2. Tools to Measure Software Energy (lab)

Sustainable Software Engineering
CS4575

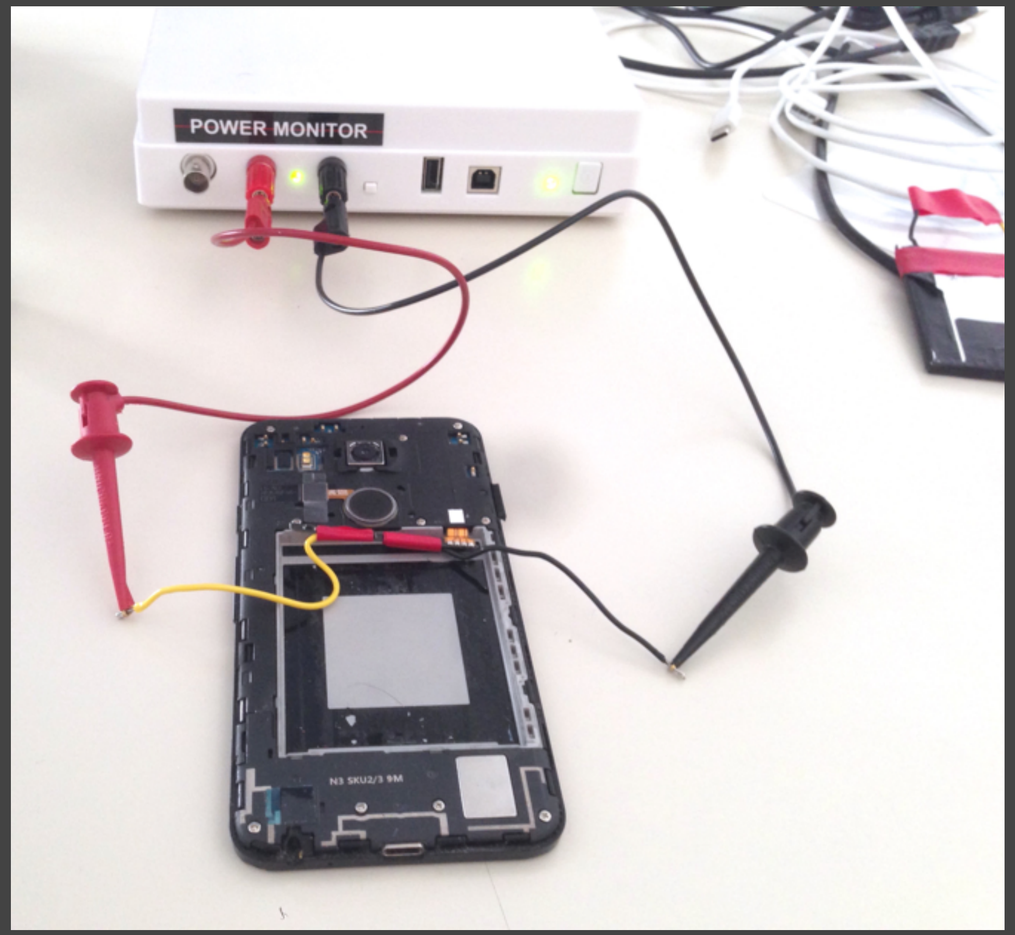
Luís Cruz
L.Cruz@tudelft.nl

Carolin Brandt
C.E.Brandt@tudelft.nl

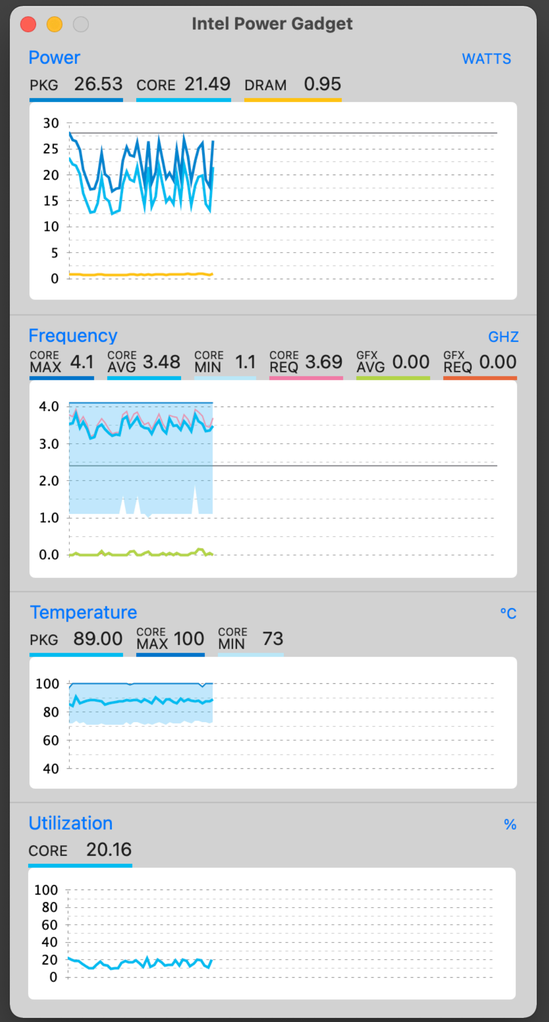
Enrique Barba Roque
E.BarbaRoque@tudelft.nl

