Energy Consumption for applications in Jupyter Notebooks

Mitali P, David V, Pia A
Delft University of Technology

ABSTRACT

In an era where sustainability and climate change are at the forefront of the global agenda, the exponential growth of the IT sector presents challenges. As the use of digital solutions grows exponentially, so does the carbon footprint associated with information and communication technologies (ICT). It is our responsibility to practice sustainable ways of developing new digital solutions and helping those around us do so too. This project aims to address the issue of sustainability within software engineering, emphasizing the critical role of energy consumption awareness. Given the widespread use of Jupyter Notebooks across various fields, we introduce an open-source utility - Jupyter Energi that uses EnergiBridge under the hood to get energy metrics for Python programs in Jupyter Notebooks.

1 INTRODUCTION

The second principle of Sustainable Software Engineering [11], emphasizes the importance of building energy-efficient applications, . The first step in this process involves measuring the energy consumed by the application. Once that is measured or estimated, users can begin to analyze how the application can be made more energy efficient. Additionally, measuring the energy consumed enables the users to assess whether their changes are effectively reducing energy consumption. Moreover, the third pillar of the Green Software Foundation strategy [6], aims to develop tools that facilitate sustainability actions, so users can assess and improve the effects of their software. This inspired us to build a utility that makes it easy for users to measure the energy impact of their programs.

Jupyter Notebooks have seen a sharp rise in popularity due to their Python-based environment that operates within a web browser and utilizes a client-server architecture. This architecture enables users to execute code on powerful machines while interacting with the interface from lightweight endpoints, like laptops. Free Jupyter Notebook services like Google Colab make compute-intensive research fields more accessible to a general audience. Python's simplicity, versatility, and extensive library ecosystem combined with Notebooks like Jupyter or Kaggle, offer features such as code execution, visualization, markdown text, and interactive widgets, making them popular platforms for conducting exploratory data analysis, prototyping machine learning models, and sharing research findings. This makes them a preferred choice in fields like data science and machine learning, where high computational power demands are common. For example, some of these popular GitHub repositories use Jupyter Notebooks:

- Probabilistic-Programming-and-Bayesian-Methods, with 26k stars [3]
- generative-ai-for-beginners, with 31k stars [10]
- Python-DataScience-HandBook, with 41k stars [12]

While using these services, it is often easy to overlook the fact that the actual computing requires a significant amount of resources in data centers. Hence, we are focusing on building a tool for Python programs in Jupyter Notebooks with the objective to provide users with insights into their code's energy consumption, to promote more sustainable computing practices. This has some advantages like :

- Developers become aware of the energy consumption of their code. This awareness can lead to more conscious decisions regarding algorithm design, resource utilization, and code optimization, ultimately reducing energy usage.
- Increasing the accessibility of tools. Enabling programmers to incorporate energy efficiency into the development process and promoting sustainable software practices.
- Giving researchers the tools necessary to easily measure the power consumption of code, to investigate sustainable software engineering practices.

To implement our open-source utility: Jupyter_Energi, we choose to use and extend the usability of an existing open-source tool for gathering energy metrics, EnergiBridge, [4]. We chose EnergiBridge primarily because of its familiarity and its support for MacOS and Windows.

2 BACKGROUND

In this section, we provide details for the two crucial parts of our architecture.

Jupyter Notebooks have two components: an interactive computing interface and a back-end kernel. The interface allows the users to write code in different blocks called "cells" which can be executed individually. The kernels are separate computational engines responsible for executing the code contained within the notebook. When a code cell is executed, the code is sent to the kernel associated with the notebook, which executes the code and returns the results. Kernels support multiple programming languages, including Python, R, Julia, and others. Each notebook is associated with a single kernel, but multiple notebooks can share the same kernel.

