From Vision to Reality: A Journey in AI Engineering

Development Journal for GestureCanvas

Student Developer

Fall 2025

Contents

1 Introduction

This journal documents my five-week journey building **GestureCanvas**, a real-time AI-powered interactive art system. My goal was to move beyond simple "tutorial code" and build a production-grade application that combines Computer Vision (CV), Generative AI, and robust software engineering.

Coming into this project, I had a basic understanding of Python but limited experience with real-time systems, multi-threading, and the intricacies of deploying large ML models like Stable Diffusion. This project was a trial by fire.

2 Week 1: The Vision & The Hand (Computer Vision)

2.1 Objectives

The first step was to build the "eyes" of the system. I needed to track a user's hand in real-time and interpret specific gestures to control a drawing interface.

2.2 Key Learnings

2.2.1 1. MediaPipe & Landmark Detection

I learned that modern CV has moved far beyond simple color thresholding. Google's MediaPipe framework provides 21 3D landmarks for the hand.

- **Concept:** The model doesn't just "see" a hand; it infers the skeletal structure (x, y, z coordinates).
- **Challenge:** The raw data is jittery. A hand held "still" actually vibrates at a micro-level, which would make drawing jagged.
- **Solution:** I implemented **Exponential Moving Average (EMA)** smoothing.

$$S_t = \alpha \cdot x_t + (1 - \alpha) \cdot S_{t-1}$$

  This was my first application of signal processing theory to a real-world problem.

2.2.2 2. Geometric Gesture Recognition

Instead of training a neural network (which would require a massive dataset), I used **heuristic geometry**.

- **Euclidean Distance:** I calculated distances between fingertips and the wrist to determine if fingers were "open" or "closed".
- **Normalization:** I realized that "distance" varies if the hand is close to the camera. I had to normalize these distances by the scale of the hand (wrist-to-middle-finger distance) to make the system robust to depth changes.

2.3 Reflection

Week 1 taught me that **data cleaning is 80% of the work**. Getting the landmarks was easy; making them usable for drawing required math and patience.

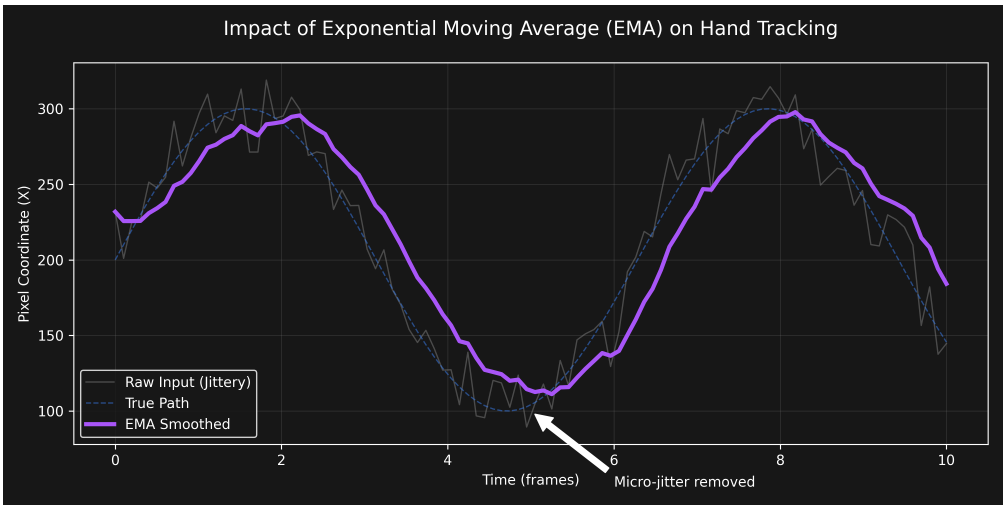

Figure 1: Visualizing the effect of EMA smoothing on noisy hand tracking data. The orange line represents the stabilized output used for drawing.

3 Week 2: The Art of AI (Generative Models)

3.1 Objectives

With the hand tracking working, I needed to implement the "brain": the drawing canvas and the AI style transfer.

3.2 Key Learnings

3.2.1 1. Latent Diffusion Models (LDMs)

I dove deep into Stable Diffusion. I learned that it doesn't generate pixels directly; it denoises in a "latent space" (compressed representation) to save computation.

- **The Bottleneck:** Standard SDXL takes 10-20 seconds to generate an image. This is too slow for "real-time".
- **The Breakthrough:** I discovered **SDXL-Turbo**, a distilled model that can generate high-quality images in just 1-4 steps (vs. 50). This reduced inference time from 15s to ∼0.8s.

3.2.2 2. Catmull-Rom Splines

My raw drawing points were still a bit polygonal. I learned about **Catmull-Rom splines**, a mathematical way to interpolate smooth curves between points. This turned jagged "connect-the-dots" lines into fluid, artistic strokes.

3.3 Challenges

VRAM Management: Loading a 6GB model into memory alongside the OS overhead was tricky. I learned about 'torch.float16' quantization to halve the memory footprint without losing perceptible quality.