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

Hardware Power Monitors



Energy Profilers



Hardware Power Monitors

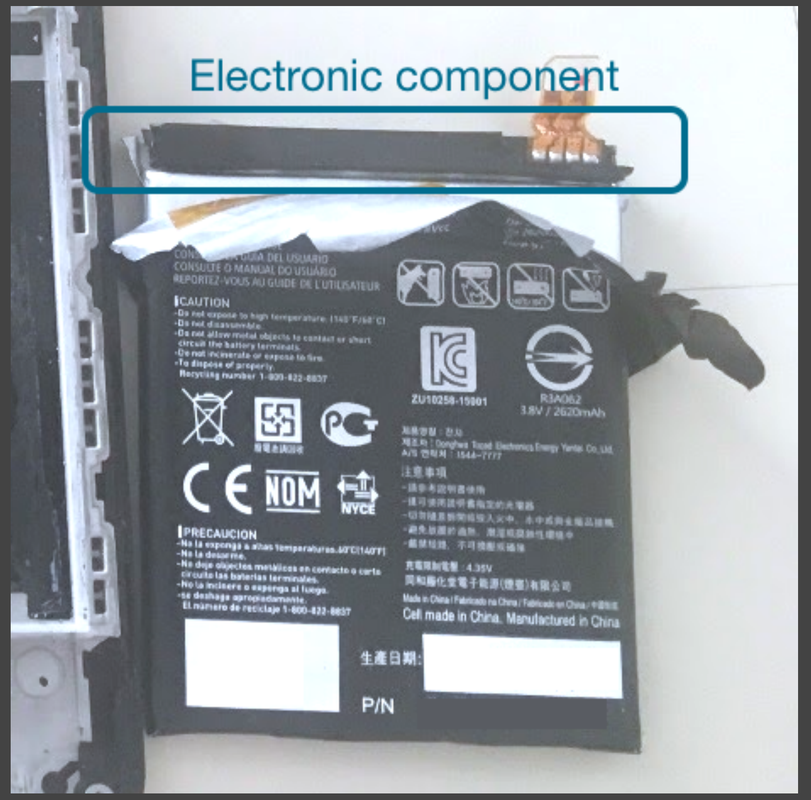
- Connects directly to the power source of the device/ component.
  - Some power monitors also replace the power source.
- Example:
  - **Monsoon Power Monitor** (for IoT and smartphones).
    - Can be fully automated using a Python API.
    - It measures and powers small electronic devices.
- There 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

