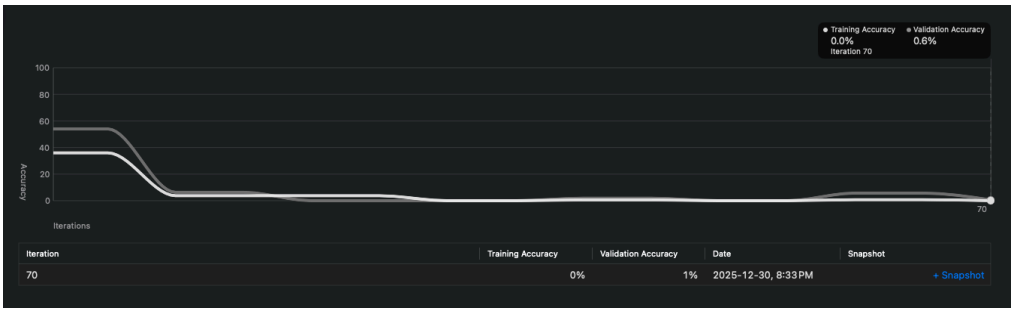
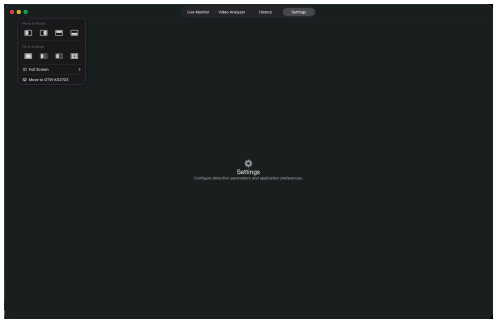
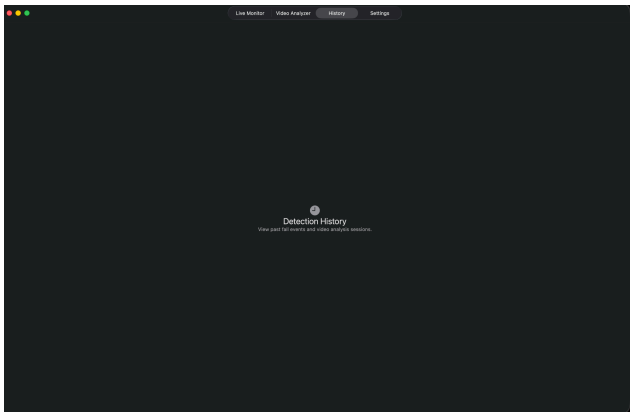
