

# 5. Tools to Measure Software Energy (lab)

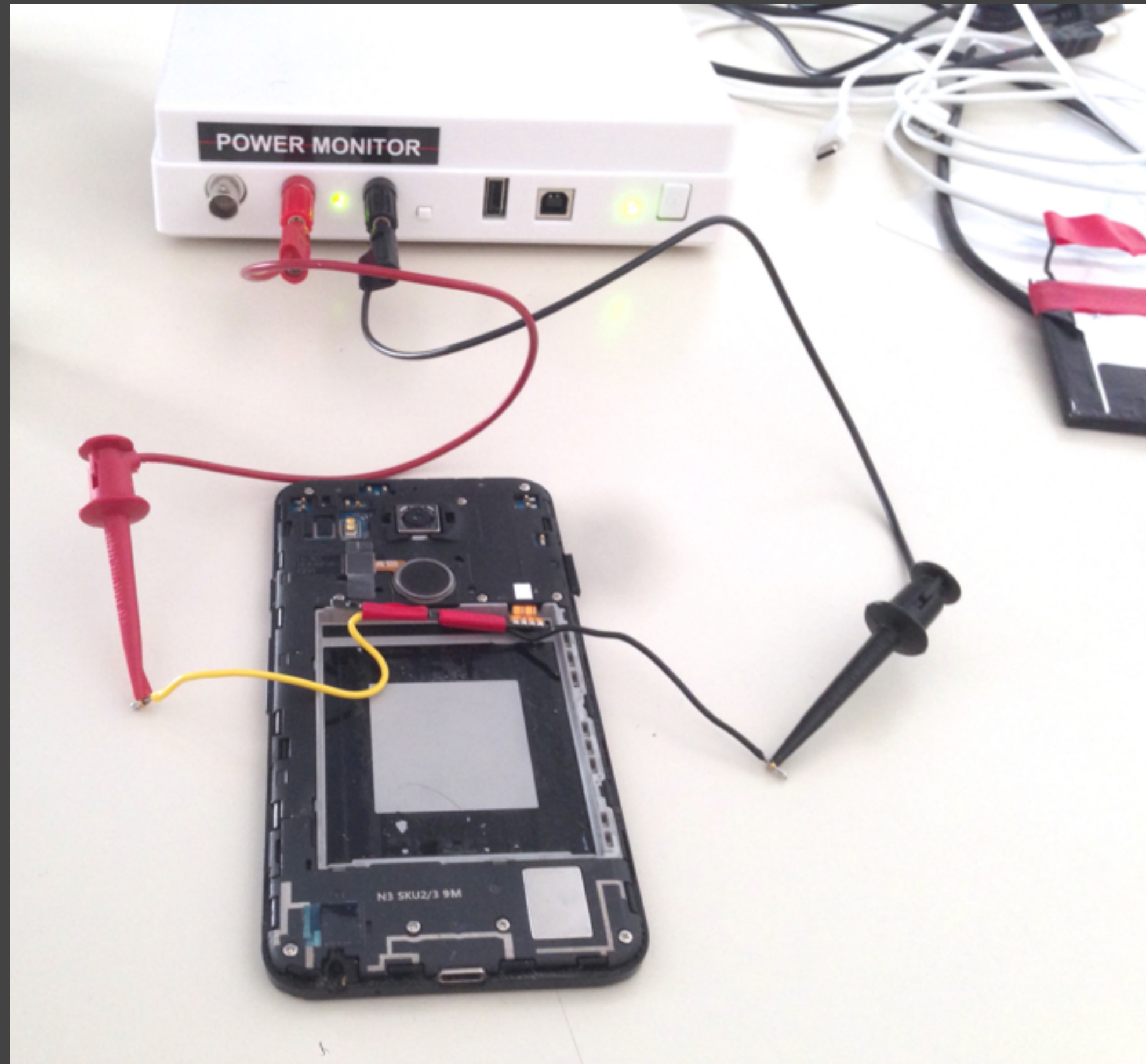
Sustainable Software Engineering  
CS4295



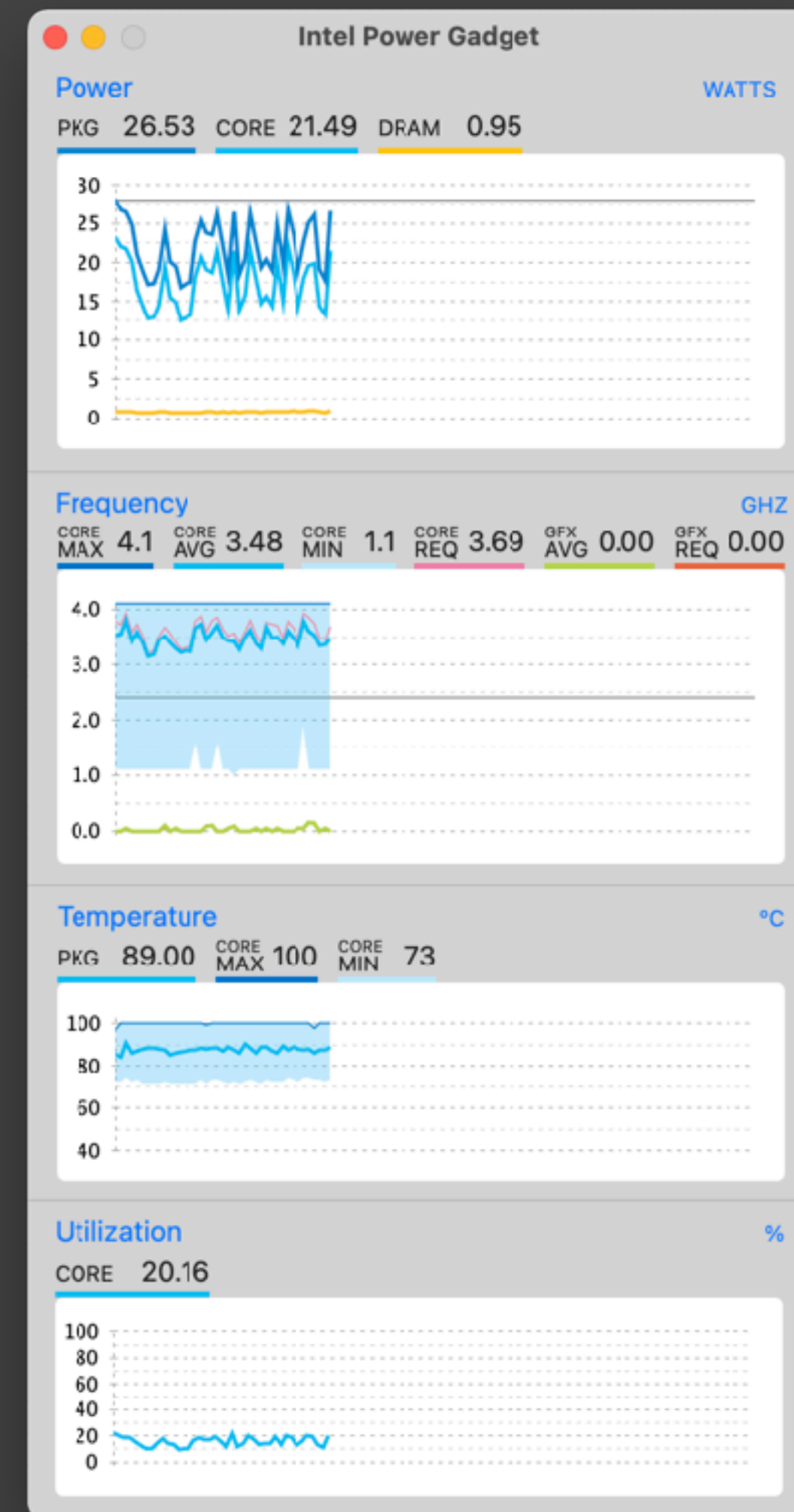
Luís Cruz  
[L.Cruz@tudelft.nl](mailto:L.Cruz@tudelft.nl)

1. Tools
2. Hands-on
3. Project 1

# Hardware Power Monitors



# Energy Profilers



# Hardware Power Monitors

- Connects directly to the power source of the device/  
component.
  - Some power monitors also replace the power source.
- Example:
  - **Monsoon Power Monitor** (for IoT and smartphones).
    - Can be fully automated using a Python API.
    - It measures and powers small electronic devices.
- There are many power/energy meters out there but for **software use cases** we need to be able to **control them using an API**.