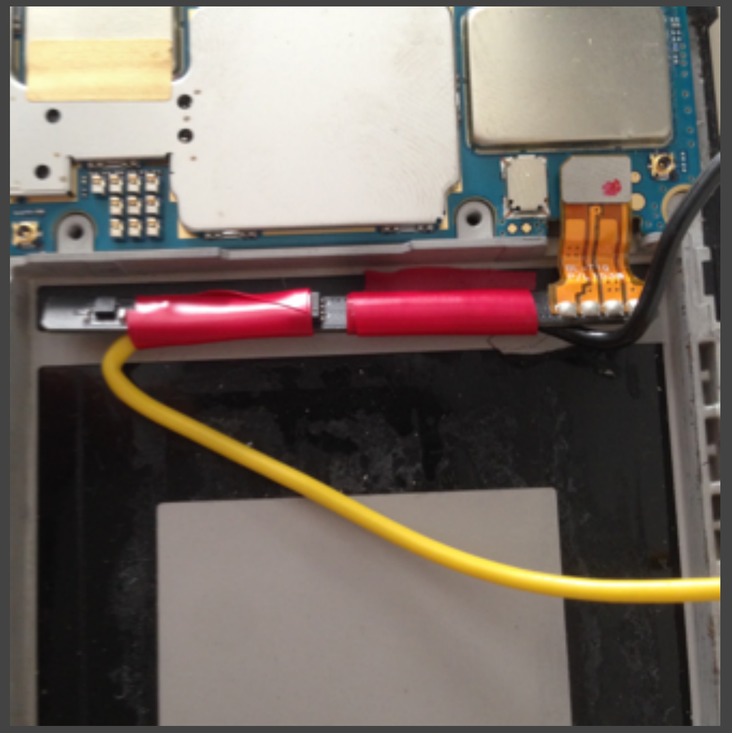
Connecting Monsoon to a Smartphone

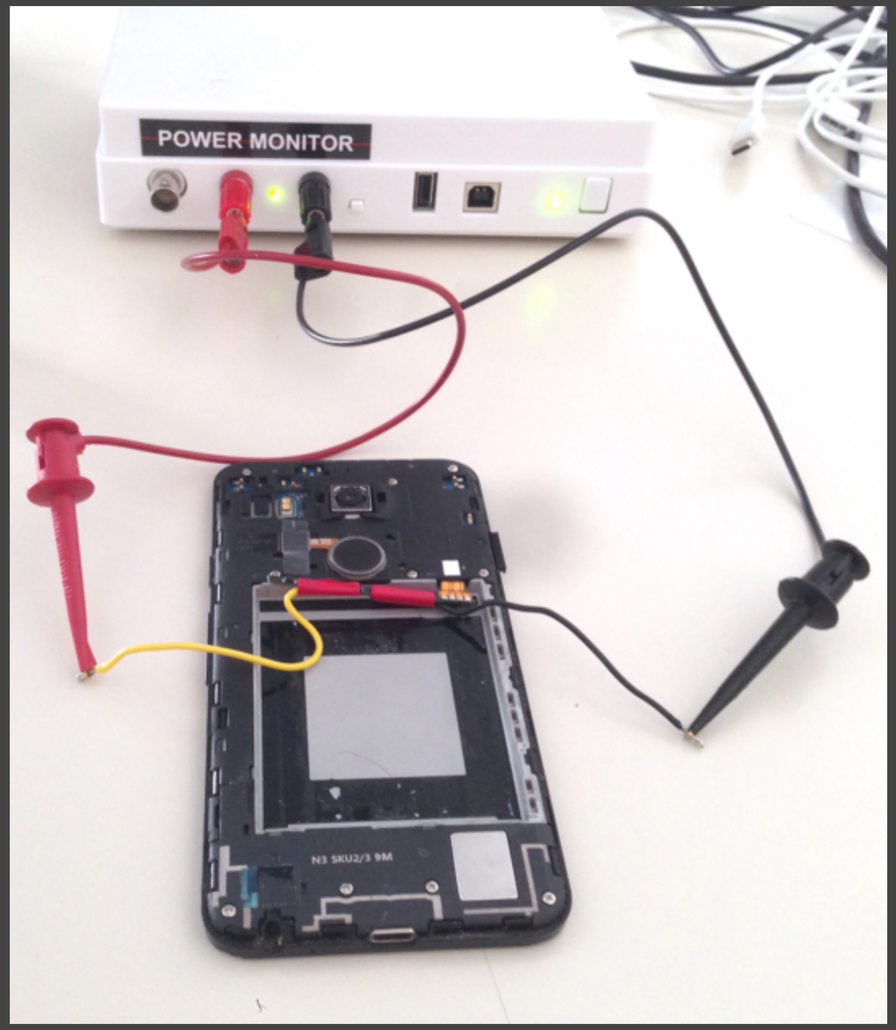
- 2. Extract the electronic component of the battery
- Modern batteries are connected through **4 terminals**:
  - **Positive**
  - **Negative**
  - **BTEMP**, battery temperature (used for safety)
  - **BST**, battery system indicator (provides info about the battery)
  - Hence, one cannot simply connect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



Connecting Monsoon to a Smartphone

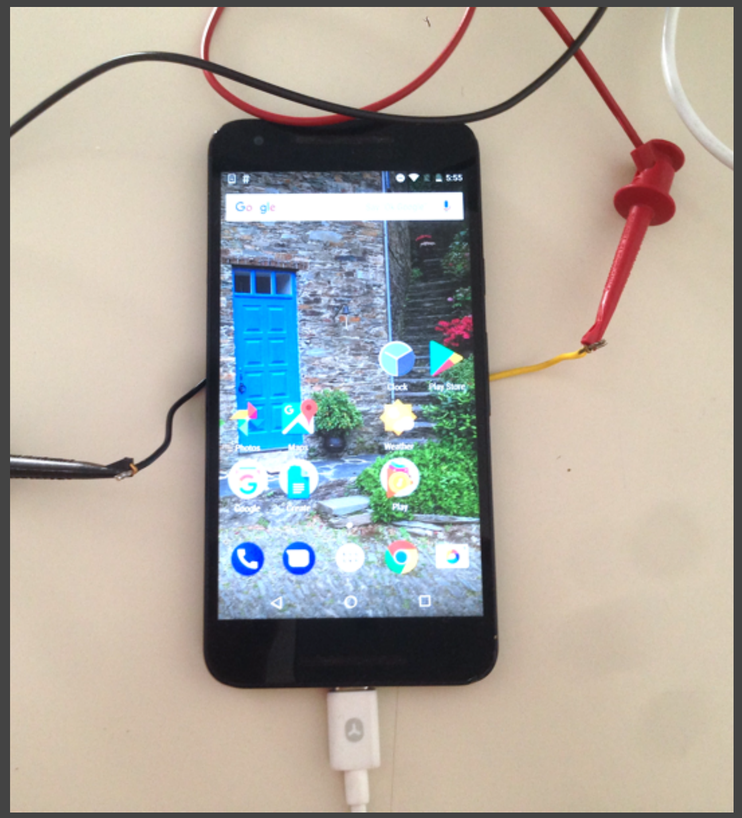
- 3. Connect the electronic component directly to the monitor.





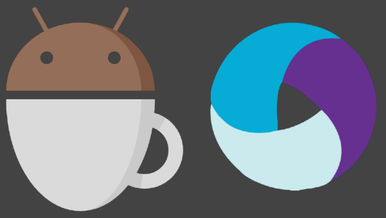
Connecting Monsoon to a Smartphone

- 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

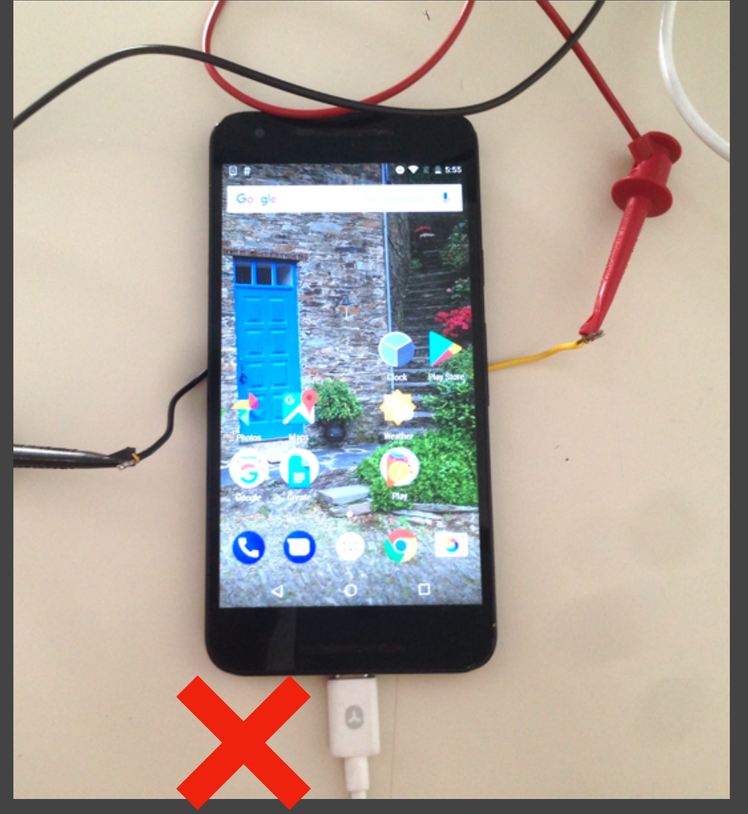


Connecting Monsoon to a Smartphone

- 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** is a library that 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**.

