Neptune Interview Response

Vishaal Saravanan

1 Approach to Implementing a Real-Time Body Tracking System for a Mobile Unity Application (In Plain English)

If I had to build a real-time body tracking system within a Unity app running on a phone (or for designing for Neptune in the near future :)), here's how I would approach it—combining insights from my task implementation. (Side note: I began by working on the project, then returned to answer questions here, which felt like the right approach.)

- **Figure out what really needs tracking:** First off, I'd decide what the system needs to *actually* do. Are we checking if the elbow is too high? Is the hand shaky? Is the posture stable? That tells me how much effort I should put into precision vs. speed.
- **Use something small but smart:** I'd use something like MediaPipe BlazePose[1] or MoveNet Lightning [2]. These models are pretty fast and don't eat up phone memory. And from my practical assignment, I know that just tracking 17 or 33 key points is often enough.
- **Simple math goes a long way:** Honestly, most of this stuff doesn't need deep learning. If the elbow is higher than the shoulder, we don't need a transformer to tell us that. I prefer starting with basic angles (like elbow-shoulder-wrist) and height checks—it's faster, easier to debug, and less scary for mobile.
- **Add smoothing early:** One thing I realized during testing and working with real-time pose estimation at my previous work is that jitter kills user trust. So I'd use a tiny buffer (5-frame deque) to stabilize the feedback. It smooths out those "oops, I moved for 1 frame" moments.
- **Get it talking to Unity:** My plan would be to either export angles/keypoints using ONNX/TFLite or just run pose estimation in a backend and stream the results into Unity (maybe using Barracuda or a lightweight bridge).
- **Make it mobile-friendly:**
  - Keep the model file small (<10MB if possible).
  - Quantize the model if needed(int8, ?).
  - Use frame skipping if the app starts lagging.
- **Build a visual overlay for debugging:** I'd add some live markers and debug texts showing current angles, posture status, and FPS. This helps tweak logic in real time, and honestly makes testing way more bearable.

1.1 AI Models and Techniques for Posture Detection and Real-Time Feedback

I would like to answer this question, with this thought in mind, " Accuracy is important, but so is interpretability and speed—" especially if you're working with a violin in one hand and learning to play it.

- **Pose Detection Backbone:** To begin, I would likely use lightweight, quantized models optimized for mobile environments, and benchmark them against other architectures if my approach shows a significant difference in accuracy. I chose MediaPipe BlazePose for my task implementation because it provided the best real-time performance while capturing enough upper-body keypoints (shoulder, elbow, wrist) to implement meaningful logic. If deploying to mobile or edge devices, I would either continue using BlazePose or consider MoveNet Lightning as an alternative.

- **Why Not ViTPose (Yet)?** While ViTPose and other transformer-based models bring impressive precision and context awareness, they're generally overkill for this use case. As discussed earlier (and more in the next section, question F) For one, they're computationally heavy—even their "tiny" versions aren't truly optimized for mobile inference. More importantly, they solve problems I didn't have yet. If I ever needed to do ultra-fine motion capture (like tracking subtle finger pressure or instrument grip tension), that's when I'd reconsider using ViTPose or a distilled version of it.
- **Geometry Over Guesswork:** Most posture issues I care about (elbow too high, shoulder raised, wrist drifting) can be captured by simple joint angle checks and comparisons. In fact, I enjoyed how I could explain the entire feedback system with just trigonometry—no black boxes needed.
- **When to Mix in a Classifier:** That said, if I needed to detect more nuanced issues like the ones which I previously discussed or some time-component realted issues such posture movements, such as shifts over time that seem fine but aren't, I'd add a lightweight classifier (converted to ONNX). This model could be trained on violin-specific labeled sequences, enabling it to recognize more subtle patterns than rule-based logic.
- **Hybrid Strategy I'd Try Again:** In fact, from my earlier interview project, I explored a hybrid logic: let one model (say, BlazePose-based rule-checks) act as the first pass. If a pose's confidence is low or crosses a certain threshold, hand over the decision to a second opinion (MoveNet)— either from a classifier or by averaging predictions from both systems. This way, I get speed where I can and precision where I need it.
- **Techniques in Practice** Apart from joint angle calculation, rule-based thresholding, temporal smoothing, and the adaptive feedback mechanism (which are already implemented in my task :)), I'm also curious about some ideas currently lingering in my head, such as:
  - **Confidence-Aware Fusion:** Switch or blend logic sources (e.g., MoveNet + BlazePose) when confidence dips or ambiguity is high.
  - **Fallback Handling:** Freeze or hide feedback if model confidence falls below a threshold for a few consecutive frames—rather than misleading the user, better to be honest.

