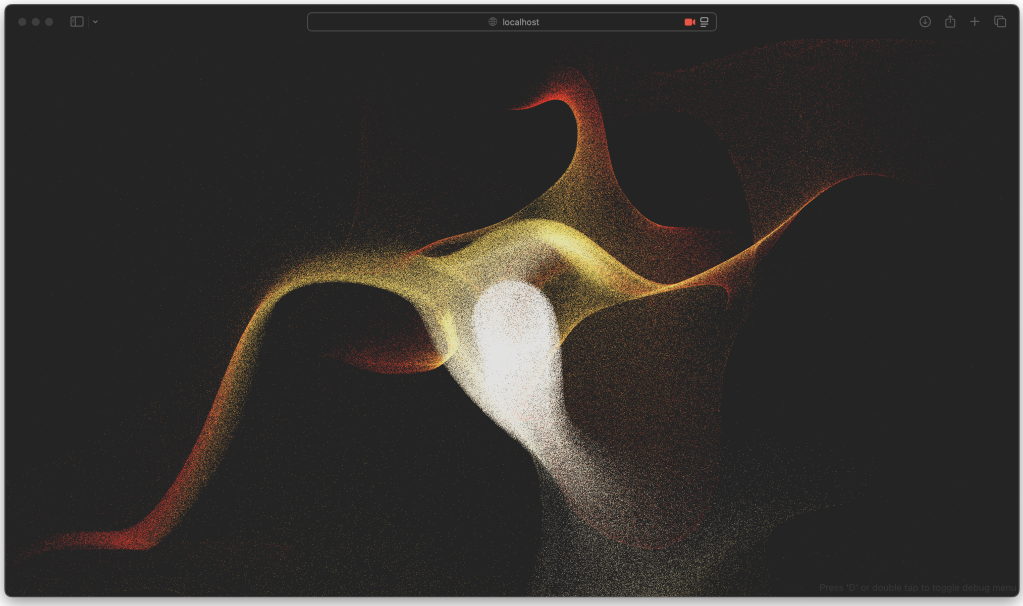


DJ Pose Estimation - Human Motion Analysis Project Report

David Köppl Sebastian Antes

June 10, 2025

1 Introduction

Combining music with visualization has become a staple of professional DJ shows and festivals. We think synchronizing light and video effects to music can enhance the listening experience. Yet entry-level and hobby DJs rarely have the resources to supplement all their music with fitting background visuals. They often perform at small venues or private living rooms with nothing more than a laptop and a basic DJ-controller. They can use existing audio-reactive visualizers, but these give performers no direct control over the visualization.

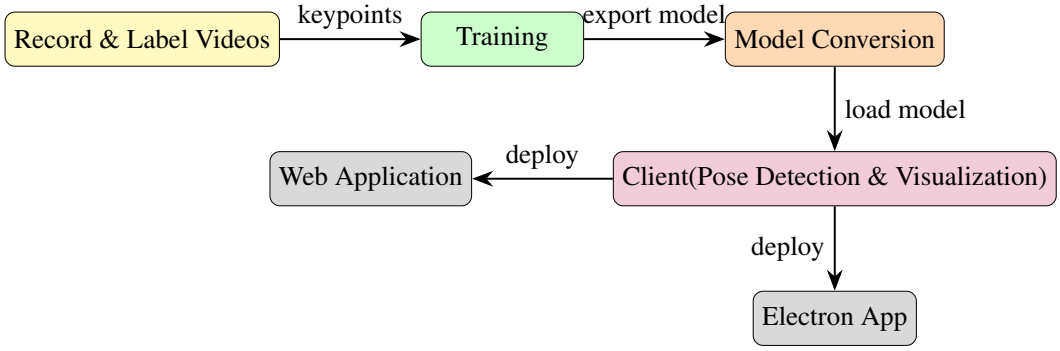
We address this gap with our pose-driven music visualizer, which uses arm gestures to drive a real-time particle system. It is designed for beginners who lack dedicated video artists, run on modest hardware and want minimal installation overhead. Motivated by these constraints, we designed a lightweight, cross-platform architecture.

Our application uses Google's MediaPipe pose estimator [1], which runs in the browser via TensorFlow.js [2]. We trained a five-pose classifier on custom videos and render a particle system for visualization using Three.js [3] with a WebGL backend. Everything is running client-side, so performers simply open a URL [4], allow webcam access and can start mixing music. This report summaries our design choices and explains the pipeline we used and how we evaluated our solution. We discuss our findings, limitations and performance trade-offs.