EnergiBridge is an open-source energy measurement utility, that offers compatibility across various operating systems and hardware configurations. This tool collects data on resource usage and power consumption from the computer, providing metrics for CPUs, GPUs, and memory in CSV files. The approach to collect metrics varies depending on the hardware and operating system in use. For Intel-based systems, EnergiBridge leverages the Running Average Power Limit (RAPL), a low-level interface used in most Intel processors. RAPL enables precise energy consumption measurement with a high sampling rate, particularly tailored for CPU monitoring. GPU metrics, on the other hand, are obtained through the NVIDIA Management Library (NVML), a software development kit (SDK) for GPU management. For MacOS devices, system management control (SMC) is utilized to access energy metrics from both CPU and GPU components. Our project uses the metrics related to system power consumption and time. To use EnergiBridge, clone the utility's open-source repository [4], and install any required dependencies

corresponding to your system's operating system. Users can run EnergiBridge's executable file with tailored command-line arguments, enabling them to adjust parameters such as measurement intervals, execution duration limits, and the generation of GPU data or energy consumption summaries. Figure 1 and 2 show examples of how to use EnergiBridge and the output generated, respectively. Note that columns (CPU_frequency and CPU_usage) for seven out of eight CPUs are hidden in Figure 2 for readability.

```
(base) piaasbjornsen@Pias-MacBook-Air EnergiBridge % sudo target/release/energibridge -o output.csv  --summary echo "Measuring energ
y consumption for this print"

Measuring energy consumption for this print
(base) piaasbjornsen@Pias-MacBook-Air EnergiBridge %
```

Figure 1: Example of CLI execution of EnergiBridge

output

Delta	Time	CPU_FREQUENCY_1	CPU_USAGE_1	SYSTEM_POWER (Watts)	TOTAL_MEMORY	TOTAL_SWAP	USED_MEMORY	USED_SWAP
0	1711633279366	3204	0	8.36043643951416	8589934592	1073741824	6398967808	204734464
4	1711633279366	3204	0	8.36043643951416	8589934592	1073741824	6399148032	204734464

Figure 2: Example output CSV for Figure 1

3 RELATED WORK

There are numerous tools to measure the energy consumption of programs, here we outline a few of the similar extensions we found for Jupyter Notebooks.

Jupyter-resource-usage: One of the popular ones is -jupyter-resource-usage [9], which uses 'psutil' to retrieve data from the kernel infrastructure. This extension displays an indication of how much resources the current notebook and its kernel are using in the form of CPU and memory. However, GPUs which play a significant role in training machine learning models, are not supported [8]. Also, this extension does not explicitly measure or display the energy consumed in Joules or Watts.

Jupyter-energy: Another open-source tool is -Jupyter-energy [5], which measures and displays the energy consumed by the notebook's server. While our work shares similarities with this tool, jupyter-energy lacks the capability for users to isolate specific code segments from the cells in the notebook for energy consumption analysis. Moreover, it also does not provide the user with data plot functionality for further analysis.

With our utility, we seek to support as many operating systems and different processors as possible whilst still giving the user room for customization and flexibility.

4 BUILDING THE UTILITY

Now that we know what Jupyter Notebook and EnergiBridge are, we will see how these two interact. Here's an overview of the architecture, Figure 3.

Jupyter_Energi is a Jupyter extension that allows you to seamlessly integrate the EnergiBridge tool into your Jupyter notebooks. It provides a module that can be imported to measure the energy consumption of a specific Python code block or cell within a Jupyter Notebook. This extension in turn runs an external executable - EnergiBridge as a sub-process via the command line which returns the energy consumption data as a CSV file. There are many energy profilers out there that measure the energy consumption from your