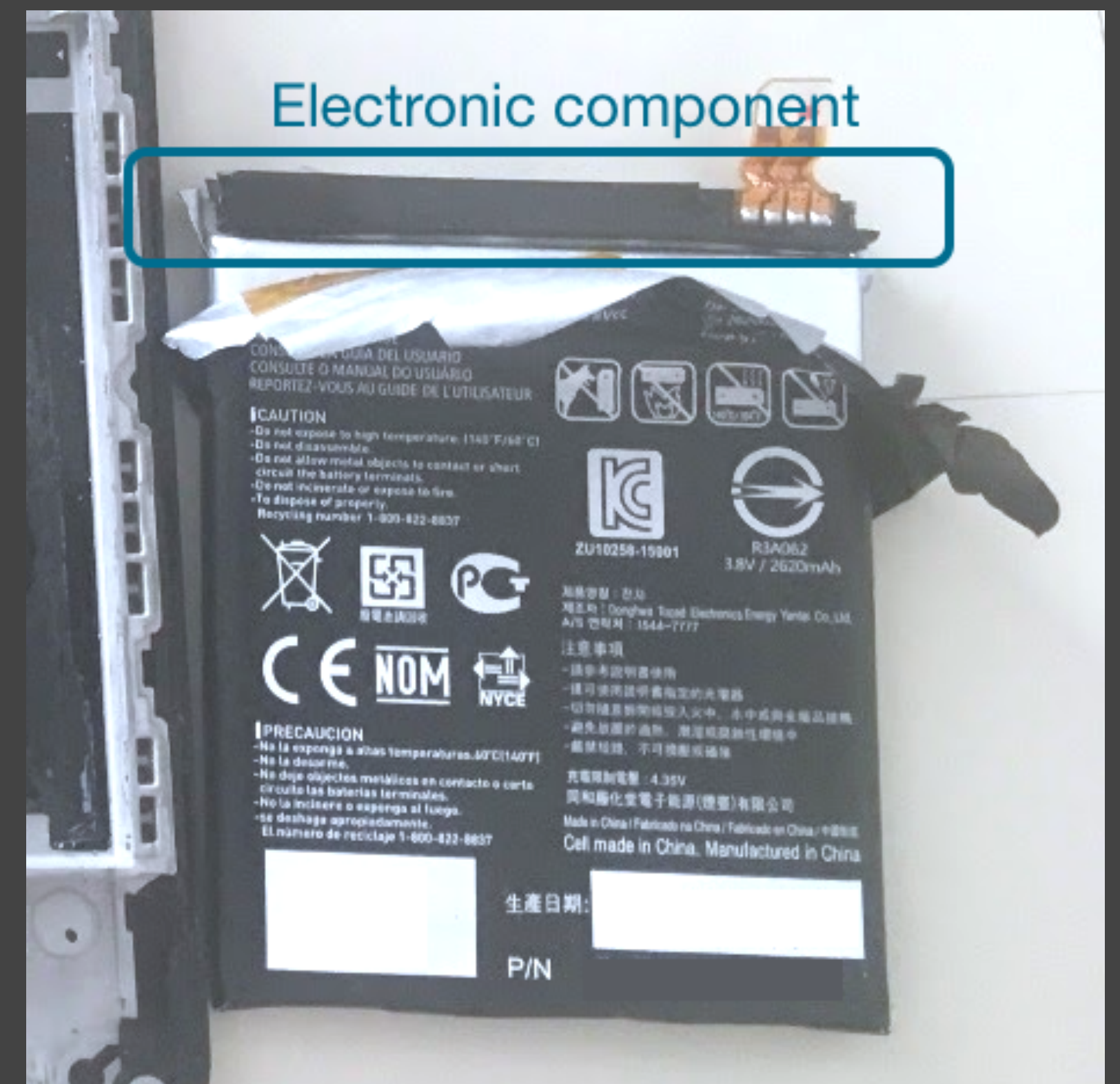
# Connecting Monsoon to a Smartphone

- 1. Disassemble the smartphone and **find the connectors** of the battery.
- **iFixit** usually has nice tutorials and blueprints.  
<https://www.ifixit.com>