2 Method

2.1 System Overview

Our system consists of an offline training pipeline and a fully client-side runtime application.



Figur 1: High-level overview of our application setup.

For each pose we recorded n labeled videos at a resolution of 1920 x 1080 and a framerate of 25 FPS. Using MediaPipe's Python API, we extracted all key-points per frame and trained our classifier using TensorFlow. The model was exported as a "SavedModel".

A simple script then transform the "SavedModel" into TensorFlow.js format, so that it can be used in our web application. The converted model can then be used in our client application, which is a React and TypeScript project that can be build to run in any modern browser using Vite or as native program using an Electron wrapper. This application uses MediaPipe's JavaScript library and TensorFlow.js to classify user poses through a web cam video feed. The detected pose then influences a particle simulation, rendered using Three.js.

2.2 Pose Estimation

We rely on MediaPipe Pose because it offers identical APIs in Python and JavaScript. This lets us extract training features offline and run real-time inference in the browser with the

[1] `https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker` (accessed 2025-06-10)

[2] `https://www.tensorflow.org/js` (accessed 2025-06-10)

[3] `https://threejs.org` (accessed 2025-06-10)

[4] `https://masdaofdisasda.github.io/visual-analysis-project/` (accessed 2025-06-10)

same landmark definition. The light model provides 33 3D-landmarks and can detect one person per frame, which meets our needs. We only expect a single DJ in front of the camera and no detailed features.

The dataset was created by recording short video clips and labeling each video after a gesture in the file name. The MediaPipe pose detector goes over each video and each frame and writes all 33 features, each with an x,y,z coordinate and a visibility value plus the labeled class to a CSV file. Although, we only need upper-body joints for our classification, we kept all landmarks. The added features did not affect training time and reserved the option to extend the gesture later.

At runtime the same model, extracts all 33 features from web camera feed in the Browser and forwards it, without additional filtering, to our classifier.

2.3 Pose Classification

We implemented a lightweight TensorFlow sequential classifier using Keras. The network architecture is based on the Keras examples [5] and looks as follows:

- **Input:** vector of 33×4 $(x, y, z, visibility)$
- **Dense 1:** 128 units, ReLU activation
- **Dropout:** 30%
- **Dense 2:** 64 units, ReLU activation
- **Output:** C units (number of poses), Softmax activation

We trained on three poses initially and later extended to five poses. We used Adam optimizer and sparse categorical cross-entropy loss. Final validation accuracy exceeded 93 % after 15 epochs.

We deliberately kept the model simple to avoid incompatibilities during conversion to TensorFlow.js format.

2.4 Particle Simulation

The per-frame label detected by the TensorFlow.js model drives a GPU particle System. We initialize a total of 1024×1024 particles ($\approx$1 048 576 total) and simulate their positions and velocities via Euler integration under a curl-noise field. Each particle carries a temperature. They spawn with 10,000 K and cool down over time to 1,000 K, which we visualize by using an approximation of the blackbody radiation curve.

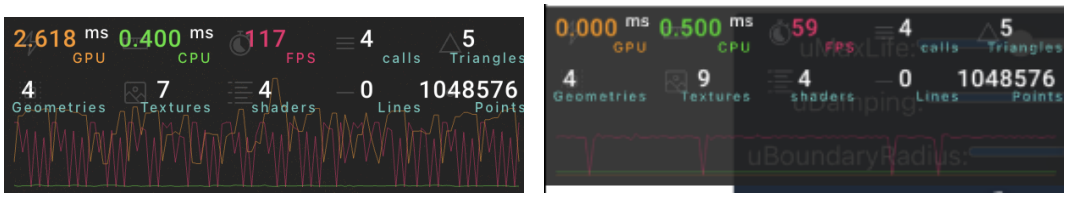
Technically, we implement the simulation with three double-buffered WebGL fragment shaders based on a blog post by Maxime Heckel [6]:

[5] `https://faroit.com/keras-docs/1.0.1/getting-started/sequential-model-guide/` (accessed 2025-06-10)

[6] `https://blog.maximeheckel.com/posts/the-magical-world-of-particles-with-react-three-fiber-and-shaders` (accessed 2025-06-10)

1. **Velocity Update** reads the previous velocity texture and position texture, applies a curl-noise force, then writes the new velocity.
2. **Position Update** reads the old position and the newly computed velocity, advances each particle by Δt, and writes the result.
3. **Render** draws each particle as a GL_POINT by sampling its position and lifetime from the latest textures.