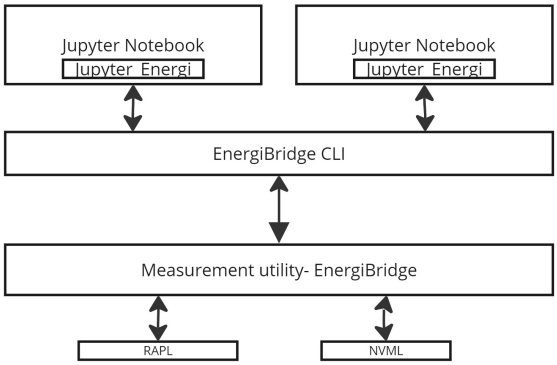

Figure 3: Architecture

computer, as summarised the blogpost *Tools to Measure Software Energy Consumption from your Computer* [2],

- Intel's Power Gadget, has an easy to use GUI and data logging to CSV but is limited to intel CPUs on MAC and Windows.
- Intel PowerLog has CLI but is also limited to Intel CPUs on MAC and Windows.
- Powerstat, also has a CLI but is limited to environments with compatible Intel CPUs.
- Nvidia-smi is an easy to use CLI tool for obtaining power-related information from Nvidia GPUs on Linux, but again, it is limited to Nvidia GPU devices on Linux.

We opted for EnergiBridge due to its capability to log data to CSV and its cross-platform support for Linux, MacOS, and Windows.

OS	Intel CPU	AMD CPU	M1 CPU	Intel GPU	Nvidia GPU	AMD GPU	M1 GPU
Linux	✓	✓			✓		
Windows	✓	✓			✓		
Mac	✓		✓	✓		✓	✓

Figure 4: Platform support offered by EnergiBridge

This multiplatform support offered by EnergiBridge presents a significant advantage, serving as a comprehensive solution for most users. This eliminates the need to install separate tools tailored to specific operating systems and processors, especially beneficial for shared notebooks like Jupyter, which may run on various end-user systems. Given the limitations of the systems at our disposal, we were able to integrate commands for the Command Line Interface (CLI) into our extension for Windows and MacOS CPUs. However, given the capability of EnergiBridge, users can easily adopt and customize our extension code to retrieve energy usage data by adding commands for Linux or modifying any existing command to get GPU usage data.

5 THE UTILITY: JUPYTER_ENERGI

Now let us dive into how Jupyter_energi works along with the requirements to use it. Our utility allows users to measure energy consumption from the cells in Jupyter Notebook in two ways, shown in Figure 5:

- By using the (#EnergiBridgeStart and #EnergiBridgeStop) markers with the code contained between the tags.
- By defining the code as a string and directly passing it to the run function.

```
import jupyter_energi

# Method 1
#EnergiBridgeStart
for i in range(1000000):
    pass
#EnergiBridgeStop

jupyter_energi.run()

# Method 2
code = """
for i in range(1000000):
    pass
"""
jupyter_energi.run(program=code)
```

Figure 5: Example snippet of Jupyter_energi

Specifying the code in the above-mentioned fashion allows us to parse the Python program within the notebook using 'nbformat', a Python library that provides a way to work with Jupyter Notebook file formats programmatically. We therefore extract code segments, the users are targeting to evaluate energy consumption. As of now, there are no better-known ways of extracting specific cell code from these notebooks, necessitating the parsing of the entire notebook.

The jupyter_energi.run(program, no_runs) function is called to run experiments. The program parameter can be used to send a Python program as a string to Jupyter Energi, if left empty Jupyter Energi will look for the delimiting comments in the notebook. no_runs specifies the amount of runs the experiment will be performed, it is by default 1. Specifying multiple runs is an easy way to collect more data, and average out any inconsistencies.

As the energy data is stored in a Pandas data frame, it allows the user to analyze and visualize it extensively according to their preferences. To plot the energy consumption pattern over time we provide a function to obtain a power(Watts) vs time(s) line plot. Moreover, Jupyter Energi offers a violin plot generation function, meaning multiple runs can easily be analysed for statistical inconsistencies. The violin plot generation function also comes with the standard deviation parameter, std_dev, that specifies removed outliers that are more than std_dev standard deviations away from the mean.