# Connecting Monsoon to a Smartphone

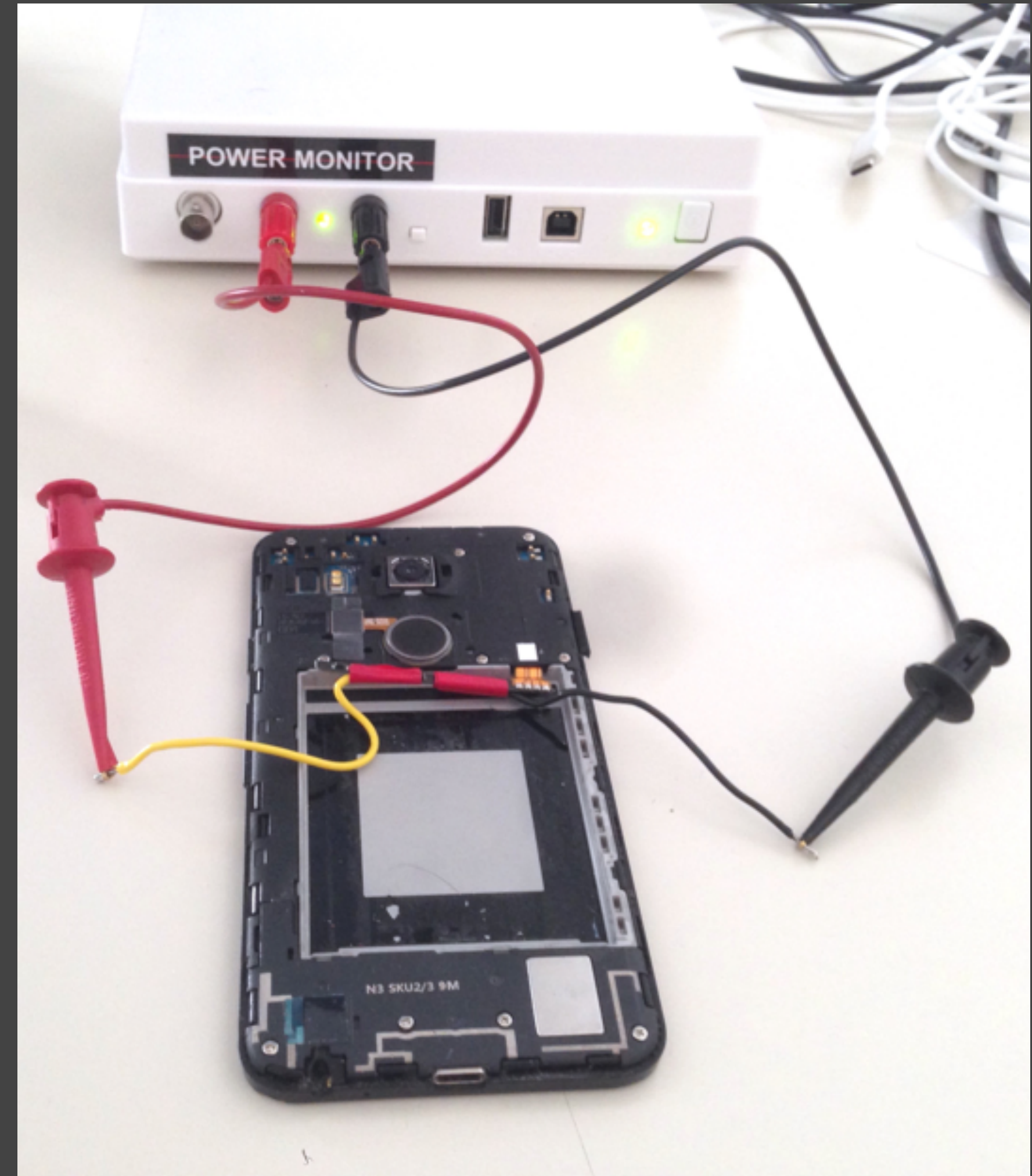
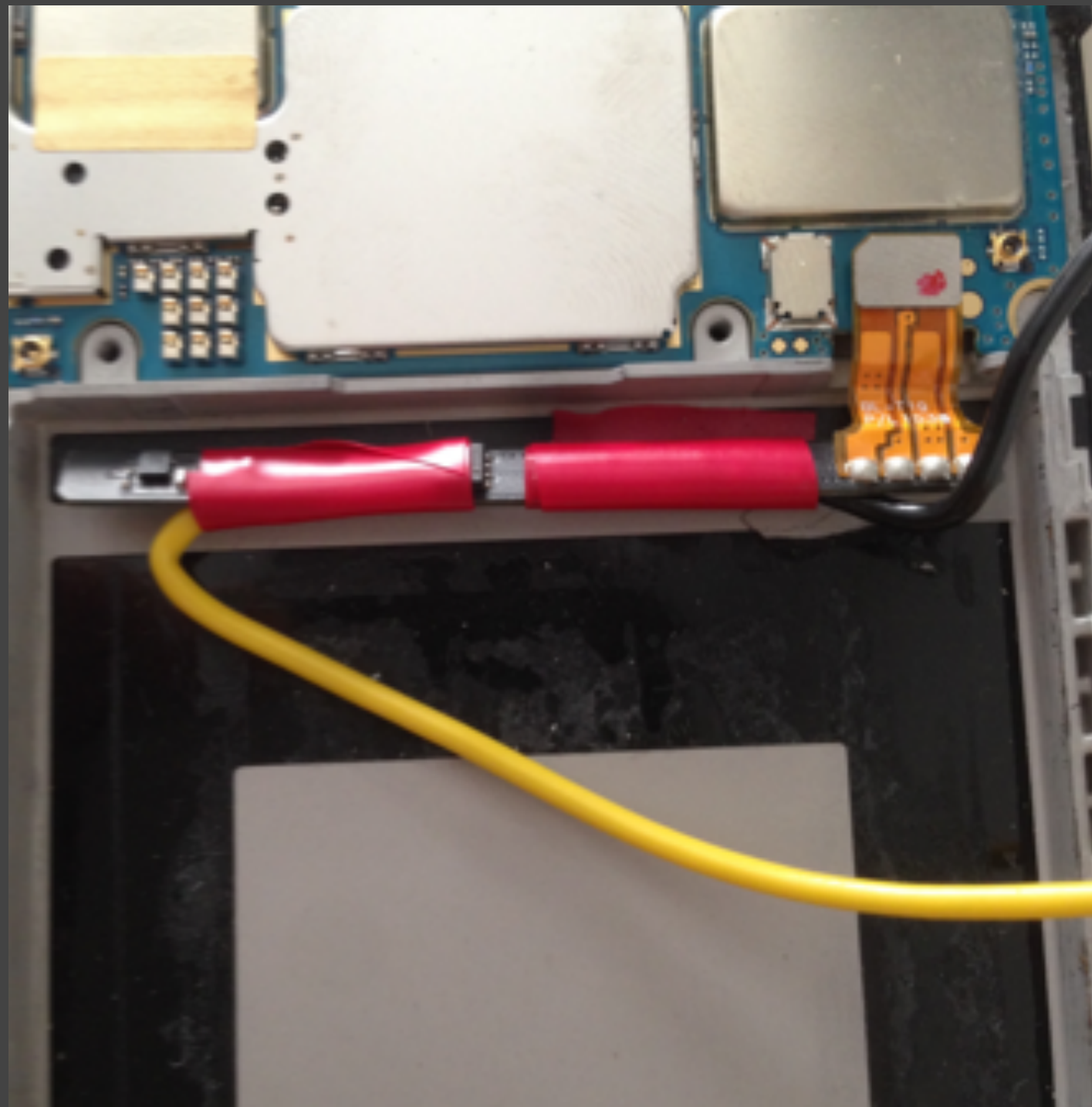
- 2. Extract the electronic component of the battery
- Modern batteries are connected through **4 terminals**:
  - **Positive**
  - **Negative**
  - **BTEMP**, battery temperature (used for safety)
  - **BST**, battery system indicator (provides info about the battery)
- Hence, one cannot simply connect + and - pins





