WORKING WITH AI · SESSION 2 · HOMEWORK

Rebuild the Detector — the nine prompts

These are the exact prompts used in the session, in order. Paste one, wait, look at what came back, then paste the next. Nothing is hidden — a tool without its prompts isn't reproducible, and that's the whole point.

Prompts 1–9 generate their OWN synthetic recording — they do not use the practice TIFF. Two reasons. The practical one: a page running in a browser cannot reach into a chat attachment, so a tool either makes its own data or asks you to pick a file off disk (that is Prompt 10). The real one: when Claude plants the events, **you own the ground truth** — you know exactly what is in there, so you can score the detector against truth instead of against your own eyeballs. Your numbers will differ from the room's. That is correct, and it is the entire lesson.

Where the practice files come in. Session2_Practice_Linescan.tif is used two ways: by Prompt 10 below, which teaches your tool to load it; and by Part A of the homework guide, where you recover the spark times and only then check them against the answer key. Plant-and-score, then recover-and-verify — two different halves of the same skill.

PROMPT 1 Describe the recording

Everything downstream depends on this. Every number stated, no analysis asked for.

```
I'm going to have you build a calcium-imaging analysis tool step by step.
Here is the full context, in four dimensions:

SAMPLE: adult mouse ventricular myocyte, resting, loaded with Fluo-4.
IMAGING: confocal line-scan, 256 spatial rows x 750 time columns, 500
  lines/second (2 ms/column, 1.5 s total), 0.30 um/pixel (~77 um line),
  pixel values are raw fluorescence F.
STRUCTURE: sparks are small/local/brief (~10 ms rise); a transient is
  global and slower (~40 ms rise); a wave propagates (a diagonal streak).
QUESTION: detect and classify these events, then measure properties that
  go beyond amplitude and frequency.

Read this back in your own words and flag anything ambiguous first.

After completing these instructions, stop and wait for the next rung before doing anything.
```

PROMPT 2 Generate & display — and your first owned decision

You choose F_0. Try all three options and watch the numbers move: an event inside your baseline window corrupts every $\Delta F/F_0$ after it.

```
Build a self-contained HTML page that generates a synthetic line-scan
matching the description (record the true event positions internally as
ground truth).

Make it realistic. A real line-scan also contains things that are NOT
events, so plant these too:
  - 4 brief bright flickers: 1-2 columns in time, high amplitude,
    several um across (like a noise spike or a dye blob)
  - 3 narrow sustained streaks: exactly 1 pixel of space, ~50 columns
    long, high amplitude (like a stuck detector pixel)
Put the artifacts in QUIET stretches of the recording, clear of the real
events -- an artifact sitting on top of an event merges with it. Keep them
out of the first 100 ms too: that window is the F0 baseline, and an artifact
inside it corrupts that row's baseline for the whole recording.
Record these internally as ARTIFACTS, not as ground-truth events.

Then:
1. Let me CHOOSE the F0 baseline window from a menu: first 100 ms, first
   300 ms, or a per-row rolling minimum. Compute F0 per row over that
   window, then delta-F/F0 = (F - F0)/F0.
2. Render delta-F/F0 as a hot-colormap heatmap (time x, space y).
Show me the heatmap. Make F0 a visible control, not a hidden default.

After completing these instructions, stop and wait for the next rung before doing anything.
```

PROMPT 3 Naive detection — watch it fail

One global threshold. You'll get several hundred 'events'. Drag the slider DOWN and watch the count FALL — no threshold rescues this.

```
Here is a page I built earlier that generates a synthetic calcium line-scan
and displays delta-F/F0 as a heatmap. Extend it, keeping everything it
already does.

Add the simplest possible detection: a single global delta-F/F0 threshold
```

```
(slider, default 0.30). Draw a box around every connected above-threshold
region. No size filter, no other criteria. Show the overlay and the count.

After completing these instructions, stop and wait for the next rung before doing anything.
```

PROMPT 4 Real detection — three gates you own

THE key step. Toggle each gate on alone and watch which junk it removes. Amplitude kills the noise speckle; duration kills the brief flickers; width kills the narrow streaks.

```
Now add three separate, independently-toggleable gates, each a real
decision about what counts as an event:
  AMPLITUDE gate: peak delta-F/F0 must exceed 1.5x the threshold.
  DURATION gate: the event must last at least ~6 ms.
  WIDTH gate: the event must span at least ~1.2 um.
Give each its own checkbox and slider so I can turn them on one at a time.
Also: classify a region spanning more than half the cell's rows as a
TRANSIENT, not a spark. Show a detections table with per-event properties.

After completing these instructions, stop and wait for the next rung before doing anything.
```