Since we were working with MacOS and Windows machines, our tool provides support for both, but depending on the hardware you might require different dependencies for Energibridge setup. figMeasuring the energy cost between code changes will allow the users to observe whether their changes are improving the energy cost over time. The plots are especially useful in this scenario.

6 VALIDATION

Validation on simple code

To properly validate the functionality of the code, a small program was written that calls sleep every second for three seconds, Figure 6.

```
import jupyter_energi

#EnergiBridgeStart
from time import sleep

for i in range(3):
    sleep(1)
#EnergiBridgeStop

data = jupyter_energi.run()
cumulative = False
time_and_power = jupyter_energi.extract_time_and_power(data, cumulative)
jupyter_energi.make_time_series_plot(time_and_power, cumulative)
Executed at 2024.03.27 16:02:28 in 10s 46ms
```

Figure 6: Simple validation code

The snippet demonstrates the ease of use of Jupyter Energi. We used the #EnergiBridgeStart and #EnergiBridgeStop delimiters to let Jupyter Energi know what code to run. After running we call the extract_time_and_power function followed by the make_time_series_plot with cumulative set to False.

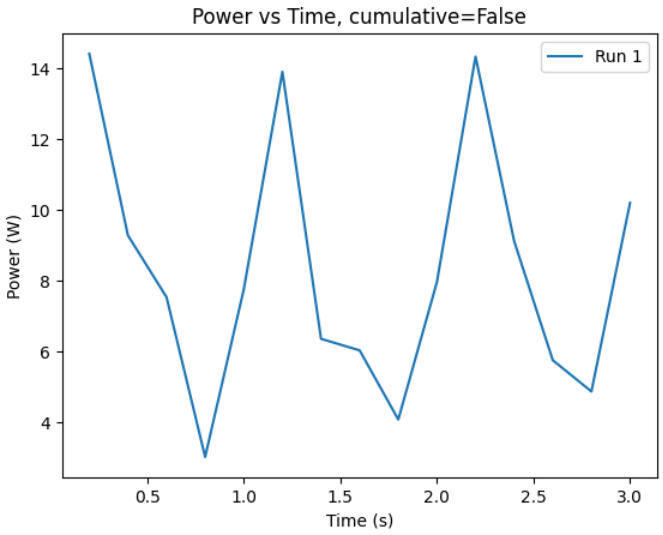

Figure 7: Time series plot for simple validation code

As can be seen in the resulting graph, Figure 7, the sleep commands are noticeable, leading to clear peaks when the program comes out of sleep. To properly validate the same program was rerun with the no_runs parameter set to 10.

As seen in Figure 8 the runs appear to show the same pattern, albeit with one significant outlier. This is to be expected as running EnergiBridge out of Jupyter Notebook leads to an environment with more background processes, and thus less consistent results.

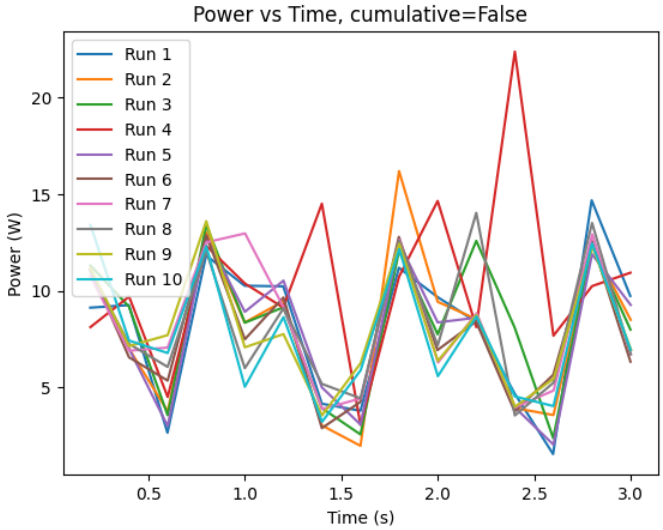

Figure 8: Time series plot for simple validation code, 10 runs

To properly inspect this outlier we rerun the time series function with cumulative set to True and are left with Figure 9.