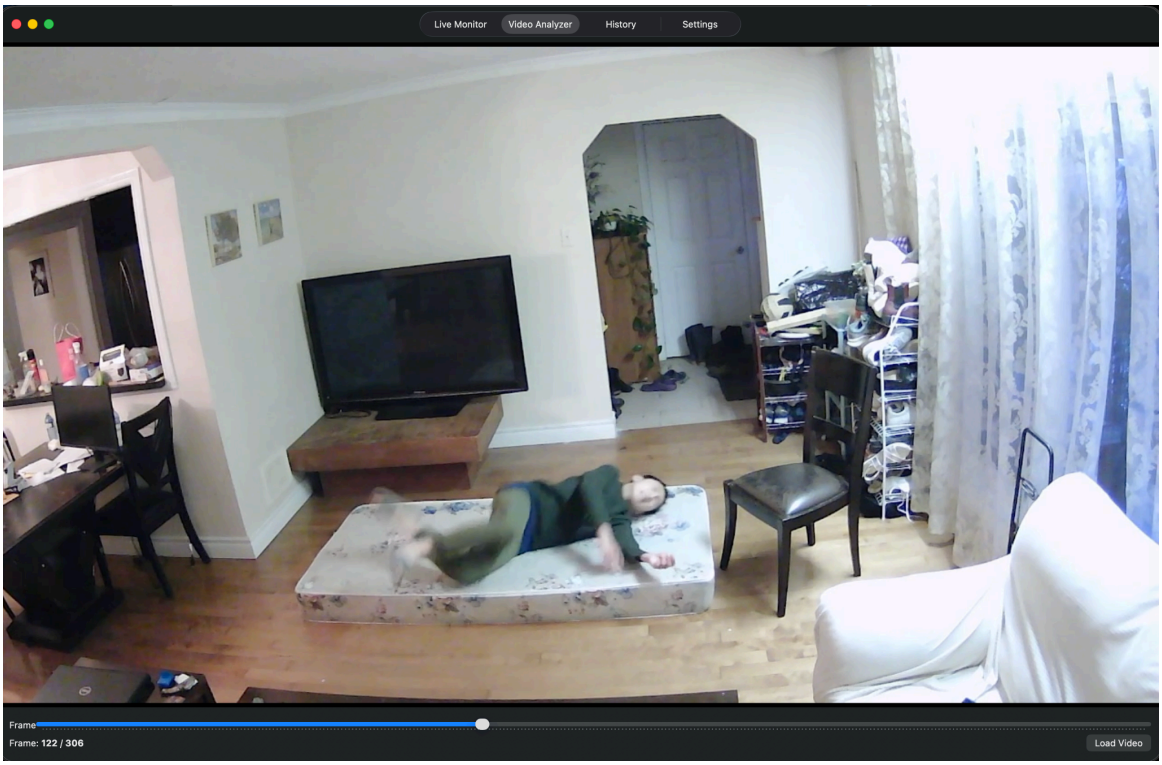
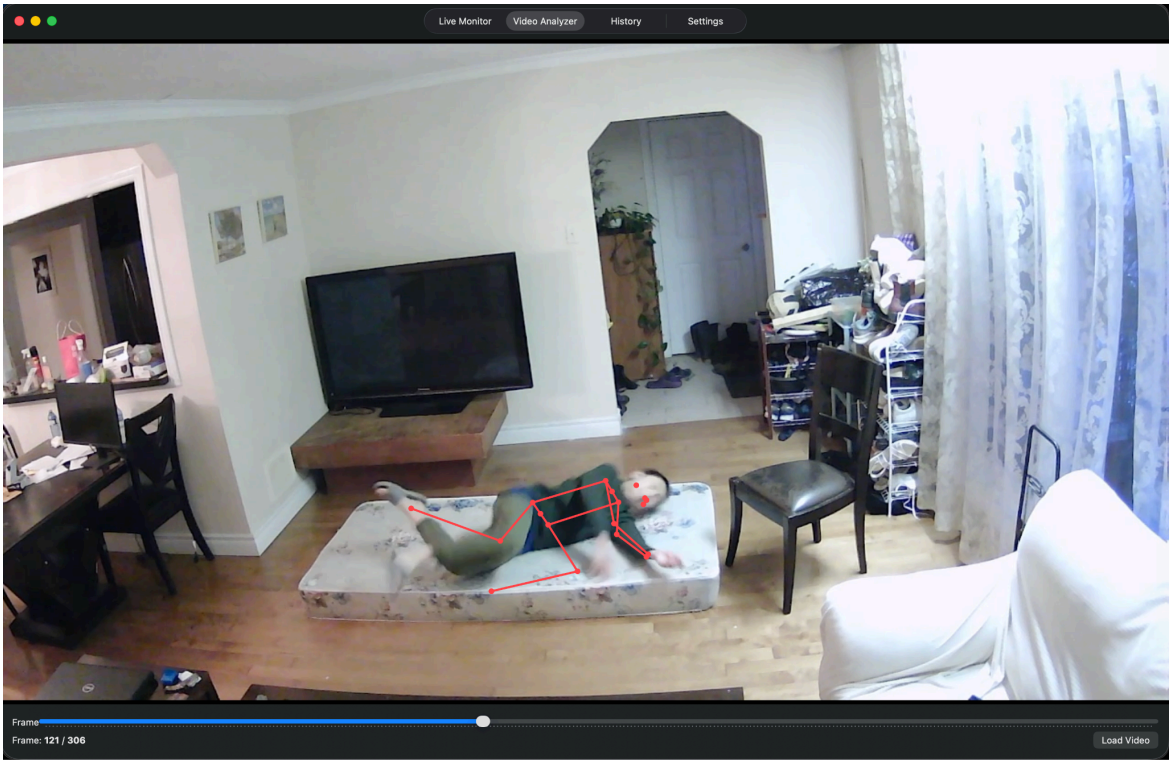
- 