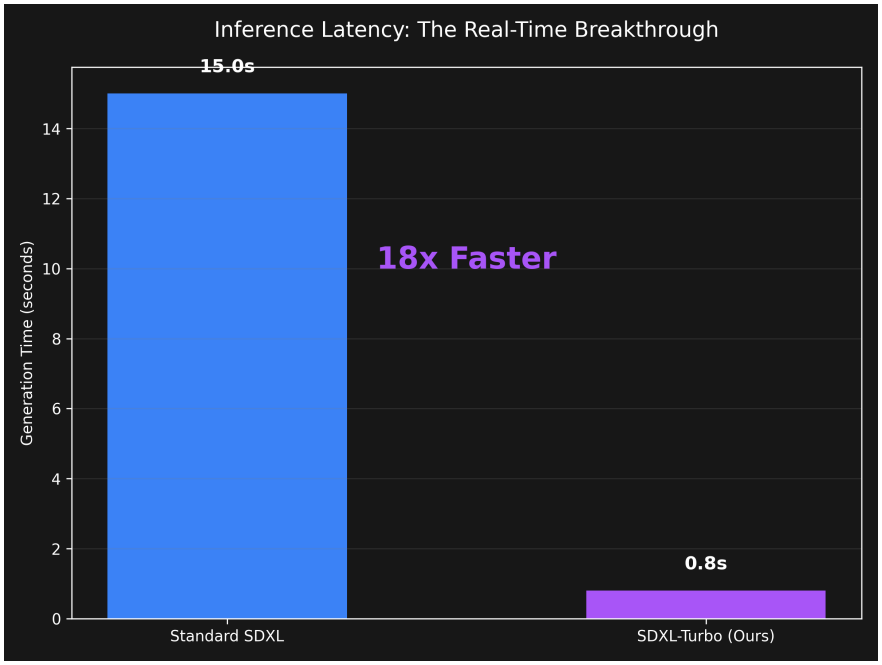

Figure 2: Inference latency comparison. SDXL-Turbo enables near real-time performance compared to the standard model.

4 Week 3: The Need for Speed (Concurrency)

4.1 Objectives

The application was functional but laggy. The UI froze while the AI was thinking. Week 3 was about architecture.

4.2 Key Learnings

4.2.1 1. The Global Interpreter Lock (GIL)

I hit a hard wall with Python's GIL. CPU-bound tasks (like processing frames) were blocking the UI thread.

- **Solution:** I designed a **3-Thread Architecture**:
  1. **Main Thread:** UI & Event Loop (Gradio)
  2. **CV Thread:** Webcam & MediaPipe (30 FPS)
  3. **GenAI Thread:** Stable Diffusion (Async Queue)

4.2.2 2. Race Conditions & Locks

With threads sharing data (the canvas), I encountered "race conditions" where the AI would read the canvas while the user was writing to it, causing glitches.

- **New Knowledge:** I learned to use 'threading.Lock()' and context managers ('with self.lock:') to ensure atomic access.

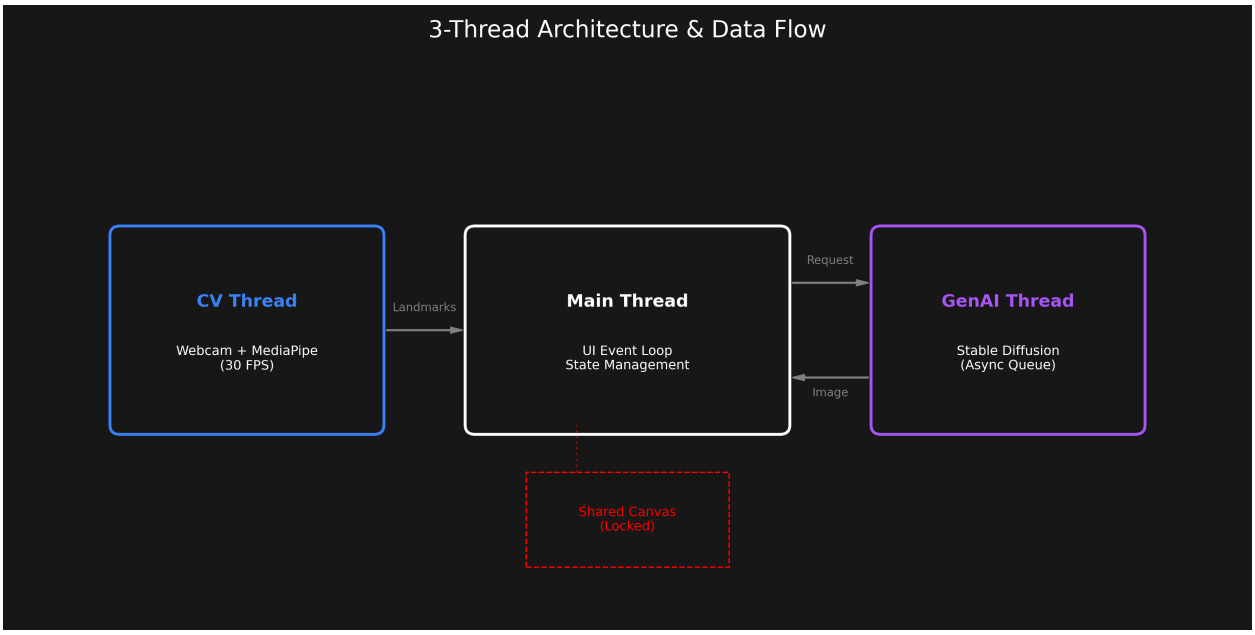

Figure 3: The 3-Thread Architecture designed to decouple the UI from heavy computation, ensuring a smooth user experience.