# Connecting Monsoon to a Smartphone

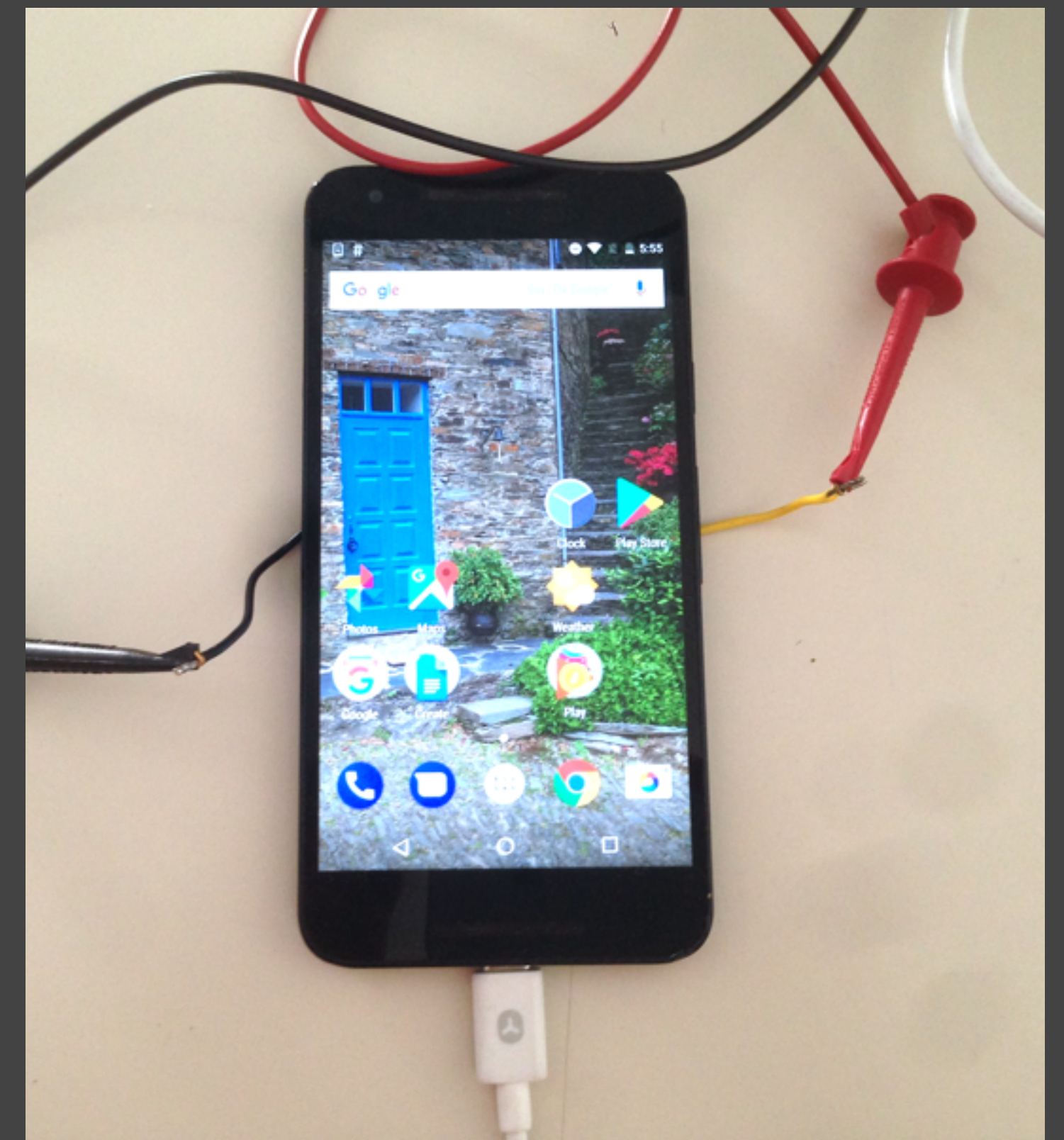
- 3. Connect the electronic component directly to the monitor.





# Connecting Monsoon to a Smartphone

- 4. Use the library **PyMonsoon** to control the power monitor.
  - <https://github.com/msoon/PyMonsoon>
  - 4.1. Set the monsoon to desired Voltage. Choose the **typical voltage** of the **original battery**. For the Nexus 5X, **3.8V** was equivalent to its battery at around 60% capacity.
  - 4.2. Start measuring





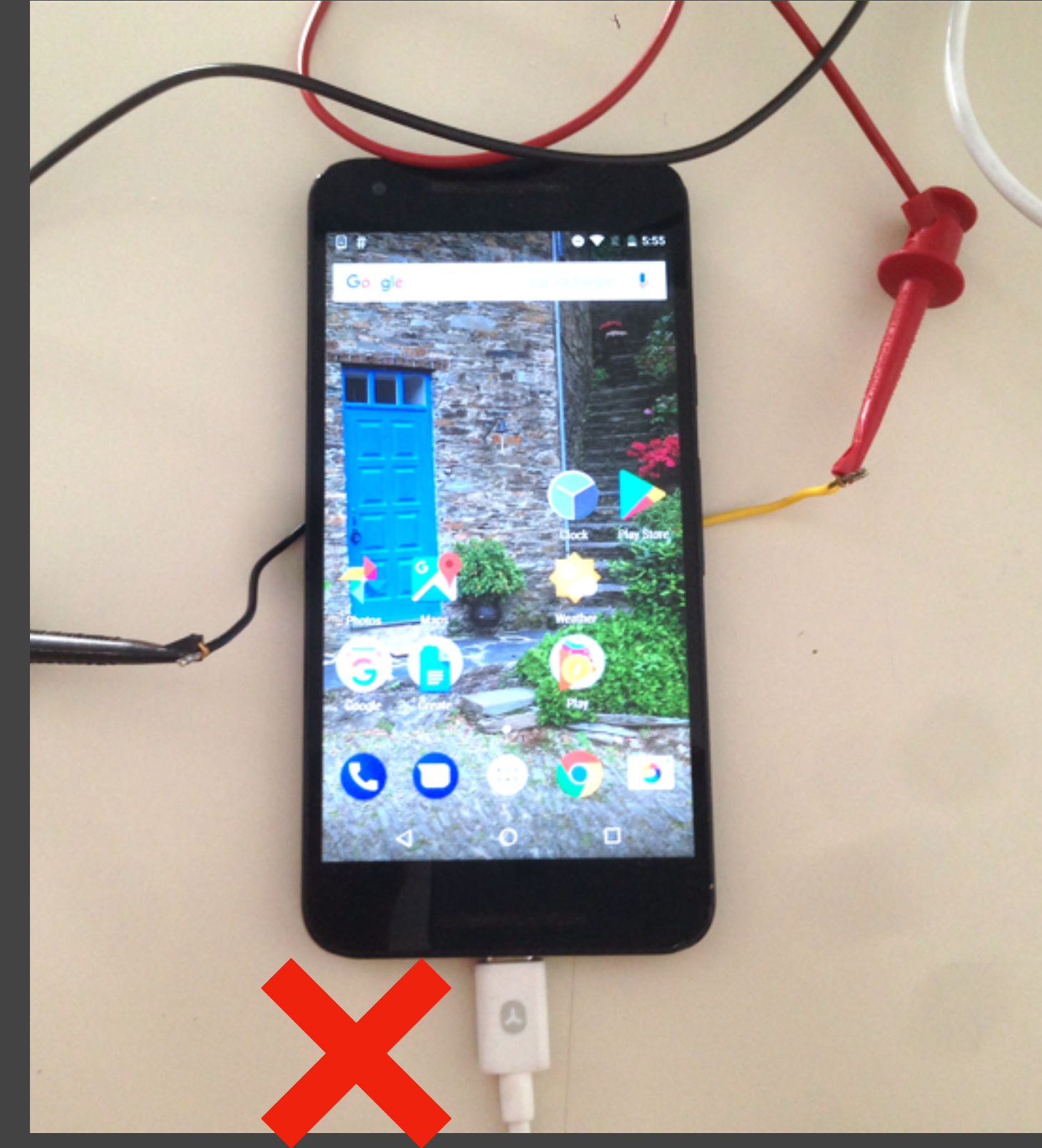
# Connecting Monsoon to a Smartphone

- 5. Automate User Interface interaction
  - The last thing you want to do is to manually interact with the smartphone while you measure energy consumption. Tests are **less accurate, less reproducible**, and, in this case, **the screen cannot not be easily accessed**.
  - Tools to automate interaction with Android phones:
    - To open, install, close apps: **adb**
    - To interact with the app: **Appium, Robotium, UIAutomator, espresso**, etc.
    - Alternative: **physalia** automates all adb interactions and PyMonsoon calls.



