# From Gesture to Heartbeat: A Visual Reinterpretation of Performing Arts

Carlota Cortés Mir

**Abstract–** This project explores the intersection between technology and performance art through a system that translates human movement into expressive visual output. It integrates real-time gesture recognition with a symbolic 3D heart interface, transforming the system into a tool for storytelling, where motion conveys emotion. The technical development involved evaluating pre-trained models, training custom classifiers, building a real-time interaction pipeline, and defining gesture-driven functionalities within a custom 3D heart interface using OpenGL. As a result, the final system maps intuitive gestures to 3D visual metaphors and sound design, allowing performers to externalize emotion and narrative through movement. This is demonstrated in a live performance based on Charles Bukowski's Bluebird, which illustrates how literature and gesture can merge into embodied storytelling. Beyond its creative approach, this project contributes to the accessibility of the arts and proposes a new language of performance based on rhythm and expression. Looking forward, future work could include full-body gesture recognition, broader emotional mappings, and collaboration with artists to continue bridging creative expression and technological development.

**Keywords–** Emotion-driven interaction, gesture recognition, human-computer interaction, performance arts

## 1 Introduction

At present, the combination of art and technology has opened new possibilities for creativity, innovation, and human connection. As art and science are complementary elements: art inspires with beauty and curiosity, while technology provides the tools to explore and express. In this context, this work proposes the development of a software capable of analyzing gestural interpretation in performing arts, such as music, reading, and dance. Where, through advanced technologies of artificial intelligence and computer vision, It generates a visual reinterpretation based on body movement, transforming the gestures of the performers into dynamic visual representations that mirror the emotions of the original works.

This project combines disciplines such as scenography, visual arts, and digital arts to take a human-centered approach, aiming to create immersive and emotionally resonant experiences adapted to various sensitivities and ways of expression.

This work's fundamental question is: *"How can performance art step beyond traditional constraints and become a form of storytelling where gestures speak for us?"*

Poetry and music, for instance, do not have a physical form or tangible characters, making them inherently inaccessible to people with hearing impairments or communicative challenges. However, Human Pose Estimation offers an opportunity to bring these art forms to life through gestural representation.

Gestures play a fundamental role in communication. When We speak, we naturally use hand movements and body language to convey meaning. These innate and intuitive gestures are embedded within human expression, and this project seeks to harness them to visually translate performance art.

By expanding the language of performance to include gesture, visual metaphors, and responsive design, this project reimagines what it means to be present and heard in performance. Technology becomes not just a tool, but a translator of inner states, making emotion visible.

Disabled people have only in recent years gained access to the arts, and even today that access is limited in many cultural institutions [1]. Yet, the arts are vital, not only to a sense of cultural belonging, but also as a way to explore emotions, and express creativity. By expanding the concept of performance and integrating multi-sensory engagement, this project challenges conventional artistic norms, allowing audiences to experience performances in new and inclusive ways. Ensuring accessibility standards are met so that all spectators, regardless of their abilities, have a qualitative and immersive experience [2].

• Contact E-mail: 1639080@uab.cat
• Work tutored by: Fernando Vilariño (Dept. of Computer Science)
• Course 2024/25

## 2 State of Art

Human-Computer Interaction (HCI) is an area of research and practice that emerged in the 1980s, initially as a specialty area in computer science and cognitive science. Since then, it has grown into a multidisciplinary field that prioritizes not only functionality, but also the way humans experience and interact with technology [3].

While its early applications focused on efficiency, the field has increasingly embraced creativity, emotion, and expression. Therefore, aligning more closely with human-centered values. This shift is especially relevant in the context of performance arts, where technology is no longer a tool but a collaborator in shaping artistic expression and audience engagement [4].

This state-of-the-art review explores current case studies, challenges, and future directions of the interplay of HCI and performance arts, highlighting how their combination is reshaping traditional artistic practices.

### 2.1 Case Studies

The following case studies illustrate the integration of advanced HCI technologies, such as motion capture systems, gesture recognition frameworks, wearable devices, machine learning models, and interactive projection, into performance arts.

**Refik Anadol – Melting Memories:** Melting Memories features multidimensional visual structures, including data paintings, augmented data sculptures, and light projections that translate motor activity in the brain into immersive, artistic experiences. It offers new insights into the representational possibilities of human memory, as shown in Figure 1 [5].

Fig. 1: Melting Memories by Refik Anadol

**Mi.Mu Gloves - Gesture-Based Music:** The Mi.Mu Gloves offer a gesture-based interface for musical expression. They translate hand movements into real-time control of sound, redefining stage presence by incorporating movement and interaction. The gloves effectively create a new language of musical communication through gesture [6].

**Troika Ranch - Interactive Dance Theatre:** This company integrates digital technologies as integral components of live dance performances, where the dancer's movements directly control video, sound, and lighting in real-time. This provides new means for a dancer to express himself and creates a deeply immersive and reactive experience [7].

Fig. 2: Interactive Dance Theatre performance by Troika Ranch

### 2.2 Challenges

The interaction between humans and technology is complex and multi-disciplinary. It entails not only addressing technical and creative aspects, but also humanistic, ethical, and philosophical issues. The key challenges in HCI in arts include:

- **Technical Constraints:** Latency issues, sensor accuracy, and hardware limitations directly impact real-time performances [8].
- **Ethical and Privacy Concerns:** To support such interactions, it is often required that the environment records contextual and user data to function effectively, which introduces challenges in ensuring data security and privacy [8].
- **Accessibility:** Advanced HCI technologies may exclude artists or audiences lacking technical knowledge or resources. Artists with less technical expertise would need to collaborate with programmers or technicians to engage with the interactive platforms [8, 9].
- **Authenticity:** The integration of AI in creative processes raises questions about authorship and artistic integrity. While HCI advocates for meaningful human control, this ideal can be complicated in practice, especially when AI systems contribute significantly to the creative outcome [8, 9].
- **Cultural Shift:** Besides technical challenges and shifts, HCI in performance arts has to evolve mutually with society changes. Current public ideas often focus on fear and job displacement as technology surpasses human capabilities. Therefore, there is a need to shape a more optimistic vision of the field and its potential [9].