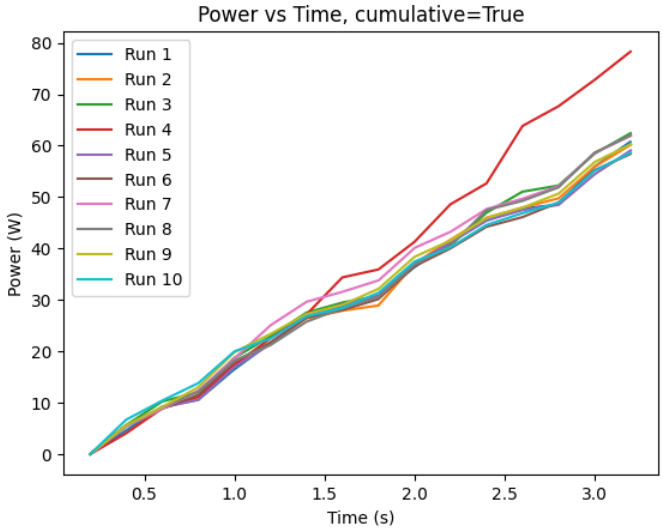

Figure 9: Time series plot for simple validation code, cumulative set to True

In this visualization, we can clearly see the outlier and are even able to see at what point in time it started diverging.

For further analysis, the violin plot found in Jupyter Energi can also be used. The violin plot calculates the total energy used per experiment and uses this to compute the plot. Therefore the significance of the violin plot grows with the amount of runs executed. To this end, a new dataset was collected where the no_runs parameter was set to 30. The time and power were then extracted with Jupyter Energi and passed to the make_violin_plot function. In Figure 10 we see no considerable outliers, and an average total energy consumption of around 70 Joule, which can also be seen in Figure 9.

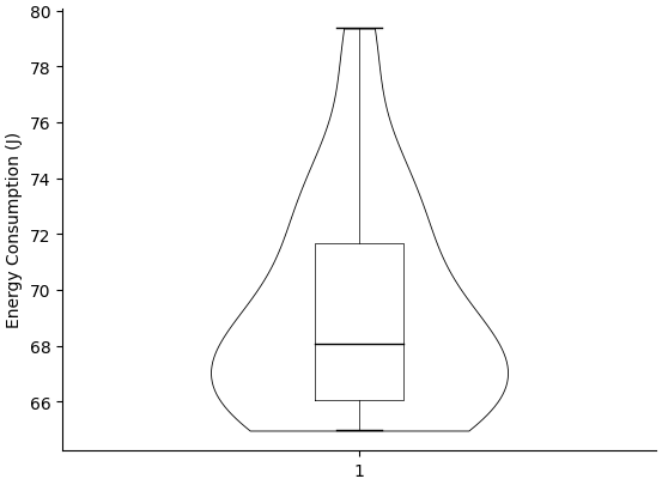

Figure 10: Violin plot for simple code, no_runs = 30

Validation on complex code

To test Jupyter Energi in an environment closer to one where it may be used, we decided to test it on a complex notebook with high computing costs. For this reason, we chose one of the tutorials from QuTip to test Jupyter Energi. QuTip is an open-source computational physics software library for simulating quantum systems, particularly open quantum systems.

