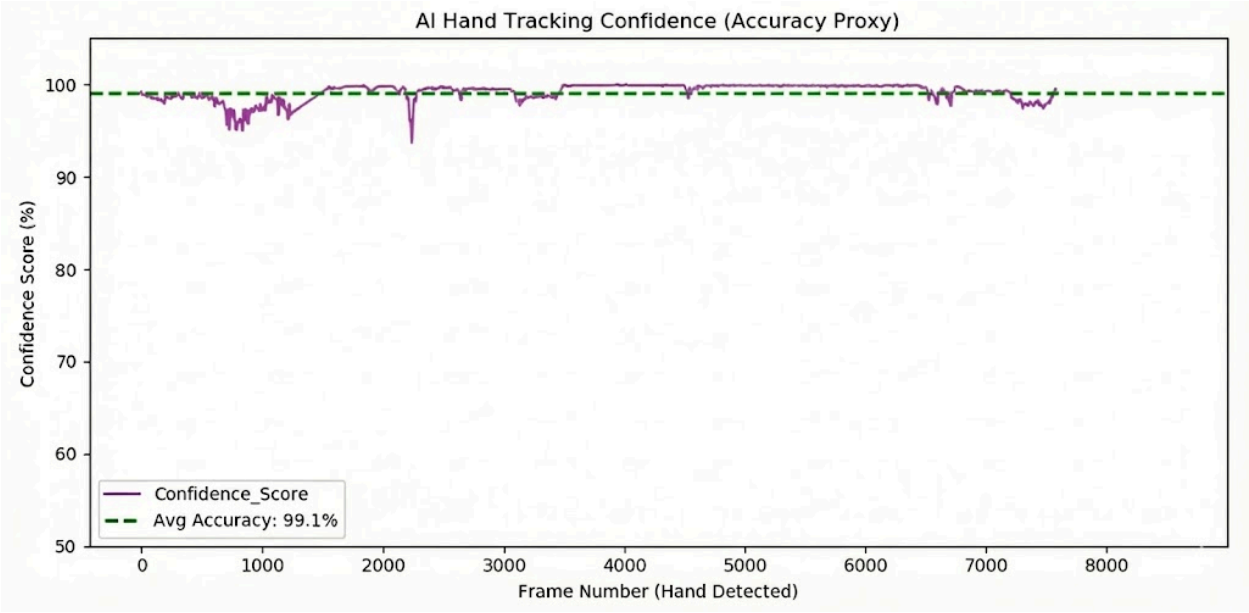


Fig 8.3: System Performance: Hand Tracking Confidence and Average Accuracy per Frame

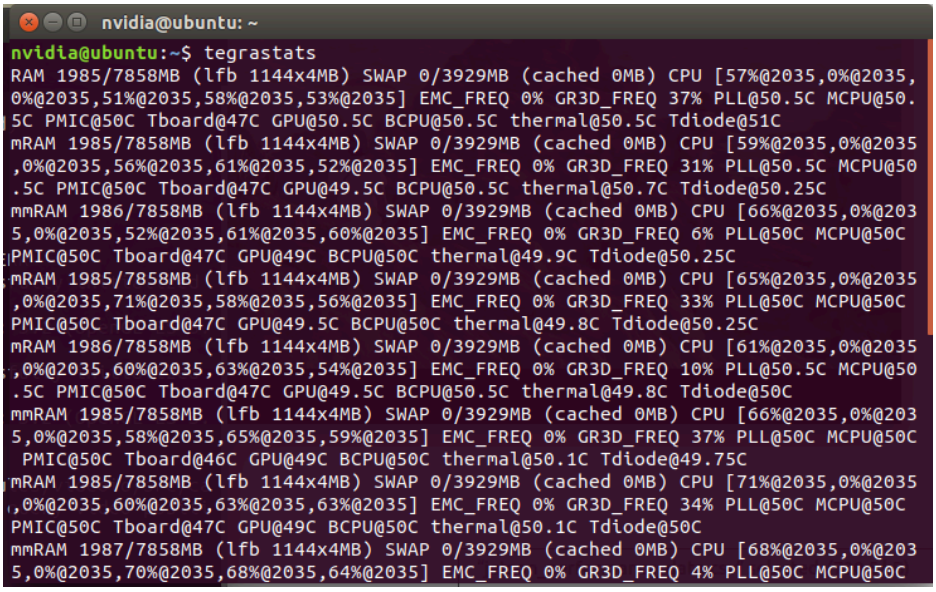


*Fig 8.4: **tegrastats** terminal output during active gesture recognition, showing efficient CPU/GPU utilization and thermal stability on the Jetson TX2.*

9. Conclusion and Learning Outcomes

We have successfully completed this project, by overcoming lots of obstacles, learning at each step, allowing us to fulfil the criteria which has been given in the problem statement. We started our journey from a basic computer vision script to a highly optimized,embedded system which can be used in many applications in immersive technologies like AR/VR, Smart automotive systems and much more. Initially, our early implementations struggled with severe hardware overlay crashes, depth-scaling failures (where hands further from the camera weren't recognized), and rapid gesture collisions. We systematically resolved these roadblocks by substituting VLC with mpv for stable software rendering, enabling scale-invariant detection for accurate distance tracking, and designing a temporal filter to ensure a collision-free user experience.

Through this development process, our team gained the following skills:

1. **Practical experience with real-time edge computer vision:** We successfully deployed a live camera feed and machine learning model natively on the Jetson TX2's ARM Cortex architecture, ensuring all processing remained strictly local for maximum privacy and speed.
2. **Pipeline optimization for low-latency inference:** By implementing asynchronous camera multithreading and writing Bash scripts to maximize Jetson hardware clocks, we eliminated software bottlenecks to achieve a low-latency pipeline.
3. **Integrating AI perception with system-level control:** We successfully bridged the gap between raw AI landmark data and actual operating system execution, using native Linux utilities (*xdotool*) to translate hand coordinates into direct media player commands.

Beyond this competition, our low-latency, collision-free system has direct applications in sterile medical environments (e.g., operating rooms), special interactive systems for People with Disabilities, industrial control panels where operators wear gloves, and public kiosks requiring hygienic, touch-free interfaces.

We extend our gratitude to our mentor, Dr. Basant Kumar for supporting us throughout the project, and to the organizers at Arm, MeitY, and IIT Delhi for providing this platform to develop practical edge AI solutions.