### 2.3 Future Directions

The trajectory of HCI is continuously expanding beyond the desktop. As technologies continue to mature, new opportunities arise, such as [8].

- **Responsive AI Performances:** Systems that dynamically change based on audience reactions and environmental conditions, creating responsive and personalized performances.

- **Cognitive Performance Interfaces:** Brain-computer interfaces that allow performers to control visual and auditory elements using cognitive states, opening new dimensions of expressive control.
- **Cross-reality Experiences (XR):** The integration of Augmented Reality (AR), Virtual Reality (VR), and Mixed Reality (MR) will enable fully immersive and multimodal performance environments.
- **Blockchain-based Art:** Technologies like blockchain can ensure authenticity and ownership of digital art and interactive artworks, supporting artistic integrity.

# 3 Objectives

For this project, my main objectives are:

- Deciphering the language of movement as a universal form of expression.
- Redefining performance art through inclusive storytelling.

# 4 Development

## 4.1 Pre-trained Models for Gesture Recognition

The first step in developing a gesture recognition system was prototyping with existing pre-trained models in order to evaluate their performance and suitability for my project needs. The aim was to test, understand, and compare how well each model could identify hand gestures, and assess whether they could be integrated directly or require further customization. I selected and experimented with three distinct pre-trained models:

### 4.1.1 Model 1: MediaPipe Gesture Recognizer (by Google)

MediaPipe's Gesture Recognizer lets you recognize common static hand gestures in real-time and provides the recognized hand gestures results and hand landmarks of the detected hands [10]. The system combines two models: Hand Landmark Model (detects hand presence and geometry) and Gesture Classification Model (recognizes gestures based on hand geometry).

The recognized gestures include: Closed fist, Open palm, Pointing up, Thumbs up/down, Victory, Love, Unknown (unrecognized gesture). Gesture examples can be seen in a demonstration video [11].

### 4.1.2 Model 2: HaGRID Dynamic Gestures (ai-forever GitHub

This model is trained on the HaGRID Dataset and focuses on dynamic gesture recognition. Unlike MediaPipe, this model can recognize gestures involving movement, making it more suitable for interactive systems [12].

It supports six groups of dynamic gestures, where each gesture can have more than one variation. The six dynamic gestures include: Zoom, Fast swipe (up/down), Click, Drag and drop, Swipe left, Swipe right, Swipe up, Swipe down.

### 4.1.3 Model 3: 3D CNN with Jester Dataset (Adithkumarba Github)

The third model I explored was a dynamic gesture recognition system built using a 3D Convolutional Neural Network architecture using TensorFlow and Keras. It was trained on a subset of just 8 out of the 25 available gesture classes in the 20bn-Jester dataset due to computational constraints [13].

The eight gestures include: Swiping left, Swiping right, Zooming in with full hand, Zooming out with full hand, Thumbs up, Thumbs down, Doing other things, No gesture.

These models were evaluated in terms of accuracy, responsiveness, and suitability for real-time applications. The evaluation results are presented in Section 5.1

## 4.2 Model Selection for Prototyping

After evaluating all three pre-trained models (Section 5.1), I decided to work with "Model 2: HaGRID Dynamic Gestures" for further development and prototyping. It stood out to me due to its high accuracy and real-time performance during testing. Additionally, the model provides all the essential resources, such as a well-annotated dataset, configuration files for training and testing, and a complete open-source implementation, making it easier to adapt and customize to my specific needs.

One of the key advantages of this model is its flexible approach to dynamic gesture recognition, which relies solely on static gesture predictions instead of full video-based training. This significantly reduces the complexity and computational cost [14].

As shown in Figure 3, the dynamic gesture recognition pipeline processes each video frame through a structured sequence of detection, tracking, classification, and gesture logic. The process does not rely on individual frames, but instead maintains a temporal buffer of recent hand states to identify gestures. Once an action is recognized, an event is triggered and there is a visual feedback.

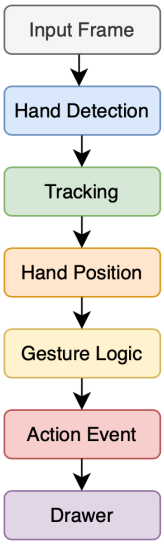


Fig. 3: Gesture Recognition System Flow

To complement this, Appendix A, Figure 6 shows a detailed example of how the system recognizes the ”zoom out” gesture across frames.

**Data Flow Summary:**

- Input Acquisition: The system continuously receives mirrored video frames from a live webcam feed.
- Hand Detection: An ONNX model detects bounding boxes and confidence scores.
- Tracking: Identified hands are tracked across frames using prediction and matching algorithms (OCSORT and Kalman).
- Gesture Logic: Movement patterns, pose transitions, and gesture duration are evaluated to determine if a dynamic gesture has occurred
- Action Dispatch: If a match is found, a corresponding event is triggered.
- Visual Feedback: The system displays recognized gestures for debugging and feedback.

This architecture allows for easy customization of dynamic gestures, as adding a new gesture simply involves defining sequences of static gesture and updating the logic files, without the need of retraining the models [14].

## 4.3 Baseline Replication and Evaluation

To gain a better understanding of the system's architecture, I began by exploring the open-source implementation available on their GitHub repository [14]. This repository includes the complete pipeline for training and testing, with multiple neural network architectures for gesture detection and classification, such as MobileNetV3small, ResNet18, ViT-B16, and SSDLiteMobileNetV3Large.

