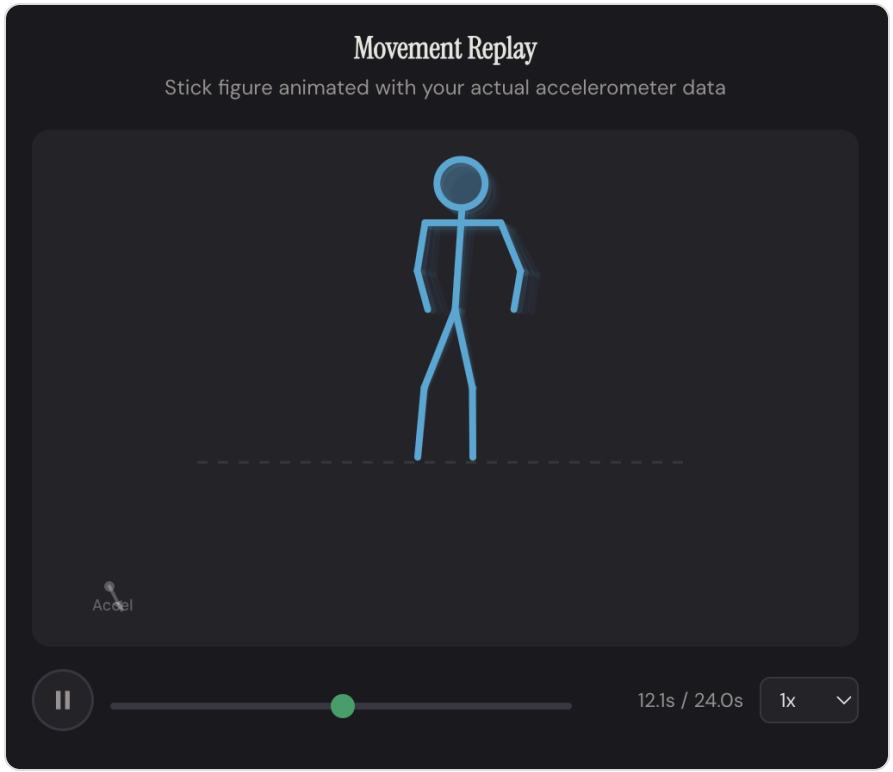
Manik S

Team t2-WR

My Work on Romberger

- **Initial ML pipeline (the inspiration)** - built the standing-vs-sitting classifier on the Kaggle MotionSense dataset (24 subjects, 17 activity folders), reaching **91.5% LOSO** with a logistic regression on 8 sway / jerk / path-length features. This served as the proof of concept that the team's Romberg classifier was modeled on.
- **Team Romberg classifier** - taking my Kaggle pipeline as the template, the team collected our own balance recordings, chopped them into 30-second windows, and Syed and Sophia trained the eyes-open vs. eyes-closed model per Thea's specification, switching from logistic regression to **SVM Linear** for lower per-fold variance (162 windows, 10 subjects, **73.9% LOSO**). I handed off my long recordings into the pooled dataset and integrated the resulting weights into the frontend.
- **Frontend - my specialty throughout the project** - built and iterated on `index.html` from day one: CSV upload + drag-and-drop, schema tolerance for three accelerometer formats (Sensor Logger iOS, MotionSense, Android), client-side feature extraction in JS, prediction UI, dark mode, demo mode, and the full layout / styling.
- **Stick-figure Movement Replay** - designed and implemented the canvas animation that replays the user's recorded sway in real time. Lets users *see* what the classifier saw, which is what makes the Romberg result feel



Movement Replay - stick figure animated from the user's actual accelerometer trace. My contribution to the visualization layer.

T-Shaped Skills, Evolved

How my skill profile changed over the long project

A T-shape means **deep expertise in one area** and **broad working knowledge across many**. The vertical of my T deepened in **frontend / client-side ML**; the horizontal widened across Python ML, signal processing, sensor-data handling, and technical writing.

Deep (vertical)

- **Frontend web development** - single-file `index.html`, CSS variables for theming, responsive layout, drag-and-drop, dark mode
- **Client-side ML inference in JS** - porting Python-trained scaler + weights into ~10 lines of dot-product JavaScript with feature parity to training
- **Canvas animation** - DPR-scaled stick-figure renderer driven by raw accelerometer arrays, EMA smoothing, scrubber + speed controls
- **CSV schema negotiation** - runtime detection of three different accelerometer column conventions

Broad (horizontal)

- **Python / Jupyter** - feature extraction, LOSO cross-validation, logistic regression / SVM training
- **Signal processing** - sway magnitude, jerk, path length, RMS, kurtosis, skewness
- **Data collection** - Sensor Logger recordings, schema cleanup, subject pooling
- **UX iteration** - demo mode, mobile overflow fixes, theme-aware Chart.js re-render
- **Technical writing** - research paper (rpaper) on the SE tradeoffs of client-side ML
- **Git collaboration** - branch hand-offs with Syed (ML), Sophia (data), Jack (Supabase)

Evolution: Started the semester comfortable in Python notebooks but new to integrating ML into a real web product. Mid-project I moved into JavaScript and rebuilt the inference path so training (Python) and serving (browser) matched feature-for-feature. By the end I was treating the frontend as the harder engineering surface - schema handling, canvas rendering, theme parity - and the model itself as a 6-number dot product.

Initial nnnn: `manik43@pe15:~/cs410/3622.txt`

Additional Comments & Plans

What's next for Romberger and for me

Reflections on the long project

- The hardest engineering problem turned out to be the **data pipeline**, not the model - auto-trimming, duplicate detection across team repos, and feature parity between Python and JS were the real work.
- **Data was the bottleneck** - chopping each recording into multiple 30-second windows was a last-resort move to stretch a 10-subject dataset. With more time, more raw data sources from more people would have been the cleaner path to higher accuracy.
- Going **fully client-side** (GitHub Pages, no backend) was the right call: zero hosting cost, privacy by default, and easy demos.
- Splitting the work was effective: I led frontend + the standing/sitting model, Syed rebuilt the Romberg pipeline on 10 subjects, Jack prototyped Supabase, Sophia drove data collection.

Plans / future work

- Push subject count past 10 - accuracy will plateau or climb, and we'll know which.
- Mobile-first record-in-browser flow so users don't need Sensor Logger as a middle step.
- Hook the eyes-open/closed model into the same Movement Replay so users see the classifier's reasoning, not just its label.
- Try a small CNN on raw windows instead of hand-crafted features - keep it client-side via TF.js.

Research paper

Title: Engineering a client-side sensor-data pipeline for browser-based health screening applications.
Argues that for this class of app the *pipeline* (consolidation, cleaning, windowing, schema tolerance, feature parity) is the load-bearing engineering work, not the choice of classifier.

Paper PDF: `manik43@pe15:~/cs410/3622.pdf` · **Source:** `manik43@pe15:~/cs410/rpaper.txt`