# Issue 1: USB cable!

- **You need the USB cable** to automate the interaction with the phone.
- When you connect the USB cable, measurements become **unreliable**.
- **Solution:**
  - Monsoon has a feature to control the USB connection (switch on/off)
    - **Option 1:** Right before starting measurements, the USB connection is stopped.
      - Works fine when when all the interaction instructions can be sent in advance and the time for the execution is already known.
    - **Option 2:** using USB, set up a **wireless ADB connection**. Stop USB connections afterwards.
      - How to: <https://stackoverflow.com/a/3623727>





# Issue 2: your app is not exclusive

- Many activities run in a smartphone device. E.g., getting push notifications, checking nearby bluetooth devices, etc.
  - Moreover, brightness may change according to environment. Different screen brightness, different results.
- You need to **reduce tasks to the bare minimum**:
  - Set brightness to a fixed value; turn off notifications, kill all user-owned processes, turn off cellular data, bluetooth, location services, account syncs; uninstall all unnecessary apps, etc.



When it comes to desktop/cloud software, the sources of **noise** are different but **the same concerns apply**.

**Each case is different** – think it through!



# Energy Profilers

- **Simple setup!** Quite reliable (if you choose the profiler wisely).
  - Recently, they are starting to rely on internal power sensors.
- Still **sensitive to noise** from concurrent processes/tasks! ⚠

# Examples of Energy Profilers



<https://www.websitecarbon.com>

**How is your website impacting the planet?**

**Estimate your web page carbon footprint:**

Your web page address

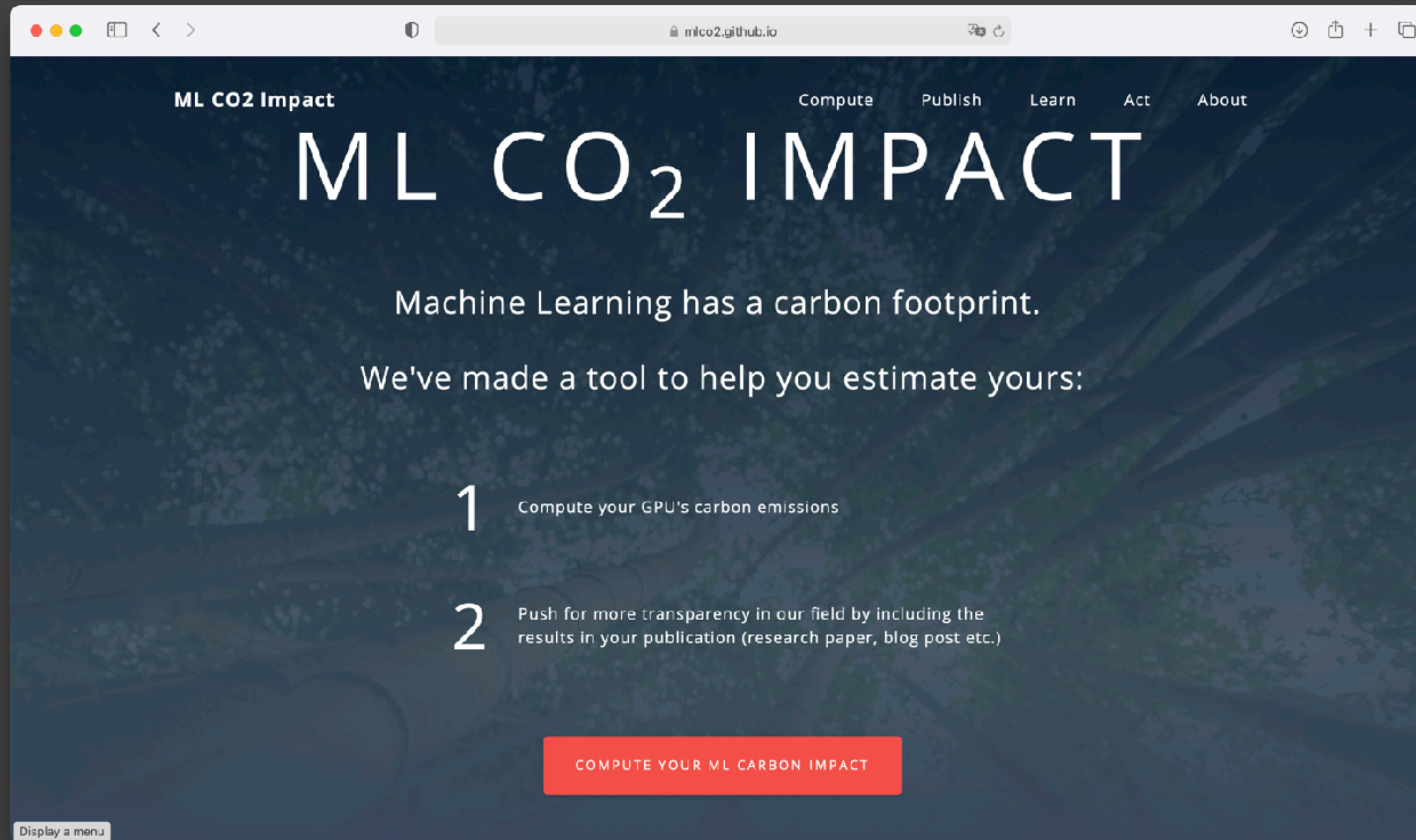
Web page URL

**Calculate**

By using this carbon calculator, you agree to the information that you submit being stored and published in our public database.