PROMPT 5 Nyquist check — measured, not assumed

Everything should read RESOLVED. That's the right answer: 500 lines/s genuinely captures a 10 ms rise. The point is you can now prove it.

```
Add a Nyquist-check panel. For each detected event, measure its 10-90%
rise time from its own delta-F/F0 trace, convert that to an effective
bandwidth (f_bw = 0.35 / rise_time), and compare to the Nyquist frequency
(half the line rate = 250 Hz). Report a continuous margin (Nyquist / f_bw)
-- no hard cutoff. Also show each event's FFT power spectrum as a small
sparkline with Nyquist marked, and the measured points-on-rise as a
companion. Cite the SparkMaster 2 spark kinetics (TTP ~15 ms) as a
published reference band.

After completing these instructions, stop and wait for the next rung before doing anything.
```

PROMPT 6 Ask Claude HOW to test it

Don't skip and don't hand it a recipe. Let it propose the methodology — known ground truth, precision/recall — before any code. This is the habit worth taking away.

```
Before I trust this detector, I want to test it properly. What's the best
way to test whether it actually works, and how would I verify the results
of that test? Propose the method before writing any code.

After completing these instructions, stop and wait for the next rung before doing anything.
```

PROMPT 7 Now implement the test

Reveal the planted events and score. Expect every real event found PLUS a couple of false positives. That honest score beats a perfect one.

```
Good. Now implement that: since this is synthetic data with known planted
positions, add a "reveal ground truth" button that overlays the true
events and reports matched / missed / false-positive counts.

After completing these instructions, stop and wait for the next rung before doing anything.
```

PROMPT 8 Metrics — the physiologist's panel

Amplitude should match what was planted. Timing should scatter around it without bias — which is exactly what the Nyquist step predicted.

```
Add a metrics panel. Per spark: amplitude, time-to-peak, FDHM, FWHM,
decay tau, and signal mass (amplitude x spatial width x duration, the
SparkMaster convention). Population: spark frequency (per 100 um per s)
and transient frequency (per s). Show means across events.

After completing these instructions, stop and wait for the next rung before doing anything.
```

PROMPT 9 Waves — detection becomes discovery

The velocity is the doorway: from a number you detected to a question you couldn't have asked before.

```
One event type is left: the wave. A wave propagates, so its bright ridge
drifts across the cell as time advances (a diagonal streak). For any
elongated region whose ridge drifts substantially, label it a WAVE and
measure the slope of that drift as a propagation velocity in um/ms.
```

```
After completing these instructions, stop and wait for the next rung before doing anything.
```

PROMPT 10 Point it at real data · the capstone

This is where it stops being a demo. Everything so far ran on data Claude made up; now it has to survive a file it has never seen. Note what the tool loses the moment you load a real recording — that loss is the honest part.

```
Now make the tool work on a real file instead of the one it generates.

Add a file picker (an <input type="file">) that loads a linescan TIFF from
my disk. The practice file is Session2_Practice_Linescan.tif:
  - 16-bit grayscale, uncompressed, single channel, little-endian
  - 256 rows (space) x 750 columns (time)
  - pixel values are raw fluorescence counts in the high hundreds to low
    thousands -- they are NOT 0-255, so do not assume 8-bit or everything
    will clip to white
Browsers cannot decode TIFF natively, so use UTIF.js from cdnjs.

Requirements:
1. Let me enter the calibration for the loaded file: line rate (lines/s),
   pixel size (um/px). Default them to 500 and 0.30 but let me change them.
2. When a file loads, recompute F0 and delta-F/F0 with my CURRENT F0 window
   choice, and re-run detection with my current gate settings. Everything
   downstream should just work on the new data.
3. IMPORTANT: once a real file is loaded there is no planted ground truth
   any more. Disable the reveal-ground-truth control and say why, rather
   than showing stale markers from the synthetic recording. The tool must
   not claim to know something it doesn't.
4. Keep the generated recording available -- a button to switch back.

Then tell me honestly what the tool can and cannot verify once it is
looking at a file whose contents it did not plant.

After completing these instructions, stop and wait for the next rung before doing anything.
```

When it's built

Compare it against Session2_LiveBuild_Reference_Detector.html — the stepper from the session. Yours will differ. Where it differs, ask why, and ask Claude. The differences are the interesting part.