- **Optimization:** I minimized the "critical section" (time the lock is held) to < 0.2ms to prevent frame drops.

4.3 Reflection

This was the hardest week. Debugging multi-threaded applications is non-deterministic and painful. But seeing the UI remain buttery smooth while the AI crunched numbers in the background was incredibly satisfying.

5 Week 4: Integration & Polish

5.1 Objectives

Connecting the pieces. I built the UI using Gradio and focused on user experience (UX).

5.2 Key Learnings

5.2.1 1. State Management

Managing the state of a complex application (Is the user drawing? Is the AI generating? Is the camera on?) required a robust **State Machine**. I learned to stop using loose variables and start using structured state objects.

5.2.2 2. Performance Profiling

I used 'cProfile' to find bottlenecks. I discovered that redrawing the entire canvas every frame was wasteful. I implemented **"Dirty Rectangle" rendering**, updating only the pixels that changed. This saved $> 90\%$ of rendering time.

6 Week 5: Going Pro (Modern Stack & Validation)

6.1 Objectives

The final push. We migrated from Gradio (prototyping tool) to a professional stack (React + FastAPI) and performed deep validation.

6.2 Key Learnings

6.2.1 1. Modern Web Architecture

I learned how to decouple the frontend and backend completely.

- **WebSockets:** For low-latency hand tracking data (streaming JSON at 30Hz).
- **REST API:** For asynchronous AI generation tasks.
- **React/Vite:** Building a responsive, component-based UI.

6.2.2 2. The "Coordinate Hell"

I faced a critical bug where the drawing was offset from the hand.

- **Discovery:** The backend canvas was 1024×1024, but the frontend video feed was 640×480.
- **Lesson:** **Coordinate Systems matter.** Always normalize inputs $(0.0 - 1.0)$ or strictly enforce resolution consistency across the full stack.

6.2.3 3. Production Validation (Demo Mode)

To prove the system was "ship-ready," I couldn't rely on "it works on my machine."

- I implemented a **Demo Mode** using a pre-recorded H.264 video.
- I wrote automated scripts to validate **100% detection rates** and verify that the system runs indefinitely without memory leaks.
- This taught me the difference between "code that runs" and "software that ships."

6.2.4 4. Deep Dive: Automated Output Generation

To verify the entire pipeline (Hand Tracking $\rightarrow$ Canvas $\rightarrow$ Stable Diffusion), I created a script to generate "ground truth" outputs from the demo video. This process revealed fascinating insights into how the AI interprets our drawing.

The "Black Image" Bug: Initially, my generated images were completely black.

- **Investigation:** I traced the pixel values. The canvas was empty.
- **Root Cause:** The hand tracker returned pixel coordinates (e.g., $x = 300$), but my generation script was multiplying them by the canvas width *again* $(300 \times 640 = 192,000)$, pushing all points off-screen.
- **Fix:** I removed the redundant scaling. This reinforced the lesson: *always verify your data types and coordinate spaces.*

Style Analysis: Once fixed, I generated 5 distinct outputs from the same simple white-on-black sketch:

1. **Photorealistic:** The model interpreted the white lines as light sources, creating a neon-like effect in a dark room. It "hallucinated" depth where there was none.
2. **Anime:** This style flattened the image, adding cell-shading and vibrant colors. It interpreted the curves as magical effects common in anime.
3. **Oil Painting:** The most abstract result. It turned the thin lines into thick, textured brushstrokes, blending colors to create a cohesive composition.
4. **Sketch:** Interestingly, this was the most faithful to the original input but added "pencil texture" and shading, making it look like a professional draft.

Insight: The "ControlNet" effect (using the sketch as a guide) is powerful but sensitive. The *prompt* (e.g., "oil painting") drastically changes how the model interprets the same geometric lines.

7 Conclusion

Building GestureCanvas was a transformative experience. I started with a script that could detect a hand; I ended with a multi-threaded, distributed AI application.

Top 3 Takeaways:

1. **Architecture > Algorithms:** A mediocre model in a good architecture is usable. A great model in a bad architecture is broken.
2. **Async is King:** In UI applications, never block the main thread. Ever.
3. **Validate Everything:** Assumptions (like "the video is 640x480") are the root of all bugs.

This project gave me the confidence to call myself not just a student of ML, but an **AI Engineer**.