However, the final lightweight ONNX models built for real-time use are not publicly available, but after looking into their structure, it seems that the gesture detection is handled by a lightweight model similar to a Receptive Field Block (RFB) architecture, while gesture classification is done using a custom model with a single Residual Block.

To replicate this structure, I first analyzed the configuration system, data preprocessing, training pipeline, and evaluation metrics to better understand how everything works and later modify the training parameters in order to build a similar lightweight model.

Based on the experimental results published in Appendix A (Figure 7), I chose to train and evaluate SSDLiteMobileNetV3Large for the gesture detection task, and MobileNetV3small for classification due to their low inference times, which is critical for real-time gesture recognition systems [14]:

### 4.3.1 Dataset: HaGRIDv2-1M

All experiments were done using the HaGRIDv2-1M dataset (Hand Gesture Recognition Image Dataset), which allows image classification and image detection tasks. The dataset contains over 1 million Full HD RGB images across 33 gesture classes, including a “no gesture” class [14]. Appendix A, Figure 8 shows all gesture classes.

Some key dataset characteristics are:

- 65,977 unique individuals and scenes.
- Mainly indoor recordings with varied lightning conditions.
- Gesture distances range from 0.5 to 4 meters from the camera.
- Annotations are provided in COCO format (normalized bounding box coordinates).
- Each image may contain one or two bounding boxes, depending on the gesture type.

The dataset is split by user ID into training (76%), validation (9%), and testing (15%), with 821,458 images for training, 99,200 images for validation, and 165,500 for testing [14].

Due to the dataset's size, I used the lightweight version where all images were resized to 512 pixels on the shortest side.

### 4.3.2 Experimental Setup

I trained both models using the provided YAML configurations [14]. In both cases, the batch size was set to 128, and training was conducted for up to 100 epochs, with early stopping triggered if there was no improvement of at least 0.01 in the evaluation metric over 10 consecutive epochs.

The training parameters and evaluation metrics are shown in Appendix A, Table 4.

Their performance is evaluated in Section 5.2.

## 4.4 Custom Lightweight Gesture Classifier: Design and Experiments

As previously mentioned, the final lightweight ONNX models built for real-time performance are not publicly available [12]. There is no information of the evaluation metrics or training hyperparameters used for these models. Therefore, I decided to design and implement a lightweight classifier from scratch, inspired by the provided crops_classifier.onnx architecture.

I began by analyzing the configuration system, data preprocessing steps, training pipeline, and evaluation metrics available in the open-source implementation [14]. After familiarizing myself with the pipeline, resolving minor code issues, and executing several training and evaluation runs, I verified that the training setup functioned correctly and that the evaluation metrics were accurate.

This process helped me understand the overall framework and prepare to build the custom lightweight model from scratch. A challenging task, considering that there's no information about the model, other than it used a single residual block.

A residual block is a type of neural network block introduced as part of the ResNet architecture. They are skip-connection blocks that learn residual functions about the layer inputs, instead of learning unreferenced functions [15]. A residual block is a stack of layers arranged so that the output of a layer is added to the output of a deeper layer in the block. After this addition, a non-linearity is applied.

This shortcut connection or skip-connection, allows the network to pass information directly across layers [16].

With this understanding, from the .onnx file, I was able to extract the model architecture, which consisted of a single residual block with a ReLU activation after the addition operation, followed by global average pooling and a fully connected classification layer. This structure is similar to a simple ResNet architecture, so I built my model based on this layout.

#### 4.4.1 Experiment 1: One Residual Block

For my first attempt, I copied the hyperparameters from the published ResNet18.yaml configuration (batch size: 124, learning rate: 0.01, optimizer: Adam), even though my model was significantly smaller. As expected, I ran into immediate issues. The model wasn't learning, during 10 epochs the F1-score remained at 0.0 and the loss didn't converge.

#### 4.4.2 Experiment 2: Two Residual Blocks

As of these poor results, my solution was to add a new residual block for more representational capacity and lower the learning rate to 0.001 because it was too high for such a small architecture. Trying to improve performance while keeping the model lightweight for inference. I didn't modify the Adam optimizer due to its adaptive learning rate and faster convergence compared to SGD.

In training, the model plateaued at an F1-score of 0.29, with an early stopping at epoch 24, with the best score at epoch 14.

#### 4.4.3 Experiment 2.1: Two Residual Blocks finetuned

To improve performance without retraining from scratch (as each epoch took about an hour), I resumed training from epoch 14 (f1-score: 0.29), using a reduced learning rate of 0,0001 and a smaller batch size of 64. This adjustment was intended to allow finer optimization after initial convergence, especially since the previous configuration had shown a flattened loss curve.

This led to a slight improvement in performance, with the F1-score increasing to 0.35 in epoch 32. However, both the F1-score and loss plateaued again, a classic sign of hitting an architectural limit.

#### 4.4.4 Experiment 3: Three Residual Blocks + Dropout

Due to the previous bottleneck, I added a third residual block, incorporated dropout (0.3), and increased the learning rate to 0.0005 for better capacity and regularization.

Unfortunately, this configuration underperformed. The model reached a peak F1-score of 0.22 at epoch 22 and plateaued for some epochs. Clearly, this was worse than the previous version. I came to the conclusion that it would have been more effective to add only the third residual block without introducing dropout or altering the learning rate, as the dropout was likely too aggressive and the previous learning rate seemed to be better suited.

#### 4.4.5 Experiment 4: Four Residual Blocks + Batch Normalization

To overcome the previous model, I decided to revert the learning rate to 0.0001, remove dropout, and introduce a fourth residual block along with batch normalization layers within each block (BatchNorm2D). The architecture of the final model is shown in Figure 4.