Display a menu

<https://mlco2.github.io/impact/>





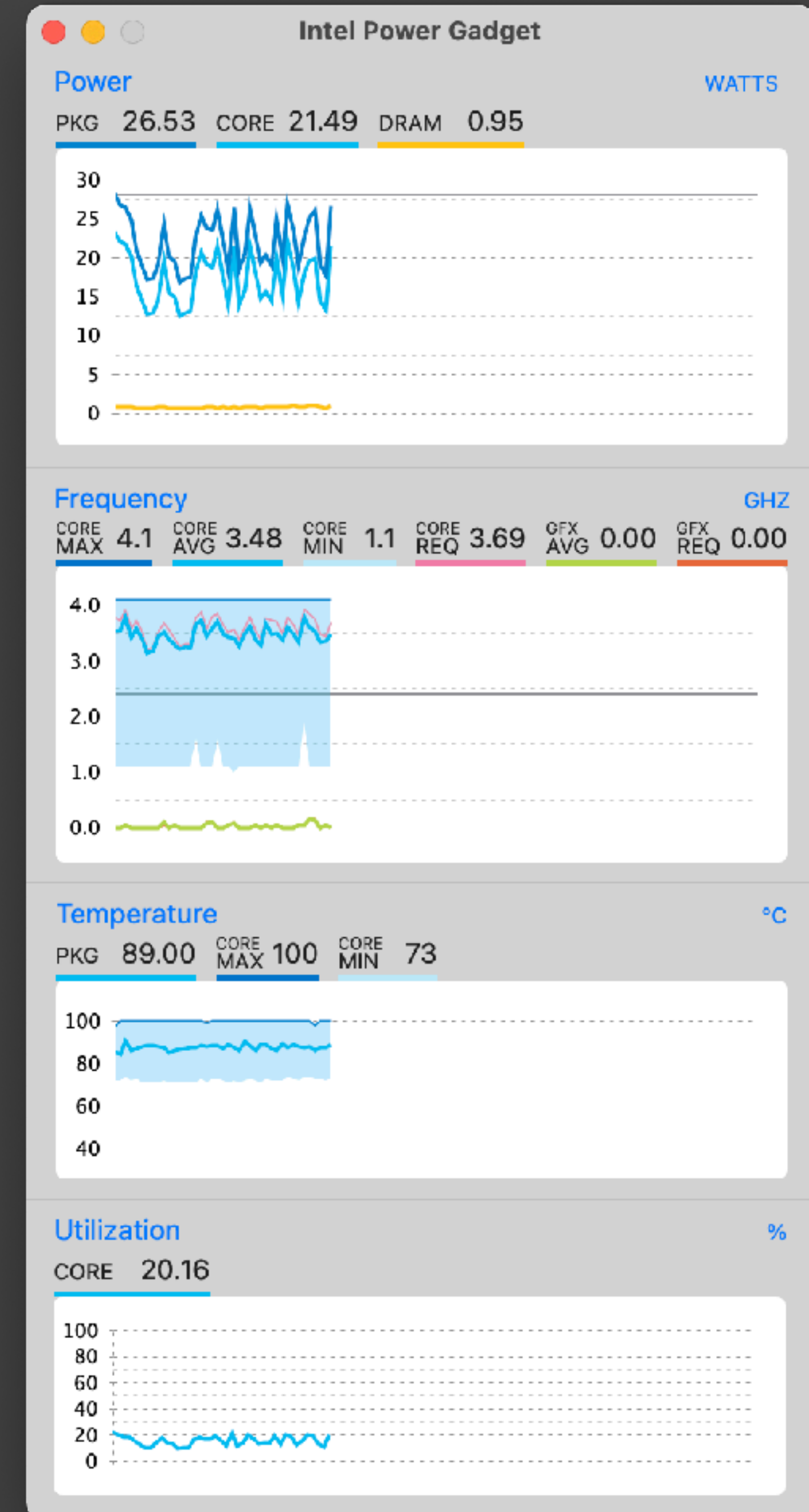
# Intel Power Monitor

- **Install:** <https://software.intel.com/content/www/us/en/develop/articles/intel-power-gadget.html>

- **To collect:** Logging > Log to File



- It will store a **CSV file** with all the collected power data. (File location is specified in the settings)
- Based on Intel **RAPL**. Works with Intel-based Windows and Macs.
- Alternative-twin for M1-based Macs: **Mx Power Gadget**. <https://www.seense.com/menubarstats/mxpg/>