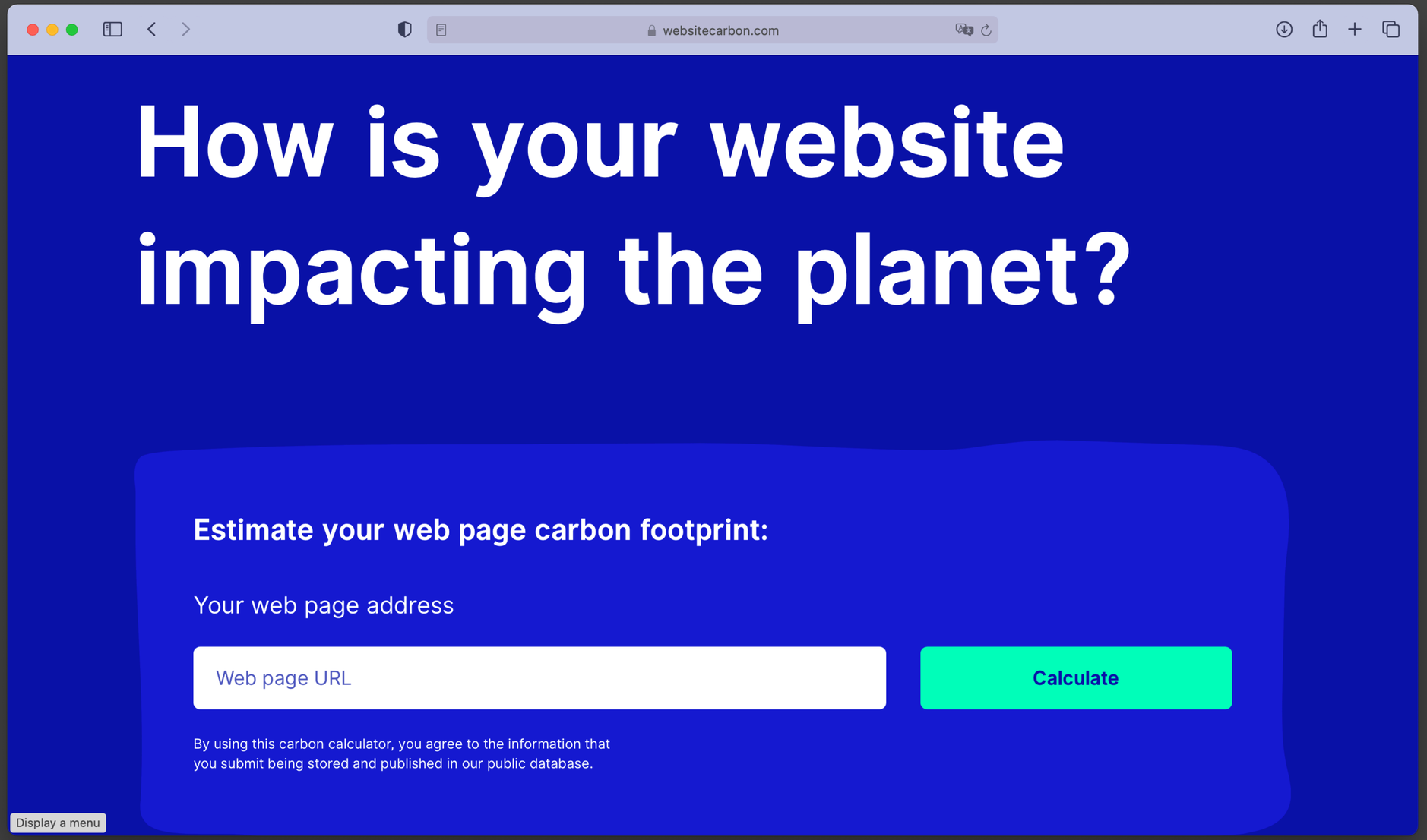
Each case is different – think it through!