- 



- Created a tab for video analyzing, inputting a .mov file and allowing users to see Apple's PoseNet model output
  - 

  - Saw that there were camera data gaps during high velocity events

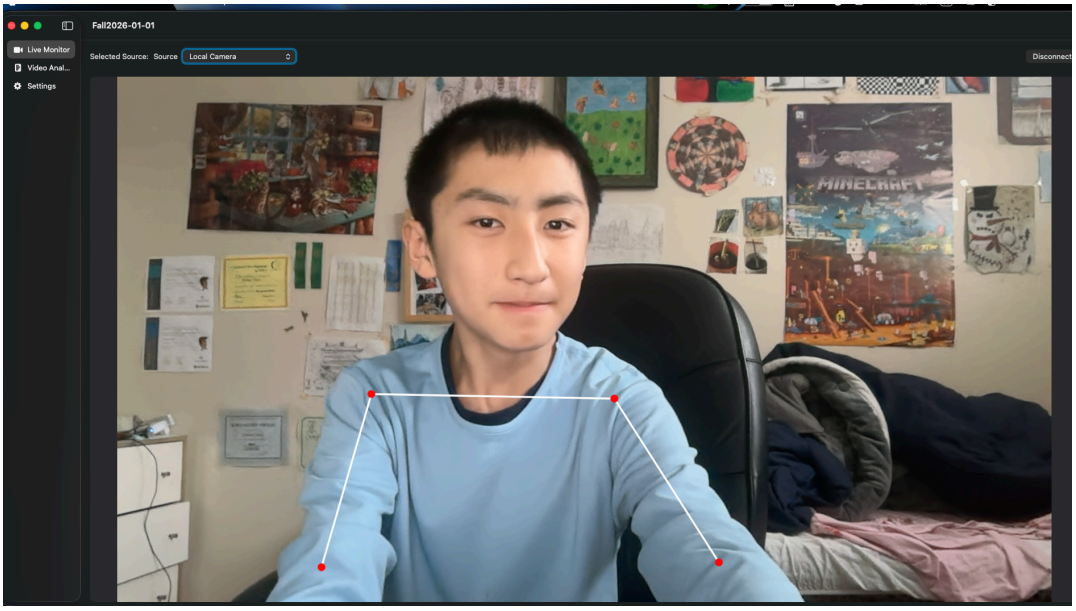
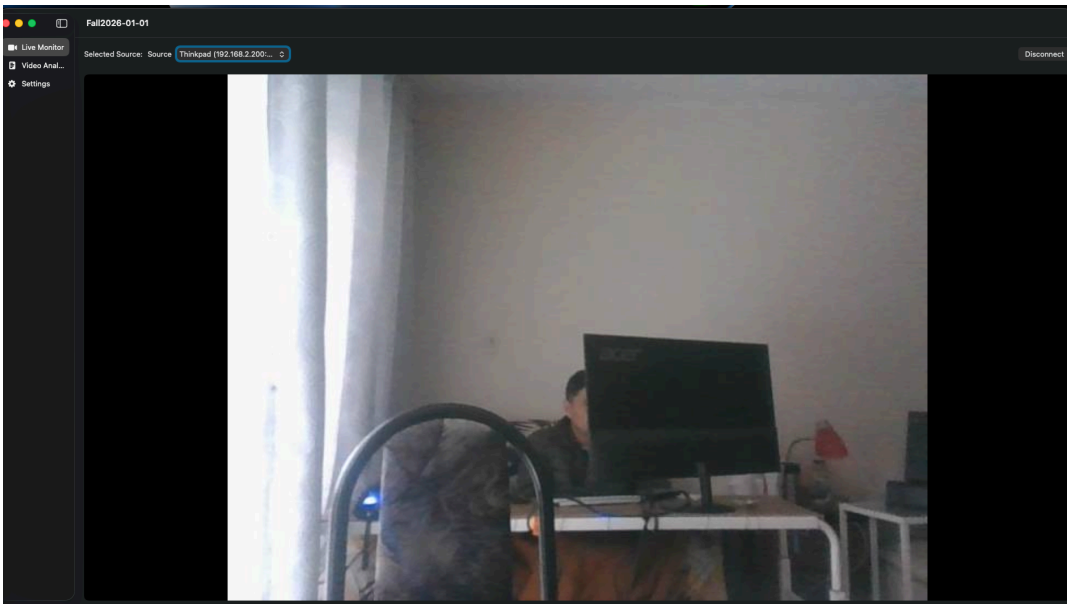
- 