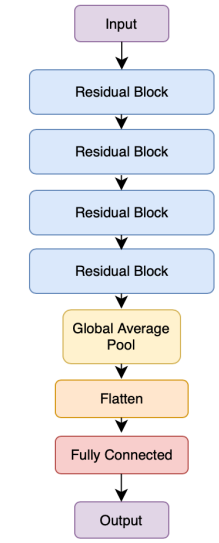


Fig. 4: Architecture of the final custom model

The performance of this final configuration is evaluated in Section 5.3

### 4.5 Customizing Gestures for Expressive Visual Interaction

As I've mentioned in Section 4.2 (Model Selection for Prototyping), the gesture recognition model offers a flexible approach to dynamic gesture recognition, allowing easy customization as gestures are defined as sequences of static gestures.

However, customizing gestures isn't just about adding new ones. It involves several steps, like updating logic files to define new events, writing the logic that detects and responds to those events, etc. My first step in customizing the system was cleaning up the code for better rescaling. I created a dedicated function for each dynamic gesture to keep things more organized and maintainable.

The original setup supported four main gesture types: swipes, zooms, clicks, and drag-and-drops. Each gesture type included multiple gesture variations. However, some of these variations felt unintuitive or ambiguous. Therefore, I decided to remove confusing gestures (like thumb-based swipes) for clarity, because my aim was to build a system that is universally understandable and expressive for performance. I ended up with the most intuitive ones: Zoom, Click and Double Click, Drag, Drop and Swipes.

## 4.6 Gesture-Driven Emotional Heart Interface

After developing and testing my gesture recognition system, I turned toward the question of visualization. What should these gestures control? How could I give them emotional and expressive meaning?

For this, I decided to work with a heart model system, a 3D object in OpenGL that I had previously developed. This system visualized a beating heart and was initially controlled with keyboard and mouse inputs. At that stage, it functioned mainly as a technical object: the heart could be rotated or viewed from different angles, but it had no emotional logic behind it.

Therefore, I began to explore how it could become a visual and emotional extension of the user, responding not to keys, but to gestures. Essentially aligning it with the broader purpose of the project: redefining performance art through inclusive storytelling, and deciphering the language of movement as a universal form of expression. I wanted to build a system where the heart didn't just beat but reacted and felt alive.

The goal was to deepen interaction by making it more expressive and immersive. By integrating real-time gesture detection, 3D visual metaphors, and sound design, I aimed to make the heart system become a living interface.

At the core of this idea lies the phrase: "Show me your heart". This concept serves as the narrative and emotional foundation of the project, shaping it as a poetic exploration of the heart as a mirror of the self. It proposes a new way to be seen, not through words, but through the pulse inside oneself.

This metaphorical approach enables users to externalize emotions that are often invisible or difficult to express, especially for those who struggle with verbal communication. Based on this idea, the heart becomes a new kind of voice where emotions are made visible and audible. Therefore, it invites and incorporates vulnerability by letting oneself be seen from the inside. This allows the heart to become a kind of emotional mirror that communicates through presence and rhythm.

For example, in the context of performative storytelling, the heart acts in synchrony with the narrative's emotional arc. It accelerates during moments of fear, slows in moments of peace, or stops in moments of shock or grief. These responses create a bridge between voice, body, and visual metaphor. Altogether deepening the emotional resonance for both the storyteller and the audience.

With the existing 3D HeartModel system, I made a series of technical modifications to transform it into a gesture-controlled and expressive interface. The objective was to merge gesture detection, sound design, and symbolic heart animations to create a responsive system that feels alive.

### 4.6.1 Original HeartModel System

The original system was a 3D OpenGL-rendered beating heart, controlled via keyboard and mouse input, where OpenGL (Open Graphics Library) is a cross-platform, open-source API (Application Programming Interface) for rendering 2D and 3D vector graphics. It's often used for visualizations, simulations, and interactive environments [17].

Its key components included:

- Camera: 3D camera control (WASD + QE), and reset with r
- Light: Sets the ambient, diffuse, and specular lighting setup
- GraphicsEngine: Handles rendering and toggles between single/multi-view perspective with p
- Model: Handles vertex updates and rotation via mouse (momentum + friction)
- Scene: Manages the overall scene

At this stage, the system had a beating animation, but its interaction was really limited and basic. It wasn't designed as a visual metaphor, but more as a technical render.

### 4.6.2 Simplifying the HeartModel for Gesture Integration

To prepare the model for gesture interaction, I first removed all mouse input to simplify the system to be only based on keyboard control. I replaced mouse-based rotation with fixed-speed keyboard controls (y, u, i, o) while still preserving friction for natural motion, avoiding stiff or rigid rotations. Besides, I assigned "r" to reset both the camera and the heart's rotation.

In the end, have all interactions controllable by keyboard, making it simpler to connect gestures (Table 1):

TABLE 1: FINALIZED KEYBOARD CONTROLS

| Action | Key(s) |
|---|---|
| Camera movement | w, s, a, d, q, e |
| Reset camera/rotation | r |
| Model rotation | y, u, i, o |
| Perspective switch | p |

### 4.6.3 Mapping Gestures to Actions

At this point, my gesture recognition system could classify: Click, Double Click, Zoom, Swipe, Drag and Drop.

So, I mapped these gestures to their corresponding keyboard functions in a way that felt intuitive (Table 2). For instance, mapping "Zoom in" to move the camera forward.