1.2 Body Tracking on Mobile Presents Several Challenges

Oh, definitely a few — body tracking on mobile is a fun puzzle, but not always easy to work with. I would like to be less elaborate on issues I have already tackled in my task implementation and more on the ones that I haven't, but certainly thought about.

1. **Real-Time Performance Bottlenecks (Tried to solve in Implementation)**
   - *Problem:* Not every phone is a flagship. Real-time inference tends to push low-end or even mid-range devices too hard.
   - *What I'd Do:* Stick to efficient models like BlazePose or MoveNet Lightning. Quantize to `INT8`, and reduce resolution or skip frames dynamically based on current FPS.
2. **Landmark Jitter and Frame Flicker (Tried to solve in Implementation)**
   - *Problem:* Even good models sometimes freak out over shadows or sudden movement—especially in low light.
   - *What I'd Do:* Use a rolling buffer (I went with a 5-frame `deque`) to smooth out transient spikes. Only act on posture flags that persist for a few frames—one frame of chaos shouldn't ruin the user experience.
3. **Posture Ambiguity in 2D (Tried to solve in Implementation)**
   - *Problem:* In 2D space, the same elbow height might mean "raised" or "neutral" depending on the body's angle to the camera.
   - *What I'd Do:* Combine Y-coordinate checks with joint angles like shoulder-elbow-wrist. I'd also let users define their own reference posture to calibrate the system—it's more personal and more robust that way.
4. **Integrating with Unity (a.k.a. the Real Boss Fight)**

- *Problem:* Getting live pose data into Unity in a way that doesn't lag, crash, or break everything else.
- *What I'd Do:* Export models to `ONNX` and use Unity Barracuda if embedding locally. If not, I'd send keypoints from a Python backend using WebSocket or UDP—easier to debug, and lets Unity do what it's good at.

5. **Lighting, Occlusion, and Camera Noise**
   - *Problem:* Pose detection gets confused in bad lighting or when the user turns too much.
   - *What I'd Do:* Keep the camera input at 320×240 to balance detail and speed, and use landmark confidence to decide when to pause or dim the feedback. If the model's unsure, the UI should say so.
6. **Device Heat and Battery Drain**
   - *Problem:* Continuous pose estimation drains battery fast and heats up the phone like it's brewing coffee.
   - *What I'd Do:* Cap the FPS, use hardware acceleration (e.g., TensorFlow Lite with GPU/NNAPI), and let the user toggle "power saving mode" if needed.
7. **What's "Correct" Anyway?(Partially, Tried to solve in Implementation)**
   - *Problem:* Everyone plays violin a little differently. Telling a seasoned player their posture is "wrong" can backfire.
   - *What I'd Do:* Let users record their own "ideal" reference poses or pick from predefined styles. Feedback becomes less preachy and more personalized.

2 Additional Questions on Real-Time Motion Tracking

A What differences have you observed between 2D and 3D pose estimation models?

2D models are like drawing stick figures on paper—they give you a clean view of where the joints are, but no idea how far they pop out or sink in. For most upper-body posture tasks (like violin playing), 2D works surprisingly well if the camera angle is stable.

3D models, on the other hand, introduce depth—they're helpful when body parts overlap or rotate (like bowing toward/away from the camera). But in my experience, they tend to be heavier, less reliable without multi-view setups or depth sensors, and harder to deploy on mobile unless you're willing to cut a few corners. I'd reach for 3D if I were tracking fine finger movements or bow tilt across planes (Maybe a violnist sitting sideways in orchastera, than facing the audience)—not for basic arm elevation.

B Can you explain how synthetic data helps improve model accuracy in motion tracking?

Synthetic data is like Taylor Swift rehearsing her choreography and stage flow in a local theatre before taking it global on the ERA's tour—except, in this case, the crowd is made of render engines. We can simulate endless variations: bowing poses under different lighting, with different camera angles, outfits, and body types. All without booking a real studio or violinist.

For motion tracking, especially something as niche as violin posture, getting real, annotated footage is expensive and time-consuming. Synthetic avatars help fill those gaps. We can ideally teach the model what "good" and "bad" posture looks like—even in rare cases like extreme bow angles or wrist misalignment—without waiting for perfect data to show up.