Energy Profilers

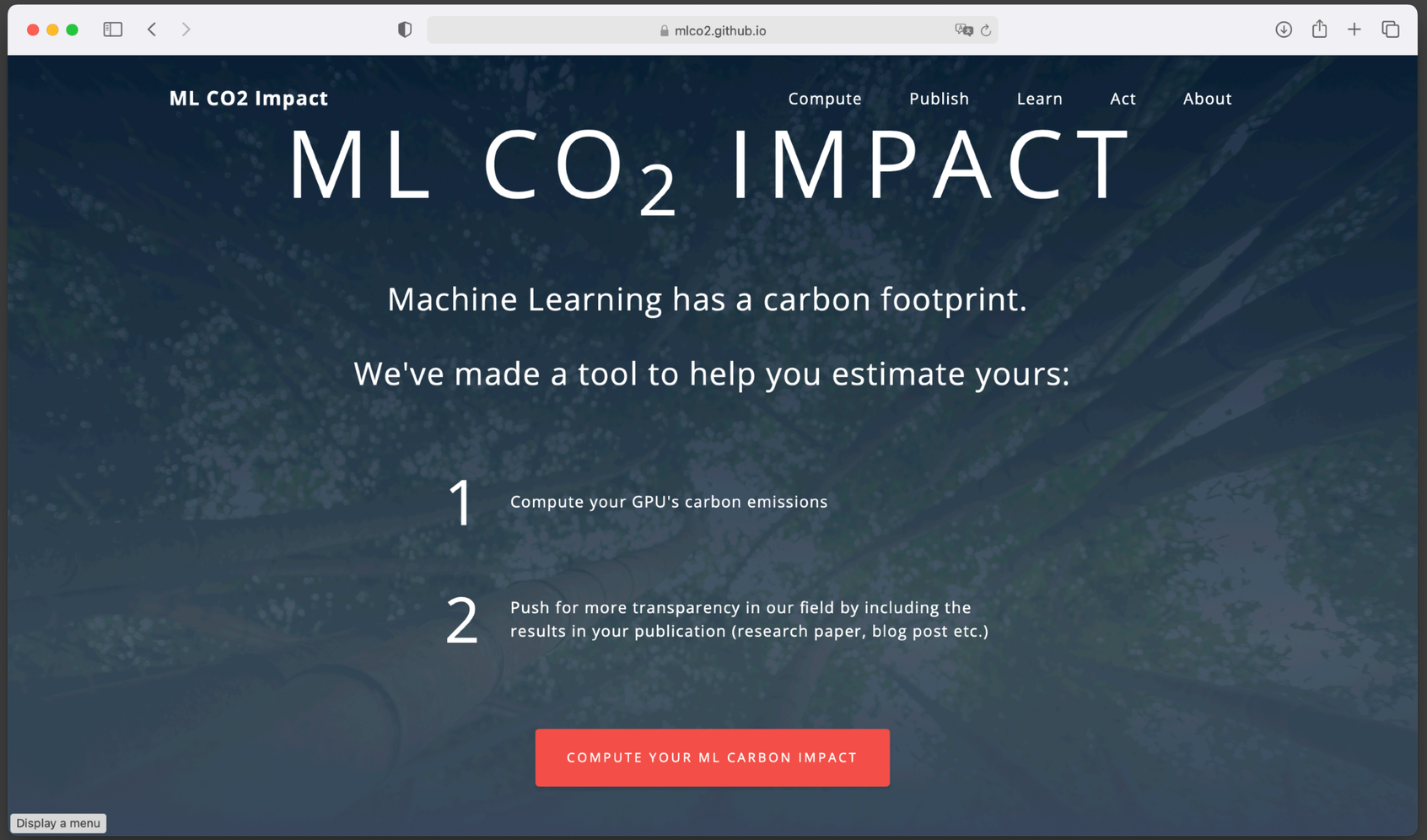
- **Simple setup**! Quite reliable (if you choose the profiler wisely).
  - Recently, they are starting to rely on internal power sensors.
- Still **sensitive to noise** from concurrent processes/tasks! ⚠️