TABLE 2: GESTURE-TO-HEART ACTION MAPPING

| Heart Action | Keyboard | Gesture |
|---|---|---|
| Change perspective view | p | Double Click |
| Reset camera and heart position | r | Click |
| Move camera forward | w | Zoom in |
| Move camera backward | s | Zoom out |
| Rotate model to the left | y | Swipe left |
| Rotate model to the right | u | Swipe right |
| Rotate model upward | i | Swipe up |
| Rotate model downward | o | Swipe down |

Camera movements (a, d, q, e) weren't mapped to gestures, as I believed they had no strong emotional metaphor.

#### 4.6.4 Modularization and System Integration

To enable gesture integration, I removed key event listeners and converted all interactive logic into callable methods, to be activated when a gesture is detected. I then created a gesture-action mapping system to connect each gesture to its corresponding method.

Additionally, I refined some interactions to feel smoother. For example, the initial zoom felt static, so I modified it to use velocity and friction, allowing the camera to ease in and out of zoom.

To test before live gesture input, I created a CLI (Command Line Interface) simulator, where by typing gesture names into the terminal it triggered heart changes in real-time and validated them visually in OpenGL.

After string-based testing, I proceeded to integrate everything into a live gesture pipeline, bringing together the three key components: real-time gesture detection via webcam, gesture-to-action mapping, and graphical rendering.

I ran into some technical problems, as macOS restricts multiple GUI threads, meaning OpenGL and webcam feed couldn't run in parallel. Therefore, I rendered the webcam feed as a live texture inside the OpenGL window in the bottom corner. Allowing the user to see both the heart and their gesture input in the same window.

#### 4.6.5 Expanding Emotional Interactions

As I explored deeper into the emotional metaphor of the heart, I added new functionalities to reflect more internal states. These being the heartbeat animation control, ambient background music, and heartbeat sound synchronization.

First, I wanted the rhythm of the heart to be able to shift as a direct reflection of the user's inner state. I started by implementing it through keyboard control to test and debug. I began with simple logic, increasing the beats per minute by 10 or 20 when pressing a key. However, I realized it was too subtle and linear. I wanted to show emotions as an impact rather than a slow and steady rise. So, instead of incremental increases, I created three distinct heartbeat modes:

- Normal: A baseline rhythm, a neutral emotional state
- Slow: A soft, calm, peaceful state
- Fast: An intense, anxious, overwhelmed state

The exact BPM values weren't calculated through a precise methodology. They were chosen after testing different speeds and selecting those that looked more natural. For instance, when I imagined a heart beating incredibly fast when a person is stressed, I wasn't thinking about exact numbers but about creating an intensity that resonates. I also added the ability to stop the heartbeat entirely to represent a moment where the heart "stops", which I believe are moments of shock or grief.

Once I had these states working through keyboard control, I moved on to connecting them to gestures. As before, I wanted the mapping to be intuitive and symbolic. So, I assigned "drag" to slow the heartbeat, "drop" to speed it up, "stop" to freeze the animation, and the reset gesture to also have the reset of the rhythm.

Additionally, I added background music and a heartbeat sound to create a more immersive experience through sound design. Both tracks were created especially for this project by Maria Badji, who knew the kind of emotional tone I was aiming for and composed them with that in mind. The background music is somewhere between chill and eerie, with an ambient, subtle tone that doesn't distract but gently supports the heart.

The heartbeat sound was a one-second pulse, which I synced with the animation of the beating heart. However, when the heartbeat sped up, the audio no longer aligned properly as it was longer than the beating rhythm. It overlapped or cut off before it could finish. To fix this, I used AudioMass [18] to create a shortened version so it could match the faster pace.

I also tied these sounds to specific functionalities. When the stop gesture is detected, the heartbeat pauses, and the background music stops. This was designed to represent an extreme emotion of shock or emotional collapse, as I've mentioned earlier. In addition, the reset gesture restarts the heartbeat animation and resumes the music.

#### 4.6.6 Final Gesture-to-Emotion Mapping

Bellow is the final gesture-to-emotion mapping, detailing what each gesture activates and the emotional meaning it conveys.

- **Little Finger** — Resets the camera, heart position, heartbeat rhythm, and background music. *Emotion:* Calm or neutral state. Returning back to normal.
- **Tap or Double Tap** — Toggles between single and multi-perspective view. *Emotion:* Multiple emotions. Reflecting how we experience something from different perspectives or feelings.
- **Zoom In** — Moves the camera forward. *Emotion:* Vulnerability and intimacy. Offering a closer look into the heart, opening up.
- **Zoom Out** — Moves the camera backward. *Emotion:* Emotional distance or withdrawal. Distancing yourself from your own emotions.
- **Swipe (left/right/up/down)** — Rotates the heart accordingly. *Emotion:* Emotional conflict or indecision. Torn between choices or different emotions.
- **Drag** — Slows the heartbeat and sound. *Emotion:* Peacefulness and safety.
- **Drop** — Accelerates the heartbeat and sound. *Emotion:* Stress, rage, or emotional overload.
- **Stop** — Stops all heartbeat animation, music, and sound. *Emotion:* Shock or emotional shutdown. A panic freeze as if your heart has stopped.

## 4.7 Literary Integration

For my live demonstration, I wanted to choose a poem that could bring the system to life and to its full potential. A poem that could fit in the context of gesture recognition, emotional expression, and storytelling.

Choosing one poem was not easy, as not every poem contains the whole emotional range I've implemented in the system. But then I came across Bluebird by Charles Bukowski [see Appendix B for the full poem text].

At its core, Bluebird is a poem about vulnerability and suppression. It begins with the repeated refrain: "There's a blue bird in my heart that wants to get out", where the bluebird that is trapped in the speaker's heart represents his tender emotions. In this sense, the poem explores the inner conflict of someone who feels deeply but is unwilling or unable to express or show that part of themselves [19]. This directly aligns with the purpose of the project, to create a system where hidden emotional states are expressed not through words, but through movement and rhythm. The system provides an interface for vulnerability and intimacy. It enables a space where emotions become visible.