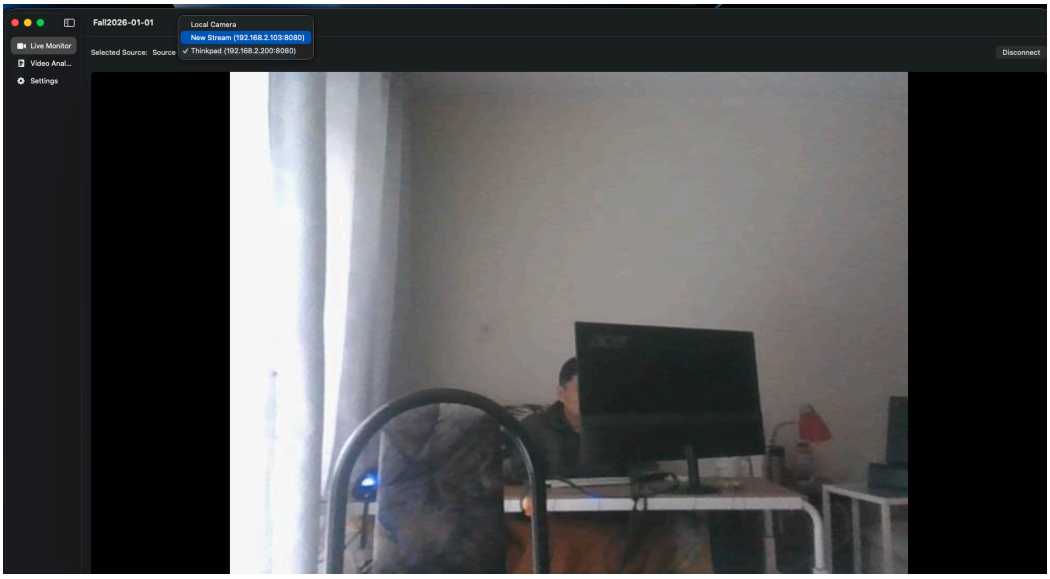
2026-12-31

- Built an extremely small jumpingjacks and marching CreateML Action Classifier Model to test model creation; model accuracy was 100% (with 6 videos). Questioned why the fall detection model was so bad.

2026-01-01

- Worked on a Swift app and incorporated socket communication with two Windows Computers as servers. Successfully established a stable camera feed to a MacBook.
    - Ran Apple's pose detection model on the video feed and was successfully able to display and get the results.

- 
Fall2026-01-01
Live Monitor
Video Anal...
Settings
Selected Source: Source
Local Camera
Disconnect
- 
Fall2026-01-01
Live Monitor
Video Anal...
Settings
Selected Source: Source
Thinkpad (192.168.2.200:...
Disconnect

- 

- Successfully implemented a simple jumping jacks and marching model into the app; will change the model for the fall detection model after finishing training.
- Thought of extra critical functions the final app and device must have:
    a two-way communication:
        - Functionality: Upon fall detection, the app should automatically open a two-way audio connection (like a speakerphone call) to a designated contact or a professional monitoring center (if you use one).
        - Value: This allows the responder to verbally check on the user ("Are you okay?"), confirm the emergency, and provide reassurance, even if the user can only respond with a whisper.
    - GPS/Real-Time Location Sharing
        - Functionality: If the user is mobile, the alert sent to the emergency contact(s) must include the user's precise, real-time GPS coordinates (latitude and longitude).
        - Value: This is non-negotiable for dispatching help, especially if the fall occurs outside the home.
    - Alert Customization and Escalation
        - Functionality: Allow the user to define a "Trust Circle" of contacts (e.g., family, neighbours, doctor). The app should follow a customizable chain of alerts (e.g., Text Contact 1, wait 30 seconds, Call Contact 2, Text Contact 3, etc.).
        - Value: Ensures that if the primary contact is unavailable, the alert escalates immediately.
    - User Cancellation Window