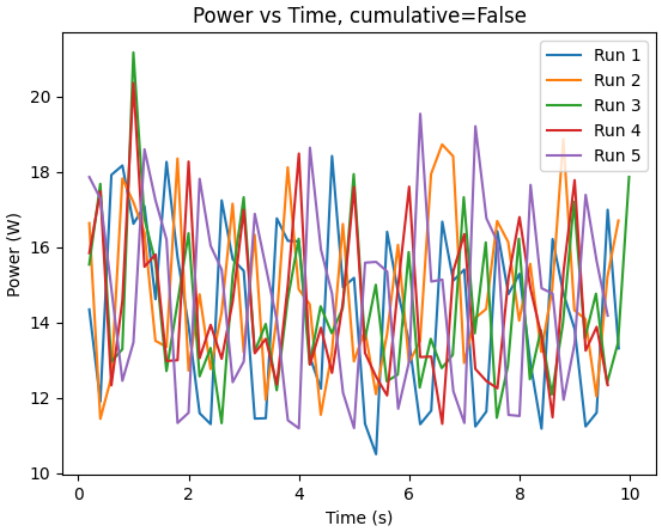

Figure 11: Time series plot for complex validation code, 5 runs

The tutorial in question is "HEOM 4: Dynamical decoupling of a non-Markovian environment", [7]. After running Jupyter Energi on the extracted code with `no_runs` set to 5, we extract Figure 11 with the time series function.

As can be seen, the figure looks very random and doesn't seem to give a lot of information. Nevertheless, plotting with cumulative set to True gives us a clearer idea of the validity of Jupyter Energi. We observe in Figure 12 that all runs follow the same line. Even though there appears to be significant variation in the individual time plots,

the runs show almost the same behaviour when accumulating the power.

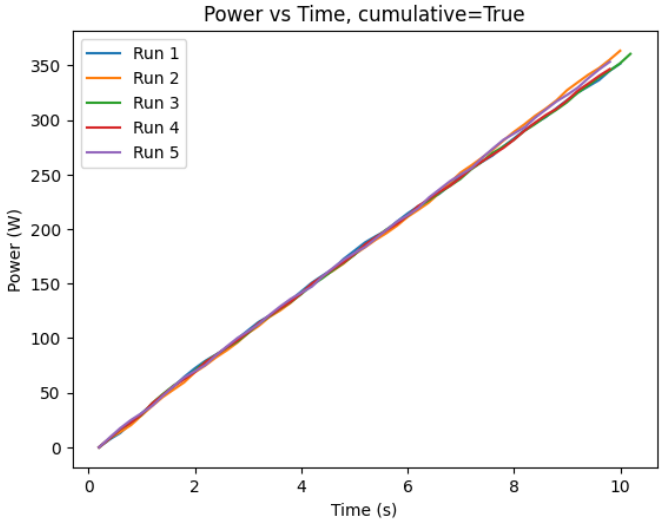

Figure 12: Time series cumulative plot for complex validation code, 5 runs

Violin plots were also created for the complex code, once again with no_runs set to 30, we observe that there are 2 considerable outliers present in Figure 13

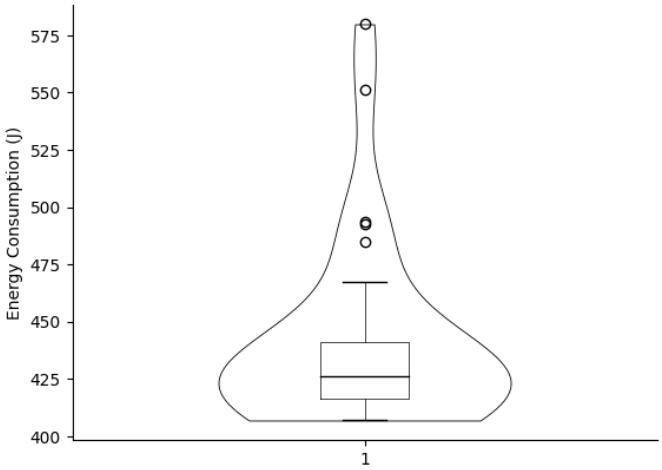

Figure 13: Violin plot for complex code, no_runs set to 30

Researchers may want to inspect their data without the outliers, and for this purpose, Jupyter Energi offers a std_dev parameter. This parameter specifies the outliers that should be removed, which are the ones differing more than n standard deviations. Re-running the function with std_deviation set to 1 produces Figure 14 as a result. Thus, after running Jupyter Energi on both simple code and complex code, we can conclude that Jupyter Energi is not only validated but also extremely useful. Offering users a simple way to measure energy consumption directly out of Jupyter Notebook and also plotting the results.