Therefore, the poem becomes a script for the system. To achieve this, I studied the poem carefully and its literary analysis [19], and did a gesture-to-poem mapping to guide the live performance.

### 4.7.1 Live Performance Mapping: Translating Poetic Emotion into Interaction

**Lines 1-5.** The poem begins with "there's a bluebird in my heart", which I interpret as a subtle moment of openness and intimacy. I perform the gesture Zoom in, bringing the camera closer to the heart to evoke vulnerability. As the speaker pulls back emotionally ("I'm not going to let anybody see you"), I mirror this with Zoom out, creating distance.

**Lines 6-13.** In this section, the speaker lists all the ways he suppresses emotion. I use the gesture Tap, switching perspectives to represent all his coping mechanisms.

**Lines 14-20.** In the second stanza, the speaker tells the bird to "stay down". For this, I use the gesture Swipe down, symbolizing almost like a push to supress emotion. Then, the three questions that follow raise the intensity as the speaker worries about being "messed up". As tension escalates, I perform the Drop gesture to accelerate the heartbeat, expressing stress and emotional overload.

**Lines 21-27.** In these lines, the tone softens. The speaker admits that he lets the bird out at night, in privacy. To reflect this, I use the Drag gesture to slow the heartbeat and introduce a sense of peace and safety.

**Lines 28-36.** In the final lines, the bluebird is gently put back and normality resumes. Therefore, I perform the Little Finger to reset the system and symbolize a return to neutral.

However, the poem concludes with the speaker saying that the pact "could make a man weep, but I don't. Do you?", shifting focus to the audience and challenging them to confront their own suppressed emotions. I use the gesture Stop to freeze the system and create a silent moment for reflection.

# 5 Results

## 5.1 Evaluation of Pre-trained Models for Gesture Recognition

**Model 1: MediaPipe -** The gestures are recognized with a very low latency, making for an excellent real-time performance. However, it is limited to a small set of static gestures and has no support for dynamic or sequential movements, which are for more interactive gesture-based interfaces.

**Model 2: HaGRID -** During testing, this model also showed real-time performance and high accuracy. Nevertheless, it is limited to specific dynamic gesture categories and may require further training for custom gestures.

**Model 3: 3D CNN (Jester) -** When experimenting with the model, the gesture classification results were inaccurate, even when adjusting the confidence thresholds. Besides, the inference speed was quite slow. However, the model showed incredible results in detecting hand, face, and pose landmarks in real-time.

## 5.2 Baseline Model Evaluation Results

**SSDLiteMobileNetV3Large – Gesture Detection:** Due to time constraints, the SSDLiteMobileNetV3Large was trained for 11 epochs, with the best results appearing at epoch 9, where both training loss and mAP improved. The model achieved a test mAP of 0.4851, which is reasonable given the limited training time. For reference, the original experiment reached a mAP of 72.7 (Figure 7), indicating potential for improvement with longer training.

**MobileNetV3_small – Gesture Classification:** This model converged quickly and early stopped at epoch 16, having achieved its best F1-score at epoch 6. The training loss curve shows rapid convergence, and the F1-score evaluation curve peaked early and then fluctuated slightly. Final test performance reached a F1-score of 89.52%, compared to 86.7% reported in the reference experiment (Figure 7).

These experiments confirm that both models are efficient, even with constrained training time. The training logs and evaluation metrics show good convergence, especially for MobileNetV3_small.

On one hand, SSDLite shows potential for accurate gesture detection but would benefit from additional training to improve its mAP. On the other hand, MobileNetV3_small converges rapidly and is well-suited for real-time classification tasks with limited resources.

## 5.3 Custom Classifier Evaluation and Insights

The final configuration (Four Residual Blocks + Batch Normalization) achieved the best results, with a validation F1-score of 0.84 and a test F1-score of 0.857.

However, the model's live performance did not meet the expectations. The original pre-trained model still performed better in real-time. While this outcome is disappointing, it is understandable given that the original

model was already built and fine-tuned to work well, and the GitHub repository lacked enough information to fully replicate it. Therefore, I decided to use the provided model for deployment, rather than continuing with my custom design.

### Insights

This experimental process provided valuable insights. It became clear that performance improves with the network depth (as shown in Figure 5 and Table 3). As I added more residual blocks, the F1-score generally increased, showing that the architecture benefits from depth.

An exception to the general trend was the model with three residual blocks, which performed worse than the two-block configuration. Nevertheless, this was likely due to the high dropout rate used during training. I believe that, without the dropout, this architecture would have outperformed the previous one.

This scaling behavior makes intuitive sense, as deeper models tend to capture more complex patterns, leading to improved accuracy. However, this is not practical in this case because increased depth also means higher computation complexity, which is not ideal for real-time applications. Simply increasing model depth is not the solution when considering inference speed and latency.