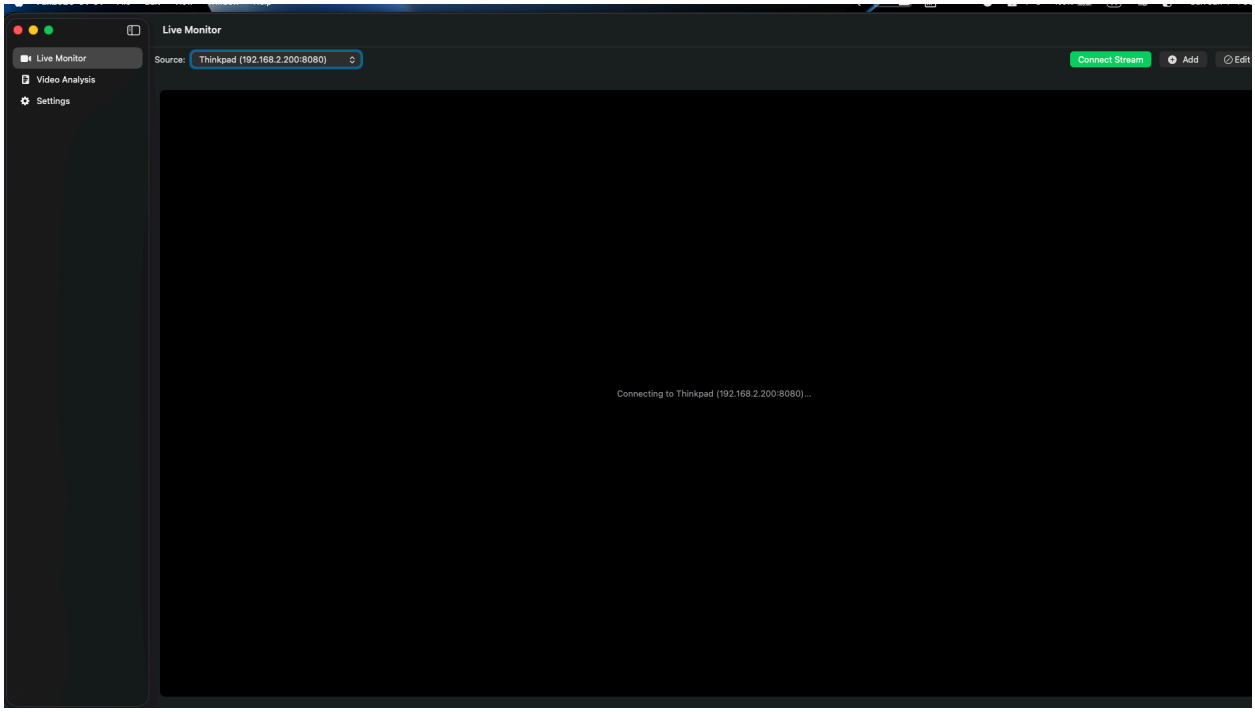
        - Functionality: After a fall is detected, start a short countdown (e.g., 10-30 seconds) during which the user can tap a large, simple "I'M OK" button to cancel the alarm.
        - Value: Prevents false alarms from high-impact movements, like sitting down quickly or dropping the phone.
    - Medical ID / Vitals Integration
        - Functionality: Allow the user to input critical medical information (medications, allergies, primary physician, DNR status) that is immediately accessible to responders once an alert is triggered.
        - Value: Provides potentially life-saving context to first responders.
    - Risk Assessment History
        - Functionality: Maintain a log of all detected falls (real and cancelled) with time, date, and location. Potentially flag periods of "Man-Down" (no movement for an extended period) as an anomaly.
        - Value: Provides valuable data for doctors or physical therapists to understand the circumstances and frequency of falls.
    - Add a small battery pack so the fall detection feed will still work in power outages.