Examples of Energy Profilers

https://www.websitecarbon.com



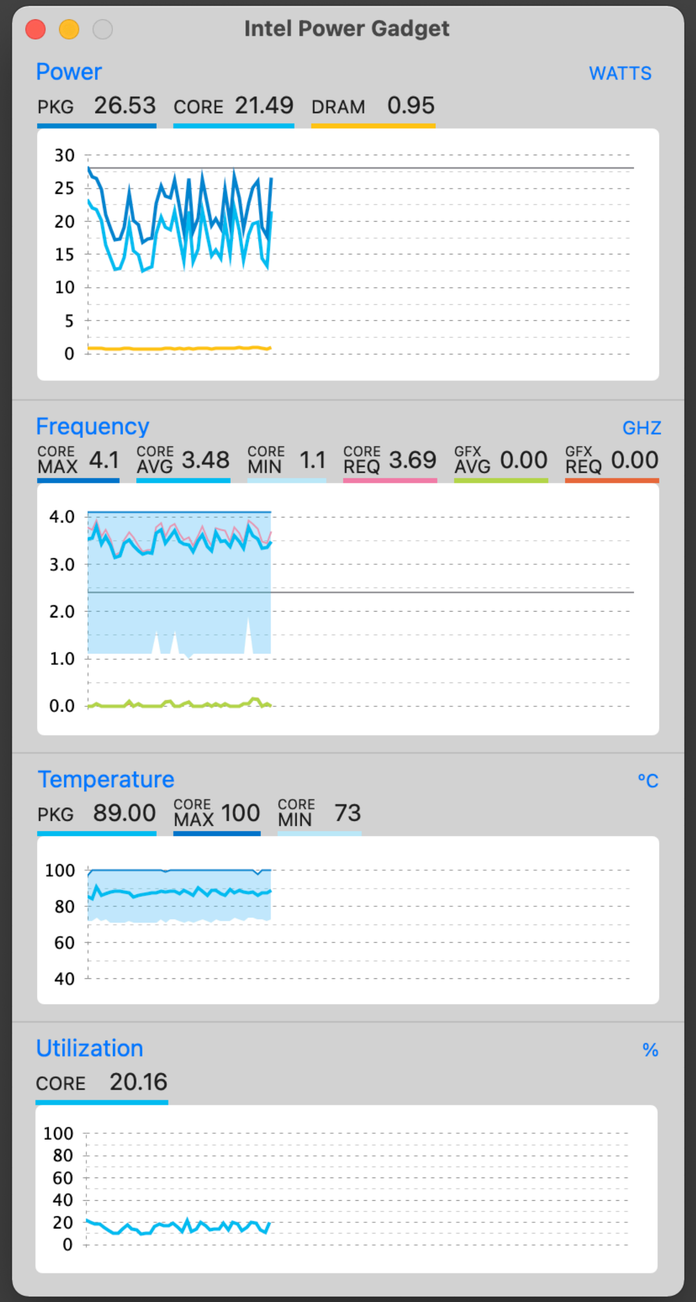
https://mlco2.github.io/impact/



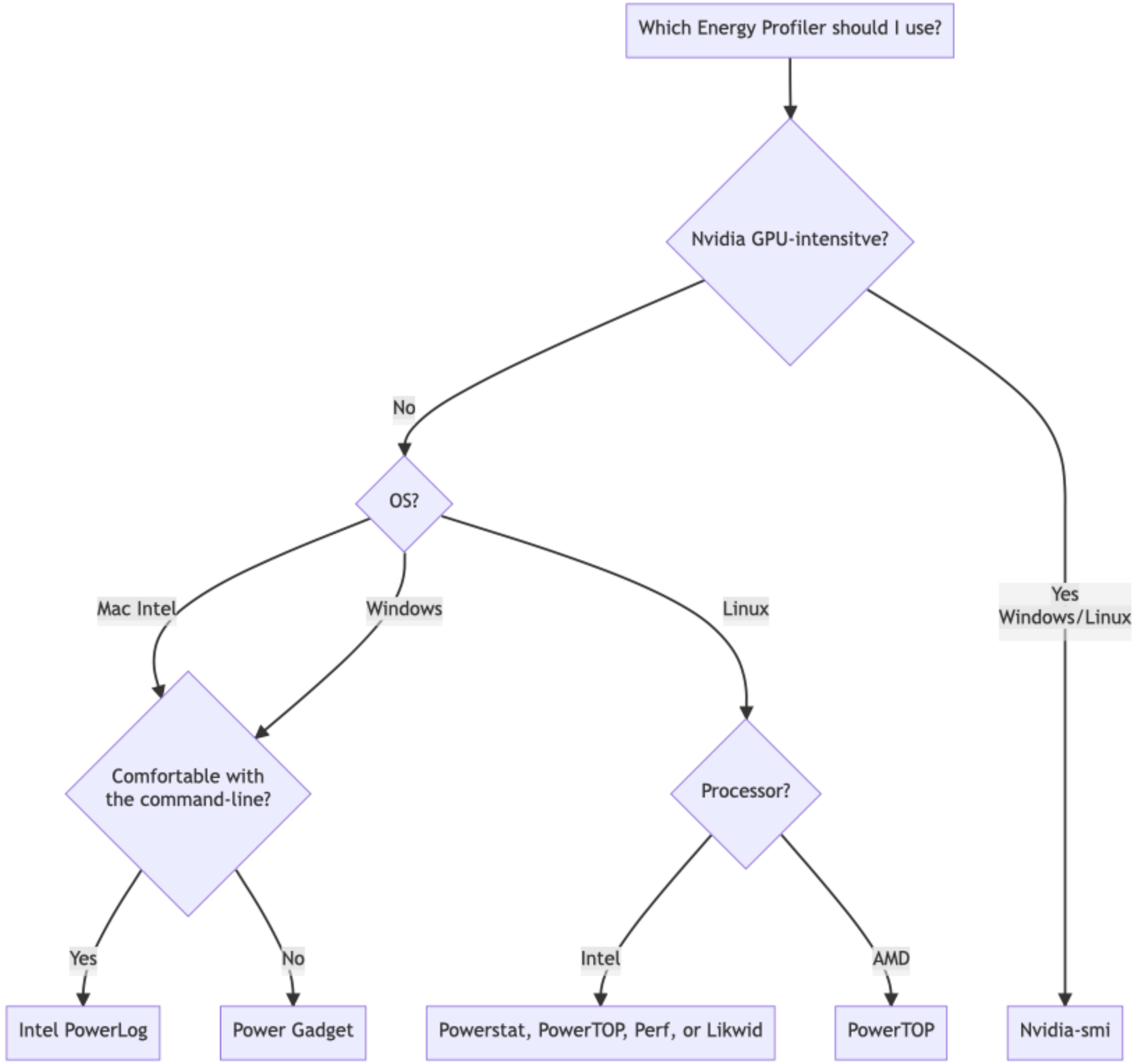
Intel Power Monitor

- **Install**: https://software.intel.com/content/www/us/en/develop/articles/intel-power-gadget.html
- **To collect**: `Logging > Log to File`