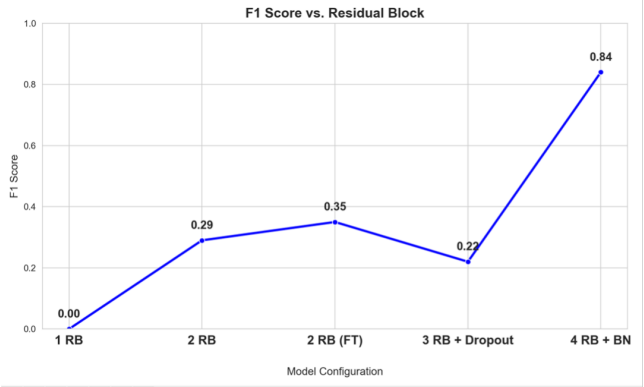


Fig. 5: Impact of residual block configurations and training hyperparameters on F1-score performance

TABLE 3: Model Performance with Different Configurations

| Configuration | F1 Score | Batch Size | Learning Rate | Extras |
|---|---|---|---|---|
| 1 RB | 0.00 | 124 | 0.01 | - |
| 2 RB | 0.29 | 124 | 0.001 | - |
| 2 RB (FT) | 0.35 | 64 | 0.0001 | - |
| 3 RB + Dropout | 0.22 | 64 | 0.0005 | Dropout=0.3 |
| 4 RB + BN | 0.84 | 64 | 0.0001 | BatchNorm |

It would have been ideal to keep the training hyperparameters constant while only varying the number of residual blocks, as this would have provided a more accurate evaluation of how depth alone affects performance. Unfortunately, due to time constraints and long training cycles, I prioritized achieving good results over a comprehensive analysis.

## 6 Conclusion

The project's goal was to explore the intersection of technology and performance art through gesture recognition. The final result, a system capable of emotionally reinterpreting performing arts through body movement, demonstrates the potential of combining these fields to create inclusive and immersive experiences.

Both of the project's main objectives were successfully achieved. First, it contributes to deciphering the language of movement as a universal form of expression by translating physical gestures into dynamic visual and emotional responses. Secondly, the system redefines performance art through inclusive storytelling by offering an interactive and expressive platform where emotions are embodied. The live performance using Charles Bukowski's Bluebird illustrates how literature and technology can converge into a new form of storytelling.

Throughout development, both pre-trained and custom-built gesture classifiers were explored, revealing the trade-offs between model accuracy, inference speed, and real-time usability. While my custom-built models showed improvements through deeper architectures, the final implementation relied on the pre-trained model for better real-time performance.

The integration of gesture recognition with the 3D heart system transformed a technical system into an expressive tool, making inner states visible through movement, rhythm, and interaction.

Future directions could include full-body gesture recognition, a broader emotional vocabulary, alternative visual metaphors, and integration with biometric or brain-computer interfaces. Collaboration with artists will also be essential to ensure creative depth.

In conclusion, this project shows that movement is more than motion, it is emotion and communication. By translating gesture into visual and auditory metaphors, the work opens new possibilities for artistic accessibility, self-expression, and human connection.

## 7 Multimedia Materials and Software

This section includes the main multimedia materials and software resources developed for this project, including video demonstrations and the source code repository.

### Video Demonstrations

- **Live Performance Demo** This video simulates a live demonstration of the system as a real-life performance scenario. It shows the performer reciting Bluebird by Charles Bukowski while executing gestures that control the 3D heart projection behind. However, the heart's visual quality is slightly reduced due to the projection.
- **Screen Recording Demo** This video complements the live demo by offering a high-quality screen recording that includes both the 3D heart interface and the gesture recognition feed. It demonstrates the same interaction as the live performance but without projection quality loss.

- **Functionalities Overview** This video focuses exclusively on demonstrating the core functionalities of the system (gesture-action), without the literary or performative context.

You can watch the Live Demo, the Screen Recording, or the Functionalities Overview

**Software** The source code for the complete system is available on Github: Github Repository

## ACKNOWLEDGMENTS

I would like to thank my tutor, Fernando Vilariño, for his guidance, support, and belief in my potential throughout this project. His encouragement pushed me to work harder and always aim to go further. Additionally, his vision of combining art and technology helped shape the core and depth of this work.

Special thanks go to Maria Badji and Adria Molina, who offered help and creative input along the way. Maria not only composed the music for the system, but also contributed her perspective and emotional insight, helping me develop the emotional mapping of gestures.

This project has been supported in part by the Grant Chair on Research and Artificial Intelligence in the field of Music/Arts (TSI-100929-2023-2).

## REFERENCES

[1] ACT, *Draft Guide 'Accessibility to the Scenic Arts'*, Universitat Autònoma de Barcelona, 2018. Available: https://webs.uab.cat/act/wp-content/uploads/sites/126/2017/03/act_io8_draft_guide_accessibility_to_the_scenic_arts.pdf

[2] National Endowment for the Arts, *Celebrating 40 years of promoting accessibility in the arts*, 2019. Available: https://www.arts.gov/stories/blog/2019/celebrating_40years_promoting_accessibility_arts

[3] J.M. Carroll, "Human Computer Interaction - Brief Intro", Interaction Design Foundation – IxDF, 2014. Available: https://www.interaction-design.org/literature/book/the_encyclopedia_of_human_computer_interaction_2nd_ed/human_computer_interaction_brief_intro

[4] M. Arveng, "The Next Paradigm Shift in Human-Machine Interaction" [Video], TEDxTrondheim, 2023. Available: https://www.youtube.com/watch?v=lC5AN4fJAcs

[5] Refik Anadol Studio, "Works" [Online]. Available: https://refikanadol.com/works/

[6] Sound On, "The Future of Music with Mi.Mu Gloves" [Video], Youtube, 2022. Available: https://www.youtube.com/watch?v=ci-yB6EgVW4

[7] K. Farley, "Digital Dance Theatre: The Marriage of Computers, Choreography and Techno/Human Reactivity", *Body, Space & Technology*, vol.3, no. 1, 2002. Available: http://doi.org/10.16995/bst.232

[8] C. Stephanidis, G. Salvendy, M. Antona, J. Y. C. Chen, J. Dong, V.G. Duffy, J. Zhou, "Seven HCI Grand Challenges", *International Journal of Human–Computer Interaction*, vol. 35, no. 14, 2019. Available: https://doi.org/10.1080/10447318.2019.1619259