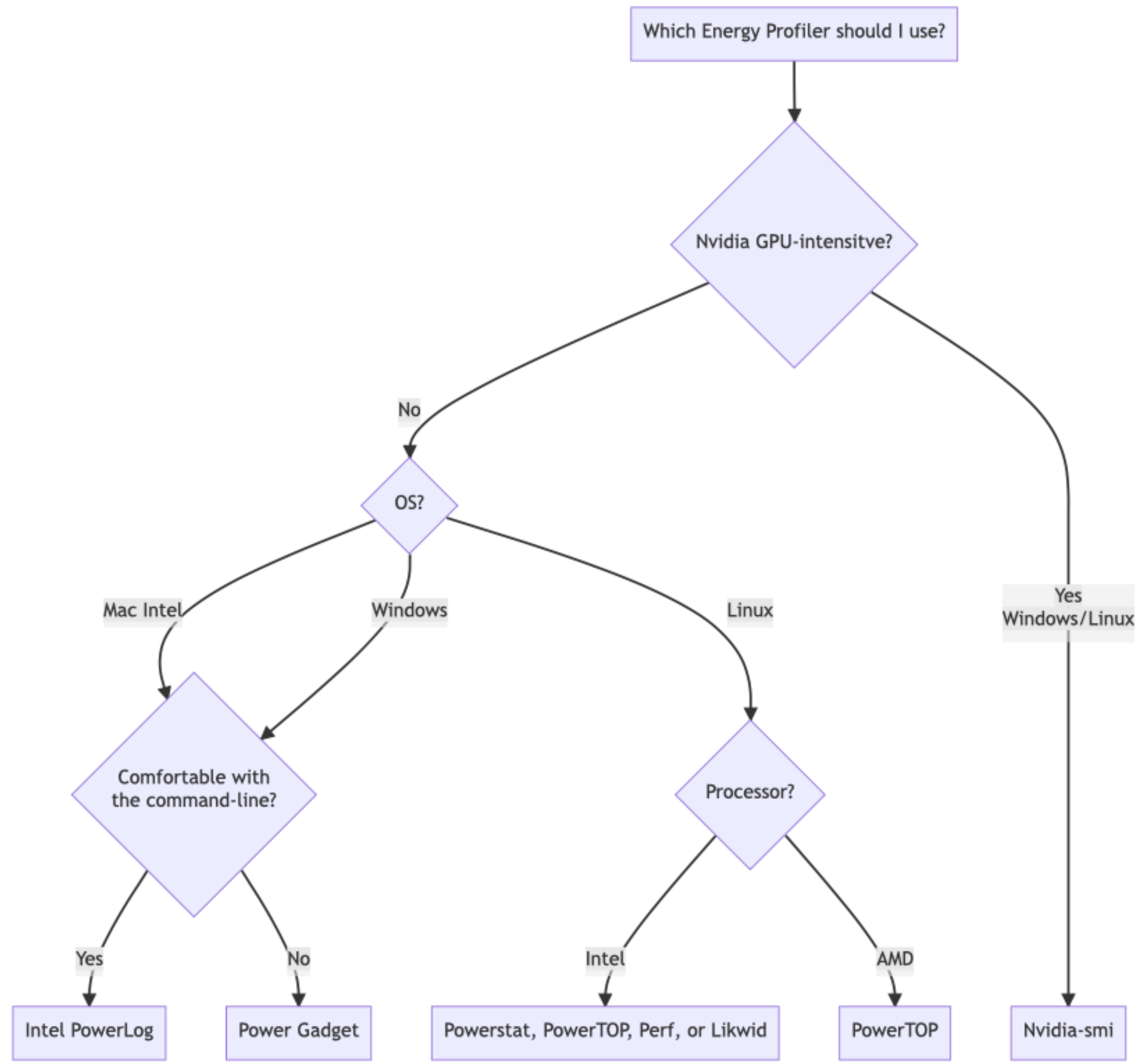




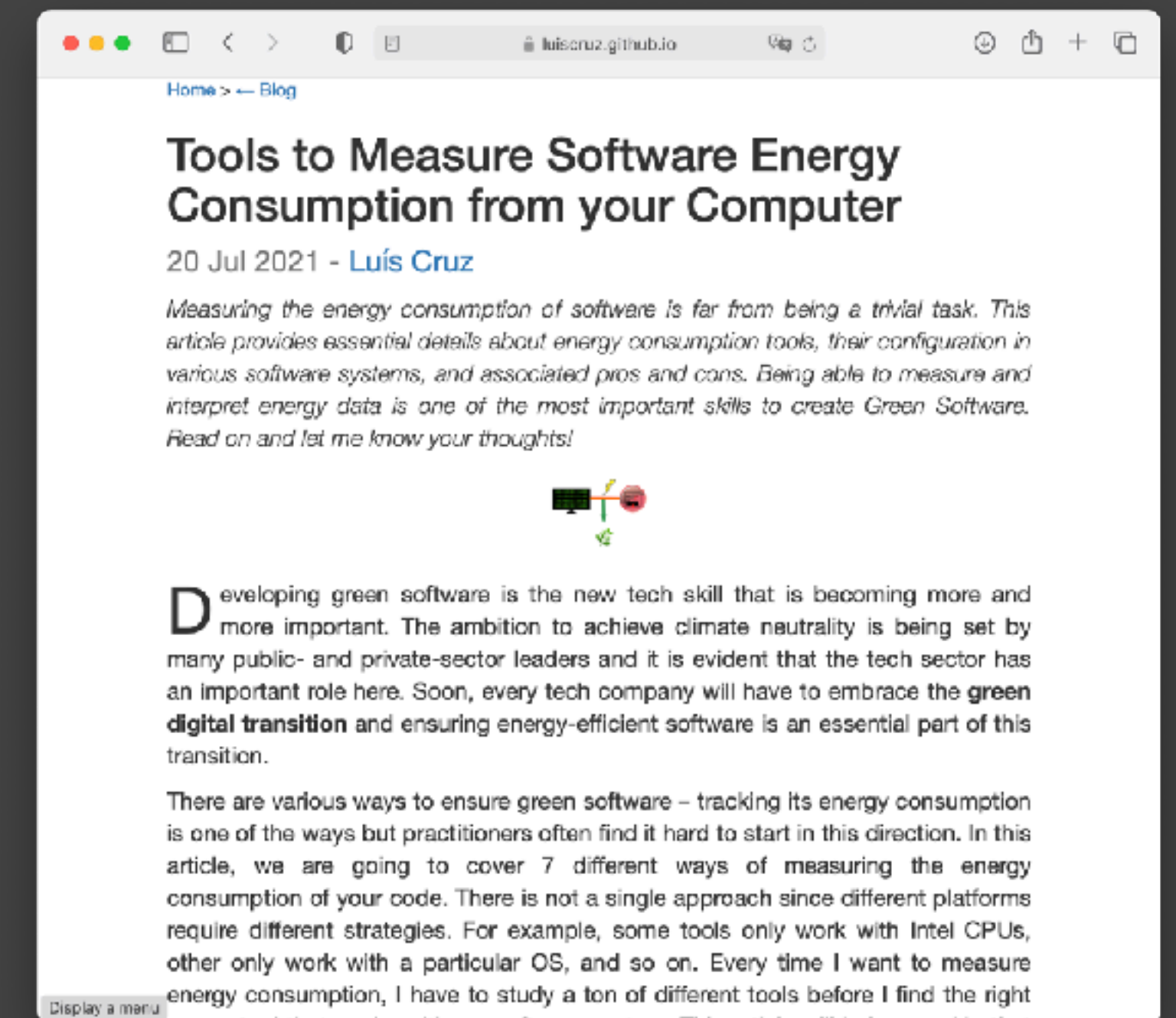
# Intel Power Monitor - Log file

- **Total Elapsed Time (sec)**. The total time in seconds in which power data was being collected.
- **Cumulative Package Energy\_0 (Joules)**. The total energy consumption of the processor.
- **Cumulative DRAM Energy\_0 (Joules)**. The total energy consumption of the volatile memory.
- (Note: IA is only the CPU cores, package is the whole CPU Package; some reports include "GT Energy" – i.e., energy consumption from the GPU).

104	<b>Total Elapsed Time (sec) = 10.030457</b>
105	<b>Measured RDTSC Frequency (GHz) = 2.400</b>
106	
107	<b>Cumulative Package Energy_0 (Joules) = 140.248840</b>
108	<b>Cumulative Package Energy_0 (mWh) = 38.958011</b>
109	<b>Average Package Power_0 (Watt) = 13.982298</b>
110	
111	<b>Cumulative IA Energy_0 (Joules) = 89.629333</b>
112	<b>Cumulative IA Energy_0 (mWh) = 24.897037</b>
113	<b>Average Package IA_0 (Watt) = 8.935718</b>
114	
115	<b>Cumulative DRAM Energy_0 (Joules) = 9.915833</b>
116	<b>Cumulative DRAM Energy_0 (mWh) = 2.754398</b>
117	<b>Average Package DRAM_0 (Watt) = 0.988572</b>

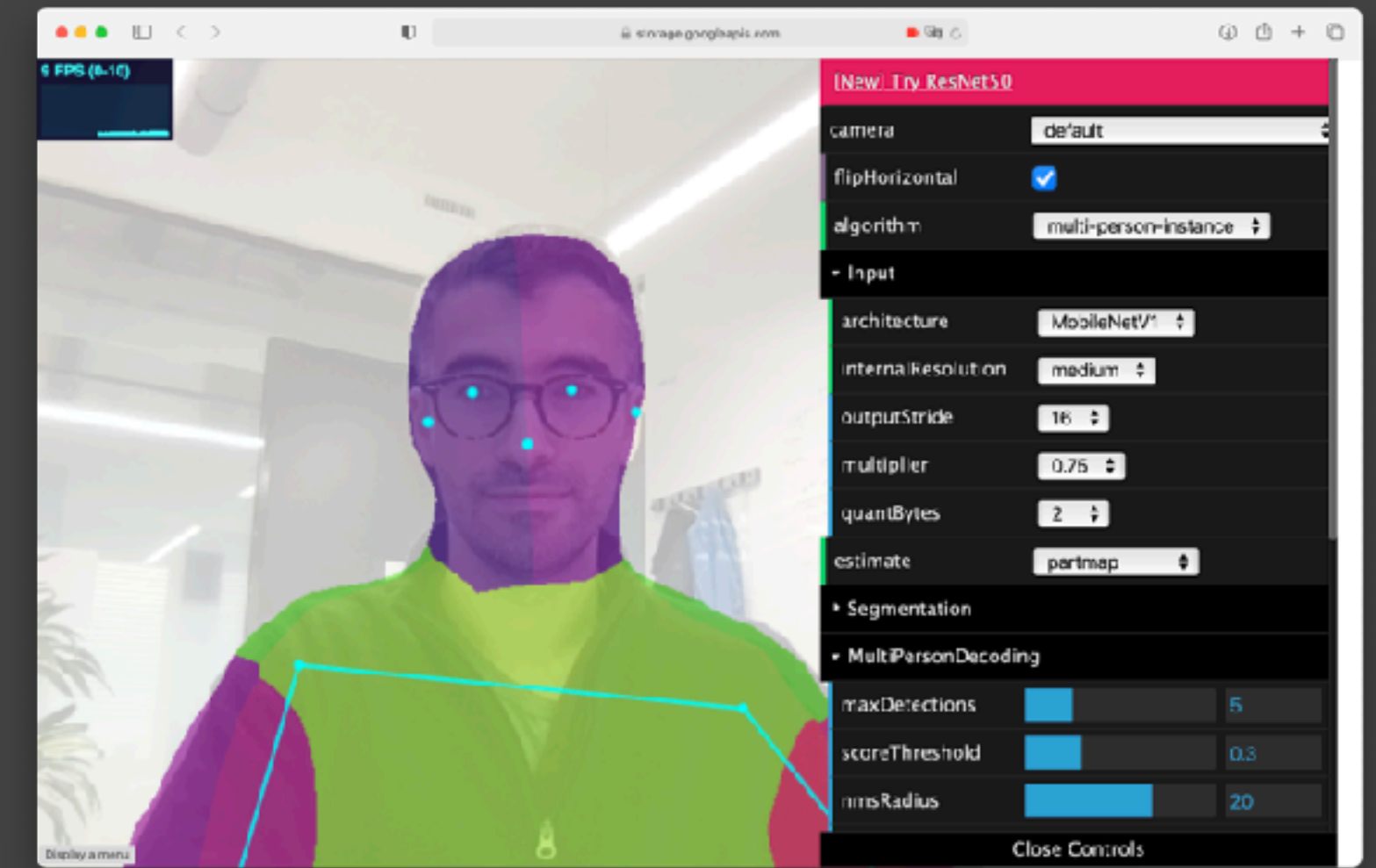


<https://luiscruz.github.io/2021/07/20/measuring-energy.html>



(Missing Apple m1 tools: mxpg, powermetrics)

# Hands-on 1



- **Install** your energy profiler (e.g., Power Gadget).
- **Collect** the energy data of using **Coral BodyPix** for **30 seconds**.  
<https://storage.googleapis.com/tfjs-models/demos/body-pix/index.html>
- **Report** the **total energy consumption**.
- **Extra-mile:**
  - Compare the energy consumption in different browsers.
  - Check the spikes and drops in **Power** and **Temperature**.