2026-01-02

- Fixed several app-related bugs.
    - $$\text{Camera Feed} \xrightarrow{\text{Orientation/Mirroring Fix}} \text{Vision Pose Detection} \xrightarrow{\text{Stabilized Observation}} \begin{cases} \text{Drawing (Skeleton)} \\ \text{ML Prediction Pipeline} \end{cases}$$
- Experienced several problems where “marching” actions were detected much more frequently (~99+%) compared to jumping jacks.

2026-01-04

- Improve upon the app and successfully implemented one-way audio communication from the mac the windows/raspberry pi servers. Improved UI design

- 

2026-01-05

- Collected even more data and decided to group all of the negative, ADL labels into one label called “normal”.

Switching Back to a Physics-Based Fall Detection Approach

2026-01-12

- Decided to revert to old YOLO methods and scrap the entire LSTM RNN model training altogether.
- Experienced problems with normalizing coordinates to allow for seamless use in multiple resolutions.
    - Solution: calibration. Developed a standalone tool (height_finder.py) to manually identify a “Reference Standing Height” for specific camera resolutions.
        - Created a height_map.txt database to store these reference values
        - Normalized metric stability: the analysis program now uses a constant REF_HEIGHT for all velocity calculations. This ensures that a 10% drop in height results in a 0.1 velocity value regardless of whether the person is standing or lying down.

2026-01-13

- Decided that calibration with the hip width is not accurate; decided to use body scaling based on the torso height.

- I decided to have the laying down and moving fast indicators first.

- The new indicators:
  - Aspect ratio. It is calculated by dividing the width of the bounding box by the height. This is meant to determine whether the person is lying on the ground
  - Velocity. It is calculated by finding the change of torso divided by the torso length.
  - Torso angle. It is calculated with Theta = arctan2(dy, dx)
    - 90 degrees means the person is perfectly vertical (standing). 0 or 180 degrees means the torso is perfectly horizontal (lying down)
- During testing, I noticed that side-on views occasionally caused the YOLO model to lose track of one side of the body (e.g., the right shoulder being hidden by the torso).
  - I updated the normalization algorithm to use Asymmetric Pair Fallback. The system now checks for the left shoulder-hip pair first; if the confidence is low or the points are missing, it instantly switches to the right pair. This ensures the "Anatomical Ruler" remains constant even during body rotation.