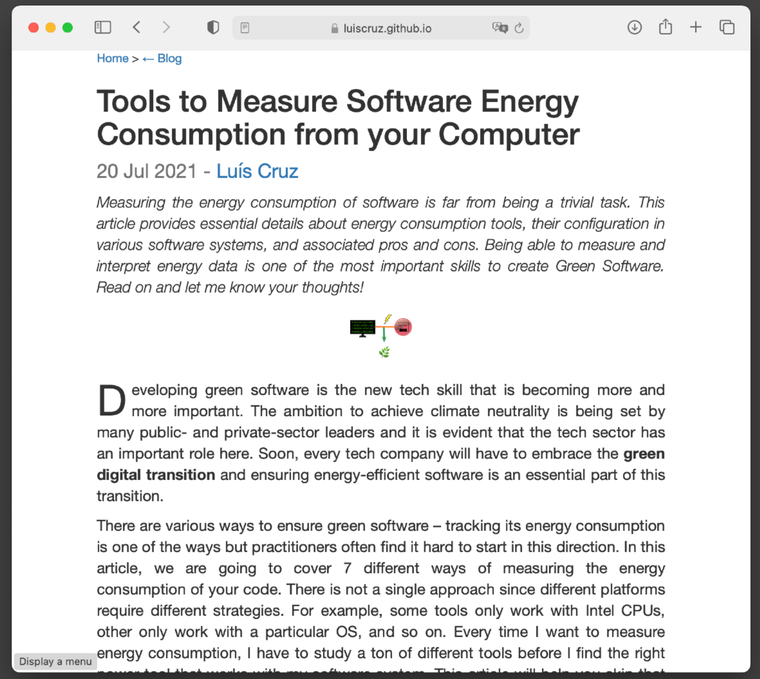
- It will store a **CSV file** with all the collected power data. (File location is specified in the settings)
- Based on Intel **RAPL**. Works with Intel-based Windows and Macs.
- Alternative-twin for M1-based Macs: **Mx Power Gadget**. https://www.seense.com/menubarstats/mxpg/



⚠️ No longer supported by Intel ⚠️



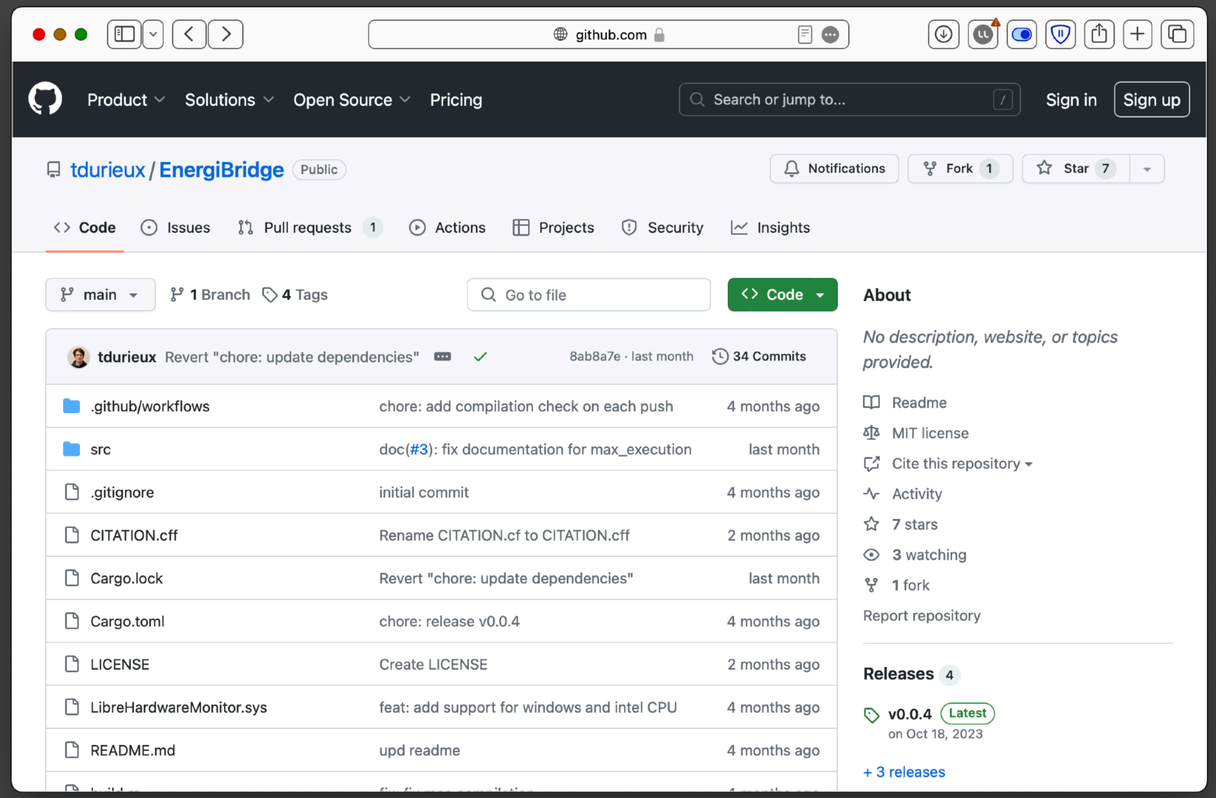
🔗 https://luiscruz.github.io/2021/07/20/measuring-energy.html



(Missing Apple m1 tools: mxpg, powermetrics)

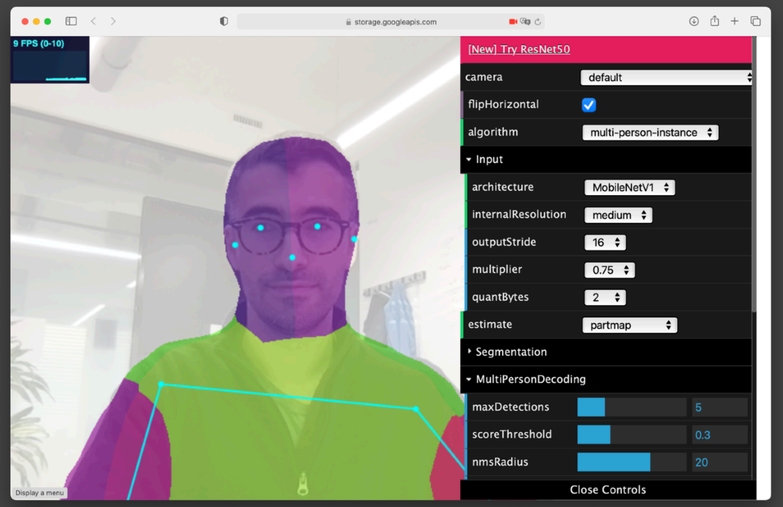
EnergiBridge

https://github.com/tdurieux/energibridge




```
> target/release/energibridge -o results.csv --summary sleep 10
```

Hands-on 1



- **Install** your energy profiler (EnergiBridge).
- **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 bash or python script.