[9] R. Jacobs, S. Benford, and E. Luger, "Behind the Scenes at HCI's Turn to the Arts", Mixed Reality Laboratory & Horizon, University of Nottingham, Microsoft Research, 2015. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2015/04/CHI15_Jacobs.pdf

[10] Google, "MediaPipe Gesture Recognizer – Python" [Online]. Available: https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer/python?hl=es-419

[11] TheCodingBug, "Gesture Recognition Using OpenCV and MediaPipe" [Video], YouTube, 2023. Available: https://www.youtube.com/watch?v=cJgDuywJv8Y

[12] Ai-forever, "Dynamic Gestures" [GitHub repository]. Available: https://github.com/ai-forever/dynamic_gestures

[13] A. Kumar, "Hand Gesture Recognition with Deep Learning" [GitHub repository]. Available: https://github.com/Adithkumarba/Hand_Gesture_Recognition_with_Deep_Learning

[14] A. Nuzhdin, A. Nagaev, A. Sautin, and A. Kapitanov, "HaGRIDv2: 1M Images for Static and Dynamic Hand Gesture Recognition" [GitHub repository]. Available: https://github.com/hukenovs/hagrid

[15] Papers with Code, "Residual Block" [Online]. Available: https://paperswithcode.com/method/residual-block

[16] P. Oommen, "ResNets - Residual Blocks & Deep Residual Learning", *Towards Data Science* [Online]. Available: https://towardsdatascience.com/resnets_residual_blocks_deep_residual_learning_a231a0ee73d2/

[17] Lenovo, "What is OpenGL?", Lenovo Glossary [Online]. Available: https://www.lenovo.com/us/en/glossary/what-is-opengl/

[18] AudioMass, "Online Audio Editor" [Online]. Available: https://audiomass.co

[19] E. Baldwin, "Poem Analysis: Bluebird by Charles Bukowski" [Online]. Available: https://poemanalysis.com/charles-bukowski/bluebird/

# APPENDIX

## APPENDIX A:

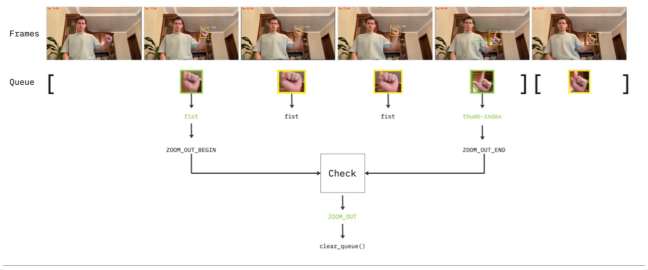


Fig. 6: Dynamic gesture recognition pipeline, exemplified by the "zoom out"

| Model | Model size (MB) | Parameters (M) | Inference time (ms) | Metrics | |
|---|---|---|---|---|---|
| | | | | F1-score | mAP |
| **Gesture Detection** | | | | | |
| SSDLite MobileNetV3 Large [35] | 10.3 | 2.7 | 26.3 | - | 72.7 |
| YOLOv10n [62] | 10.33 | 2.7 | 85.67 | - | 88.2 |
| YOLOv10x [62] | 121.5 | 31.6 | 1145.6 | - | **89.4** |
| **Full-Frame Classification** | | | | | |
| ResNet18 [18] | 42.7 | 11.2 | 37.8 | 98.3 | - |
| ResNet152 [18] | 222.65 | 58.2 | 226.94 | **98.6** | - |
| MobileNetV3 Small [20] | 6 | 1.6 | 5.1 | 86.7 | - |
| MobileNetV3 Large [20] | 16.3 | 4.2 | 11.34 | 93.4 | - |
| ViTB16 (pretrained) [13] | 327.4 | 85.8 | 350.9 | 91.7 | - |
| ConvNeXt Base [36] | 334.2 | 87.6 | 320 | 96.4 | - |
| **Hand Detection** | | | | | |
| YOLOv10n [62] | 10.3 | 2.7 | 75.8 | - | 87.9 |
| YOLOv10x [62] | 120.8 | 31.6 | 1150.7 | - | **88.8** |

Fig. 7: Experimental results of the models by the authors


Fig. 8: All 33 gesture classes of the HaGRID dataset

TABLE 4: TRAINING PARAMETERS

| Model Training Details | | |
|---|---|---|
| SSDLite | Optimizer | SGD |
| | Weight Decay | $5^{-4}$ |
| | Learning Rate | $1^{-2}$ |
| | Scheduler | LambdaLR |
| | Scheduler Params | Sinusoidal function |
| | Evaluation Metric | mAP |
| MobileNetV3 | Optimizer | SGD |
| | Weight Decay | $5^{-4}$ |
| | Learning Rate | $5^{-3}$ |
| | Scheduler | StepLR |
| | Scheduler Params | Step size: 30, gamma: 0.1, $T_{max}$: 8 |
| | Evaluation Metric | F1-score |

## APPENDIX B: POEM – BLUEBIRD BY CHARLES BUKOWSKI

There's a bluebird in my heart
that wants to get out
but I'm too tough for him,
I say, stay in there,
I'm not going to let anybody see you.
There's a bluebird in my heart
that wants to get out
but I pour whiskey on him
and inhale cigarette smoke
and the whores
and the bartenders
and the grocery clerks
never know that he's in there.

There's a bluebird in my heart
that wants to get out
but I'm too tough for him,
I say, stay down,
do you want to mess me up?
You want to screw up the works?
You want to blow my book sales in Europe?
There's a bluebird in my heart
that wants to get out
but I'm too clever,
I only let him out at night sometimes
when everybody's asleep.
I say, I know that you're there,
so don't be sad.
Then I put him back,
but he's singing a little in there,
I haven't quite let him die
and we sleep together like that
with our secret pact
and it's nice enough
to make a man weep,
but I don't weep...
Do you?