# Retrospection

## Hands-on 1

- Are the measurements repeatable?
- What were the confounding factors?
- How can we automate this process?

# Energy testing

(Different from energy monitoring)

1. Create a **reproducible scenario** of the execution of your software. Preferably this should be an automated script – e.g., using a unit test framework.
2. **Execute the scenario** in a version of your software. Use the energy profiler to measure the energy consumption.
3. Improve your software in parts of the code that you suspect have low performance.
4. Execute the same scenario with the **new version**. **Compare the energy data** in this version with the previous one.  
Energy is lower, **test passes**; energy is higher **test fails**.

# Hands-on 2

- Create a reproducible scenario. (Usually easier with command-line interfaces)
- Automatically start/stop energy profiling.

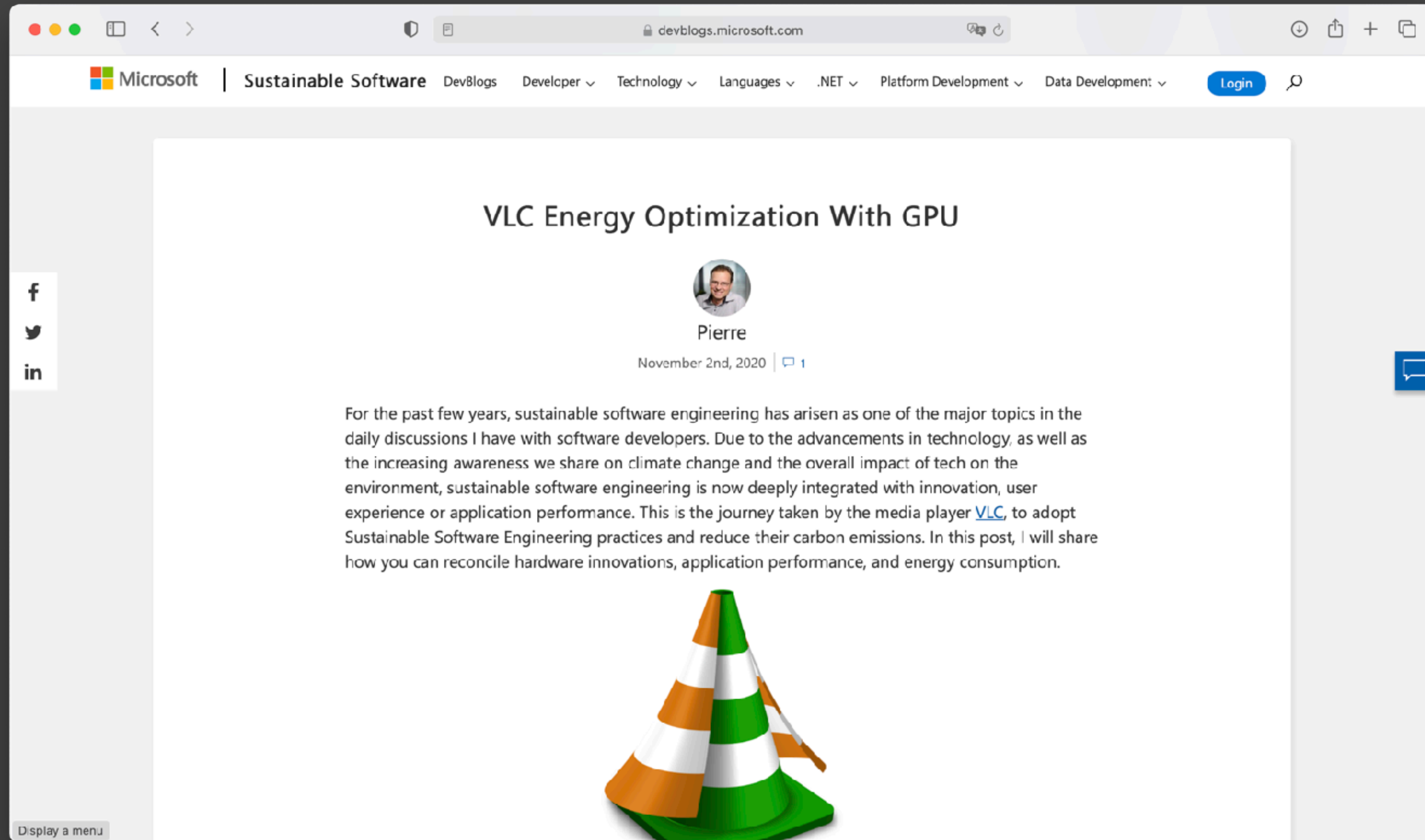


# Project 1

- Deadline: **March 3**
- **Compare energy consumption** in common software use cases.
  - Examples:
    - **Different versions** of the same app;
    - **Same use case** but different apps
    - Same version, same app, but different **user settings** (e.g., enable/disable GPU optimisation)
    - Same version, same app, but different **running environment**
- Submission via **PR** (markdown).
  - Blog-style report (**markdown**, approx **2500 words**).
  - Points if the experiment is **automated** and if there is a **replication package**.

# Pierre Lagarde. **VLC Energy Optimization with GPU**

<https://devblogs.microsoft.com/sustainable-software/vlc-energy-optimization-with-gpu/>



The screenshot shows a web browser window displaying a Microsoft DevBlogs article. The browser's address bar shows the URL <https://devblogs.microsoft.com/sustainable-software/vlc-energy-optimization-with-gpu/>. The page header includes the Microsoft logo and navigation links for Sustainable Software, DevBlogs, Developer, Technology, Languages, .NET, Platform Development, and Data Development. A 'Login' button is visible in the top right. The article title is 'VLC Energy Optimization With GPU' by Pierre Lagarde, published on November 2nd, 2020. The article text discusses the importance of sustainable software engineering and energy optimization in VLC. A VLC logo (a traffic cone) is positioned at the bottom center of the article content. On the left side of the page, there are social media sharing icons for Facebook, Twitter, and LinkedIn. A 'Display a menu' button is located in the bottom left corner of the page.

# Kay Singh. **Apple Silicon M1 Power Consumption Deep Dive Part 1: Safari vs Chrome**

<https://singhkays.com/blog/apple-silicon-m1-video-power-consumption-pt-1/>

Kay Singh

Search Archives About Contact RSS Feed

## Apple Silicon M1 Power Consumption Deep Dive Part 1: Safari vs Chrome

April 2, 2021 · 9 min · Kay Singh

**APPLE M1 POWER CONSUMPTION DEEP DIVE PART 1: SAFARI VS CHROME**

Apple Safari Chrome N YouTube

▼ Table of Contents

- Testing Methodology
  - Software & Hardware
  - Power Consumption Measurement
- Test 1: YouTube streaming