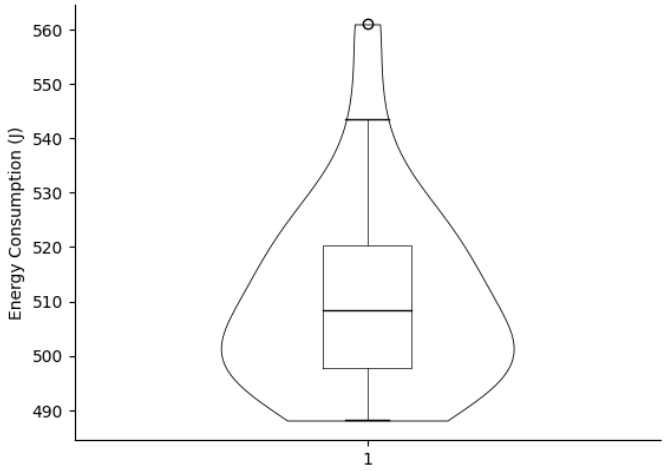

Figure 14: Violin plot for complex code, std_dev param set to 1

7 LIMITATIONS AND FUTURE WORK

While Jupyter Energi presents a valuable tool for measuring energy consumption in Jupyter Notebooks, it also has limitations mentioned below:

Dependency on EnergiBridge: Jupyter Energi heavily relies on EnergiBridge for energy consumption measurements. As a result, the accuracy and reliability of the measurements are directly impacted by the performance of EnergiBridge. Any limitations or inaccuracies in EnergiBridge's measurements will affect Jupyter Energi's effectiveness.

Influence of background activities : Other activities running on the system affect the accuracy of energy consumption measurements. These processes can potentially consume CPU resources and memory, leading to variations in power usage over time. Users should be aware that Jupyter Energi does not exclusively measure the energy usage of the notebook's program. It incorporates all background processes into its measurements also, due to its utilization of EnergiBridge. In environments with heavy background activity, this may result in less precise measurements for the notebook, potentially affecting the energy consumption analysis.

Note that our evaluation was conducted in a scenario where all the background processes were not removed. It was rather a realistic portrayal of many tasks running simultaneously. As such, no strategies to minimize the impact of various biases during the collection of energy data were considered. However, if users want to minimize the bias factors affecting the energy measurements of their Jupyter Notebook program, we recommend running our tool while minimizing other computing tasks on the system. Some strategies can include closing all unnecessary applications and services in the background, turning off notifications, and connecting to only required hardware. This can help minimize bias factors in energy measurement and can lead to closer to accurate and reliable results for power consumption. For more details, we refer to *Green Software Engineering Done Right: a Scientific Guide to Set Up Energy Efficiency Experiments* [2].

There are also a lot of areas this extended utility can be improved:

Enhanced functionality: Future versions of Jupyter Energi could provide users with additional details on how energy consumption is measured. Offering more detailed insights into the measurement process can help users better understand the utility's capabilities and limitations. Moreover, expanding the utility to incorporate more functionality for other aspects of EnergiBridge, such as retrieving data from GPUs or Linux support, could be a natural next step for future work. Offering more functions for visualization could also provide users with enhanced insights into their code's energy consumption patterns. For instance, the usage of scatter plots can show correlations between energy consumption and variables like CPU or memory usage.

Simplify installation and setup: Streamlining the installation process can enhance user experience and adoption of Jupyter Energi. Extending it into a standalone package for publication on Python package managers like pip can simplify installation for users. A seamless installation experience can encourage more developers to integrate energy consumption analysis into their workflow.