More importantly, synthetic data trains the model to be robust to noise: odd backgrounds, unusual limb lengths, or weird shadows. And the best part? We control everything. We can label postures as "ideal" or "needs correction" right at generation—no expert annotator needed(it would be great, if we had though, to have a mixed annotation strategy). It's not just data—it's tailored rehearsal for the model.

C If our AI model struggles with detecting a violinist's bowing motion, how would you debug and improve its accuracy?

I'd take a layered approach as I discussed my orginal answers in section 1.

- **Visual Checks:** First, render the key points on the video—sometimes the model's technically "working," but it's picking up the wrong elbow.
- **Data Audit:** Check if the dataset has enough examples across bowing speeds, body sizes, angles, and lighting conditions.
- **Augmentations:** Add more training diversity—like low-light simulation and motion blur.
- **Model Tuning:** If CNNs are missing context, I'd experiment with dilated convolutions or a tiny transformer head for better joint relationships (I know this is the bottleneck for mobile devices, but we need to set benchmarks for further evaluations).
- **Fine-Grained Labels:** Sometimes, what we're missing isn't raw accuracy—it's semantic clarity. I'd add labels like "wrist tilted left" or "shoulder slightly elevated" to guide the model better (Essentially, sub-pose estimation, rather entire movement)

D What optimization techniques would you use to run a real-time pose estmation model on mobile devices?

I feel this is where the real engineering fun begins, from research to real-world.

- **Quantization:** Move to 8-bit inference (INT8) without losing much accuracy.
- **Model Pruning:** Strip away unnecessary nodes—lighter model, faster response.
- **Convert Smartly:** Use ONNX or TensorFlow Lite and run it through hardware-optimized backends like CoreML or TensorRT.
- **Crop and Skip:** Don't process full frames every time—crop to ROI or skip every other frame if posture is stable.
- **Use Ready Models:** Honestly, BlazePose is fast because Google already did the hard work—I'd use that whenever possible.

E Let's say we need to track finger positions on a violin for precise motion feedback. How would you approach this problem?

Ah, finger tracking— this I would like to think of as a learner's nightmare (I took few classes for guitar) and the AI engineer's puzzle; either way I'd go with:

- **Model:** MediaPipe Hands is surprisingly solid. If I need better resolution, OpenPose's hand module could work.
- **Data:** FreiHAND, EgoHands, and synthetic fingerboard animations (Blender, Unity) for training (I googled them to get sense of it, but ideally, I would do more research before starting my work with). I also reckon Real-world data is scarce, so synthetic augmentation is critical.
- **Approach:** I'd start with a 2D hand model, then consider upgrading to a ViTPose variant only if I need long-range dependency modeling (like tracking finger tension or more subtle behaviours).

F Which AI architectures would you choose for real-time motion tracking on mobile?

My rule of thumb: start simple, then scale, and always remembering myself of my philosophy: K.I.S.S (Keep it simple sober)

- **Phase 1: Prototyping:** MediaPipe BlazePose or MoveNet Lightning. Proven, efficient, and works on most phones.

- **Phase 2: Experiment with complex models:** ViTPose is amazing for precision but heavy. If needed, I'd use MobileViT or a pruned/distilled version.
- **Phase 3: Can I find compromise?** CNN for detection, transformer for classification. Combine both only when needed, especially for per-frame semantic labels like "left thumb up."

G If we want to track a violinist's playing posture in real-time, what features would you extract?

I'd ideally go with geometry and some form of heuristics, but I need to do much research before concluding.

H What strategies would you use to keep FPS high while running real-time AI in a mobile game?

Here's what I'd do to keep the app running smooth:

- **Prune and Quantize:** Smaller models = less work per frame.
- **Frame Skipping:** Only run inference every N frames—interpolate or cache when nothing's changed.

References

[1] C. Lugaresi, J. Tang, H. A. Nash, M. McCormick, T. Denman, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, and M. Grundmann, "Mediapipe: A framework for building perception pipelines," *arXiv preprint arXiv:1906.08172*, 2019.

[2] TensorFlow Lite Team, "Movenet: Ultra fast and accurate pose detection." `https://www.tensorflow.org/lite/models/pose_estimation/overview`, 2021. Accessed: 2025-03-25.