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**.

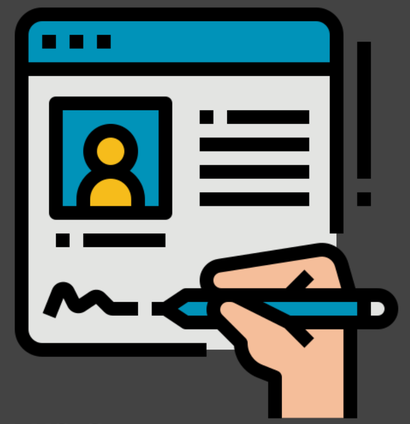
Hands-on 2

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

Project 1

- Deadline: **March 1**
- **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**).
  - Replication package.
  - Points if the experiment is **automated**.

Group registration

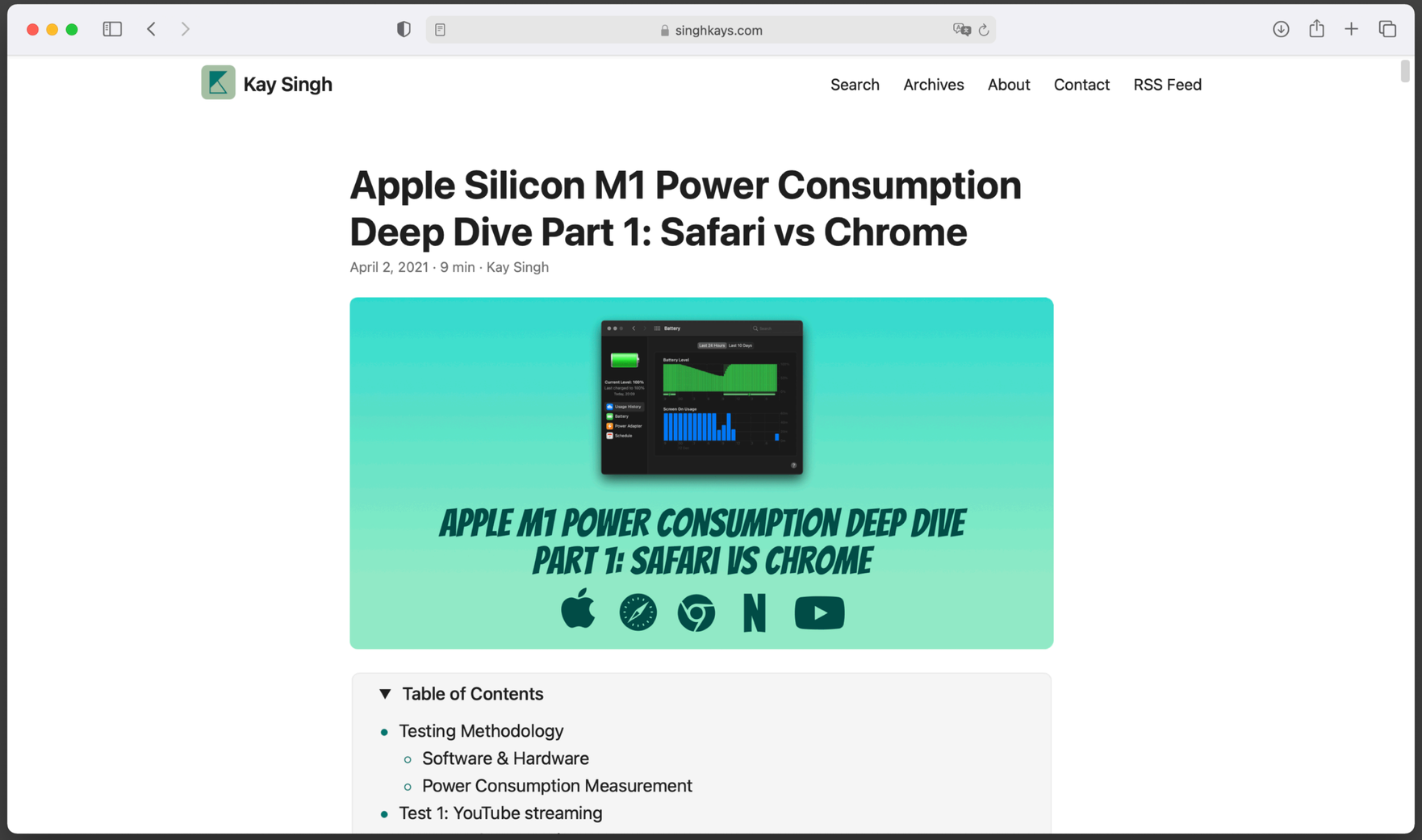


- Brightspace > Collaboration > Groups and sign up for one of the groups under "Project Groups (P1 and P2)"

- If you are looking for a group or teamembers, use the mattermost channel “~Searching-group-members”.
- The deadline for self-registering as a group is end of this week, so by Sunday, 16th of February, 23:59.
  - We might do final adjustments afterwards.

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/



(For project 1, you don’t need to dive deep into hardware details)