A debug overlay, toggled by the D-key or double-tapping on mobile, displays the live webcam feed with landmark projections, the predicted class label, and a set of GUI sliders for tweaking the particle simulation. Our WebGL-only approach avoids compute shaders and runs between 60-120 FPS on M4 Pro MacBook (Fig. 2a) and at stable 60 FPS on iPhone 15 Pro (Fig. 2b).



(a) Profiling graph for MacBook with M4 Pro at a resolution of 3024x1964 pixels.

(b) Profiling graph for iPhone 15 Pro at a resolution of 2556x1179 pixels.

Figur 2: Profiling graph for test devices.

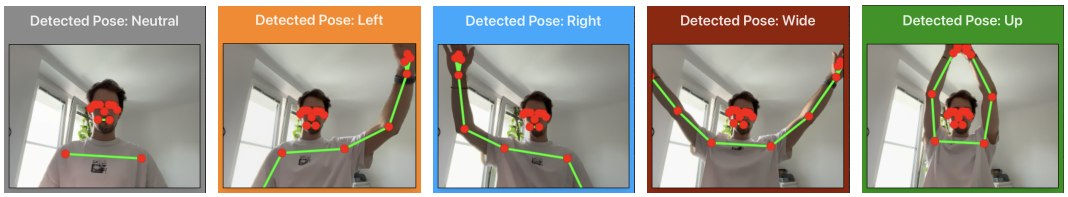
3 Experimental Setup

We recorded our training videos using webcams from Apple MacBooks, Dell XPS 15 and an external Emeet Nova webcam, to have the same fidelity and angle we expect from real-world scenarios. The model was trained on an desktop AMD Ryzen 5 5600X and an desktop AMD Ryzen 9 5900X.

For our classifier we choose five distinct poses:

- both hands down (neutral)
- right hand up (right)
- left hand up (left)
- both hands up sideways (wide)
- both hands up (up)

We expect that these poses produce keypoints that are easily differentiable and clearly classifiable. Fig. 3 shows all poses classified by our model and visualized with the application's debug overlay.



(a) Neutral (b) Left (c) Right (d) Wide (e) Up

Figur 3: All five detected pose classes

The five poses were mapped to the particle system by applying a force on them and adjusting the camera. The “neutral” pose does not add any influence. The “left” and “right” pose turn the camera and applies a rotation force according to their direction. The “wide” pose zooms the camera out from the origin and adds a force to push the particles outwards. The “up” pose just adds a force that moves particles upwards.

To make the visualizer audio-reactive, the curl velocity of the particles is mapped to the volume of the microphone input.

4 Results and Discussion

4.1 Strengths

The model detects the five target poses in real time while maintaining good performance on various hardware platforms and under varying lighting conditions. The system runs at a speed that allows smooth control of the particle system, even on mobile devices, without any noticeable delay. The results validate our decision to use a lightweight model together with browser-based deployment.

The detected poses create eye-catching visual effects in the particle system. The gesture mappings create an intuitive and immediate response when users perform them during DJ performances.

The system has an easy setup process as users can access the visualization through an URL and grant webcam permission without needing any installations, driver updates, or calibration procedures. Our solution provides easy accessibility for beginner DJs due to its user-friendly interface.

4.2 Weaknesses

The overall system performance remains good, except for some minor problems we noticed regarding the inconsistent results in pose classification. The system identifies the “left” pose with slightly lower accuracy than it does other poses. The system misinterprets raised arm movements because users do not fully extend their arms and because of small asymmetries or partial arm visibility.

The system experiences occasional misclassifications during pose transition periods. The transition from “neutral” to “up” often results in incorrect pose identifications of “left”, “right” or “wide” during intermediate frames. The system produces unwanted visual effects because of unintended in-between detections. The use of frame-wise classification without temporal smoothing or pose transition modeling leads to this detection limitation.

Another limitation comes from the environment in which DJs normally perform. Dim lighting conditions and colored strobe effects can reduce the tracking accuracy of MediaPipe´s pose estimator. Since the model relies on visible keypoints from the webcam, poor lighting conditions can cause failed detections, impacting the reliability of control. Additional illumination of the DJ could help the system guarantee constant tracking in dark nightclubs.

5 Conclusion

We presented a lightweight, pose-driven music visualizer that allows DJs to control visuals through simple arm gestures using only a laptop with a webcam and microphone. Our system runs entirely in the browser, requires no installation, and delivers real-time performance across devices. While the classifier works well under typical conditions, we observed limitations in transitions between poses and low-light environments. Future work could improve robustness with temporal smoothing and expand gesture vocabulary to offer more expressive control.