User studies: An under-explored aspect is the actual impact of Jupyter Energi with users. Conducting user studies can provide valuable feedback on the utility's effectiveness and usability. By seeking feedback from users, we can gain insights into how Jupyter Energi influences energy-conscious decision-making in software development. User studies can inform iterative improvements and feature enhancements based on real-world usage scenarios.

We encourage contributions to the project, and invite you to take a look at the Jupyter_energi repository on GitHub (*https://github.com/mitalipatil99/Jupyter_Energi*) [1], and provide any feedback or suggestions for improvement.

8 REFLECTIONS

With the development of Jupyter Energi, we hope to empower users with a tool that highlights the energy impact of their software on the environment. In the development and validation of our utility, several significant insights emerged, which bear discussion both in terms of technical implementation and broader implications for the software development community.

Unlike measuring CPU or RAM usage, measuring energy consumption lacks established standards. It requires consulting different libraries and services depending on the hardware being used. Measuring energy consumption for modern systems is far from easy and predictable. Operating systems with power-saving strategies can complicate obtaining reliable results. This unpredictability also complicates attributing energy consumption to individual processes. While it's possible to track the CPU cycles or RAM usage of a process, energy usage is influenced by various factors that make it harder to track. For instance, the activation of a fan, which consumes energy, often results from the interaction of multiple processes running concurrently rather than the action of a single process.

Having said that, there remains significant merit in enhancing current tools that estimate energy usage on systems. Introducing a billing model tied to the energy consumption of the software could more accurately represent associated costs, thereby incentivizing users to monitor the energy footprint of their code. This not only promotes environmental consciousness but also fosters a culture of cost-awareness within the development community. Furthermore, open-source initiatives could play a pivotal role in users being able to access such tools and use them with ease with the help of community support and documentation. We look forward to a future where there is wider adoption and active participation in building a sustainable software ecosystem.

REFERENCES

[1] Pia Asbjørnsen, Mitali Patil, and David Vos. 2024. EnergiBridgeWrapper. (2024). https://github.com/mitalipatil99/Jupyter_Energi

[2] Luís Cruz. 2021. Green Software Engineering Done Right: a Scientific Guide to Set Up Energy Efficiency Experiments. (2021). https://luiscruz.github.io/2021/10/10/scientific-guide.html

[3] Cameron Davidson-Pilon. 2024. Probabilistic Programming Bayesian Methods for Hackers. (2024). https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers

[4] Thomas Durieux. 2022. EnergiBridge. https://github.com/tdurieux/EnergiBridge. (2022).

[5] Marcel Garus. 2022. jupyter-resource-usage. (2022). https://github.com/MarcelGarus/jupyter-energy

[6] Green Software Foundation. 2022. Our theory of change - defining the strategy for the GSF. (2022). https://greensoftware.foundation/articles/theory-of-change

[7] Jupyter. 2022. HEOM 4: Dynamical decoupling of a non-Markovian environment. (2022). https://nbviewer.org/urls/qutip.org/qutip-tutorials/tutorials-v4/heom/heom-4-dynamical-decoupling.ipynb

[8] Jupyter Server. 2022. Fixing memory measurements for notebooks in use. (2022). https://github.com/jupyter-server/jupyter-resource-usage/issues/12

[9] Jupyter-server. 2024. jupyter-resource-usage. (2024). https://github.com/jupyter-server/jupyter-resource-usage

[10] Microsoft. 2024. Generative AI for Beginners. (2024). https://github.com/microsoft/generative-ai-for-beginners

[11] Microsoft. 2024. Sustainable Software Engineering Overview. (March 2024). https://learn.microsoft.com/en-us/training/modules/sustainable-software-engineering-overview/

[12] Jake VanderPlas. 2024. Python Data Science Handbook. (2024). https://github.com/jakevdp/PythonDataScienceHandbook