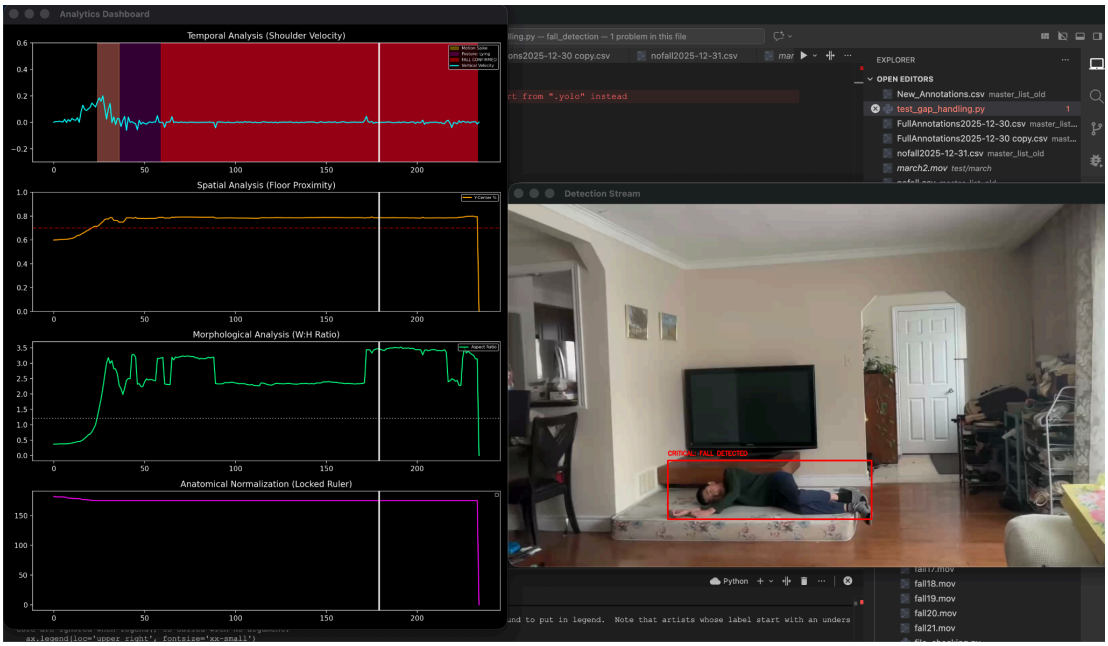
    - This change increased the data's **Continuity Score**, meaning fewer "zero-value" gaps in the velocity graph when a person turns during a fall.
    - Decided to use a running median or exponential moving average for torso length.
        - Instead of using *this* frame's torso length, the system looks at the last 15 frames and takes the median value. This ignores "wonky" detections where a hip disappears for a split second, keeping your ruler rock-solid.
- The final fall detection pseudocode logic:
    - Fall logic:
        - Trigger Phase (The Spike): Did the Vertical Velocity exceed the IMPACT_THRESHOLD?
        - Validation Phase (The Result): In the 1 second after that spike, did the person end up in a "LyingDown" state (Aspect Ratio > 1.2 AND Torso Angle < 45°)?
        - Recovery Check (The False Alarm Filter): In the 5 seconds after the fall, does the Aspect Ratio return to "Standing"? If yes, it might be a trip-and-stand or a "staged" fall. If they stay down, it's an emergency.

```
FOR each frame:
    GET all detected people (IDs)

    FOR each Person_ID:
        // 1. UPDATE THE RULER (Robustness)
        current_torso = Calculate_Torso_Distance()
        Person.Torso_History.append(current_torso)
        stable_ruler = Median(Person.Torso_History)

        // 2. CALCULATE PHYSICS
        velocity = (Current_Y - Last_Y) / stable_ruler
        angle = Calculate_Torso_Angle()
        ratio = Width / Height

        // 3. DETECTION LOGIC (The "Heuristic")
        IF (velocity > FALL_THRESHOLD):
            Person.Potential_Fall = True
            Person.Timestamp = Current_Time
```

```
        IF (Person.Potential_Fall == True):
            // Check if they ARE horizontal within 1 second of the spike
            IF (Current_Time - Person.Timestamp < 1.0 second):
                IF (ratio > LAYING_THRESHOLD AND angle < 45_DEGREES):
                    SIGNAL "FALL DETECTED"
            ELSE:
                // If 1 second passed and they are standing, it was just a
fast movement
                Person.Potential_Fall = False
```

    - Distinguishing Falls from Exercises
        - Pushups: The velocity is constant and rhythmic. A fall has a massive "acceleration spike" followed by a sudden stop (zero velocity).
        - Lying Down to Sleep: The velocity is low. The person "descends" slowly. The Aspect Ratio changes, but the IMPACT_THRESHOLD is never hit.
        - Rolling: The Aspect Ratio stays "Lying down" throughout. There is no vertical velocity spike from a standing height.
- Final:
    - I have finalized the fall detection as a Temporal State Machine to distinguish between actual falls and regular activities like sitting or exercising:
    - Trigger Phase (The Spike): The system constantly monitors vertical velocity. If the velocity exceeds the IMPACT_THRESHOLD, a "Potential Fall" event is timestamped.
    - Validation Phase (The Result): The system waits for a 1-second (30 frames) "Confidence Window" following the spike.
    - Stillness Verification: At the end of the window, the system checks two conditions:
        - Is the Aspect Ratio > 1.2?
        - Is the Torso Angle < 45°?
    - Final Confirmation: If the subject is horizontal and immobile after the 1-second window, the fall is confirmed. If the subject returns to a standing aspect ratio during the window, the trigger is discarded as a "Trip-and-Recovery" or a "Fast Sit."
    - Added velocity smoothing. Instead of a single frame, we now use a "Sustain Window." If a spike occurs, the fast_descent state stays active

for a set number of frames (e.g., 10 frames), creating solid blocks on the graph.

- 

2026-01-14

- Successfully implemented the Python fall detection algorithm on the Swift app.
- Decided on adding email notification functionality alongside the already implemented push notifications
-