Delft University of Technology

Sustainable Software Engineering
CS4415

Energy Consumption Reporter

Authors:
Aron Hoogeveen (K.A.Hoogeveen@student.tudelft.nl)
Delano Flipse (D.E.Flipse@student.tudelft.nl)
Rodin Haker (R.Haker@student.tudelft.nl)

Coach:
Luís Cruz (L.Cruz@tudelft.nl)

Abstract

Energy consumption is becoming increasingly important as the discussion about a sustainable (digital) world continues. To reduce carbon emissions, developers needs more tools to be aware of the changing energy consumption of their software. There exist tools that provide insight into the carbon impact of your software. However, most tools report the carbon footprint for the running system as a whole and do not provide a way to track energy efficiency improvements. In this paper, we present a set of Python packages that can report the energy usage of methods, as well as integrate into existing testing frameworks. This enables developers to be aware of, and validate, the energy consumption for their software.

March 28, 2024

TU Delft Delft University of Technology

Contents

1 Introduction

In our increasingly digital society, there is a steady growth in the amount of software being developed and used. The energy consumption of data centers alone is projected to reach 2.12% of the world's total carbon emissions by 2030 [1]. Despite these numbers, software developers are lacking in energy awareness during the process of software development [2, 3]. This is an ongoing challenge that needs to be addressed.

A proposed solution is to provide tools that promote energy awareness in a way of working that is familiar to software developers [3]. We propose a tool that allows developers to gain insights into a software's energy consumption by leveraging a process familiar to them: software testing. By making energy consumption part of testing, we can make low energy consumption a non-functional requirement. We can re-use principles of non-functional testing to prevent excessive energy usage, such as benchmark testing and regression testing. By measuring the energy consumption of longer running tests (such as integration, end-to-end or acceptance tests), we can produce energy consumption insights for key scenarios of the software. This can be used to find energy consumption outliers (by leveraging benchmark testing) and to identify a sudden increase in energy consumption (by using regression testing). In order to create a tool that provides value we need to answer the following questions.

Question 1: Which energy metrics do software developers need to create awareness for the energy consumption of their software?

Once we know which metrics are useful, we need to get access to them. Therefore we need to know:

Question 2: What solutions can we use to measure the energy consumption of software?

The remainder of the paper is organized into the following sections. In Section 2 we will look into the literature to find relevant energy metrics, to answer Question 1. Then, in Section 3 we research possible methods of acquiring these energy metrics to answer Question 2. Next, in Section 4 we detail the design and implementation of a proof of concept that uses a relevant method from Section 3 to produce the metrics defined in Section 2 and integrates this into an existing test framework. In Section 5 we discuss a validation of the designed proof of concept. Then in Section 6 we talk about the limitations of the system and gained insights. Lastly, in Section 7, we give our conclusions and recommendations for future research and development.

2 Identifying Relevant Energy Metrics

In this section, we highlight our target group, their needs, and find the relevant energy metrics that they need.

2.1 Identifying the Target Group

Our target group primarily comprises developers involved in software development projects. By catering to developers, we aim to empower them with the insights and tools necessary to make informed decisions regarding energy efficiency in their software designs and implementations.

Expanding on this, developers play a pivotal role in shaping the carbon footprint of software systems. Their choices in coding practices, algorithm selection, hardware utilization, and architectural design significantly influence the energy and power consumption of software applications. Therefore, equipping developers with a comprehensive understanding of energy and power consumption is essential for fostering a culture of energy-conscious software development.

2.2 Identifying the Energy Metrics

For our target group we need to define metrics that allow them to make an informed decision about energy consumption. Based on existing literature we identified the following metrics as candidates for our tool [4, 5]:

- **Energy (J)**: Energy measure in joules is used to determine the energy consumption of a one-off task [4, lecture 3]. Using this metric can be useful to measure the energy consumption of a key scenario that involves user interaction. Energy can be measured using hardware, software reading hardware sensors, or software approximation. It is directly related to carbon emission, but the ratio of emissions per unit of energy depends on the source of energy.

- **Average Power (J/s)**: Power measured in Watts is used to determine the energy consumption of a continuous task [4, lecture 3]. This metric can be useful to measure the power consumption of a key scenario of a software product that is done passively, or where execution time is not relevant. Both the measurement of, and conversion to carbon emission, are similar to how energy is measured.
- **Energy Delay Product (J·s)**: To optimise for both lower power consumption and the duration of a task, the Energy Delay Product (EDP) can be a useful comparison metric. It can be derived from the energy used and task duration. As such, it is a derived property of either energy or power combined with the task duration. Measuring the duration of the task is therefore enough to capture the EDP.
- **Carbon Emission (kgCO_2eq)**: As the main contributor to climate change, this is a relevant metric to have. Yet, measurement is a challenging process. Energy and power can be directly measured using relatively straight-forward methods. This is not the case for carbon emissions, as carbon emissions for electrical systems are indirectly produced. Carbon emissions of energy depend on the carbon intensity of the source of energy. This intensity can vary greatly based on time of day, location [6]. This metric is relevant for the measurement of running systems, but less so for measuring sub-parts of the system during development. Especially as the carbon emission can differ between the system that developers use.

Conclusion

We conclude that the most relevant metrics needed to create awareness in our target group are energy, power and EDP. Although carbon emissions would be a useful metric, it is not feasible to include it given the time constraints of this work. Still, the chosen metrics will give enough insights into energy consumption to make informed decisions during the development of software.

3 Comparing Methods of Energy Consumption Measurement

There are multiple ways to measure the energy consumption of software. These can be subdivided into two groups: hardware power monitors and software energy profilers [7].

Hardware power monitors (HPM) are devices that can power your device instead of the normal power supply unit. They determine the actual energy consumption of the system by measuring input voltage and input current. A great advantage of these devices is that it can precisely measure the total power of a system. However, there are two drawbacks. The first being that a HPM reports the power consumption of the total system. It is not possible to isolate the power consumption of a single application. The second and largest drawback is that requires a physical device that needs to connect to the system under test. Most software tests run automatically on cloud servers in continues-integration / continues-development pipelines. In these cases it is not possible to connect a HPM to the device running the tests.

Alternatively, there are Energy profilers (EPs) th run as software on the devices. This mitigates the second drawback of the HPMs and are therefore preferred. There is a broad variety in EPs and how they measure or estimate the power consumption. In the following subsections we list some energy profilers that we considered for this project. For each of them we discuss their advantages and disadvantages.

3.1 EnergiBridge

EnergiBridge[1] is a relatively new tool that aims to be a cross-platform energy profiler [8]. From its documentation we know that it is "designed to collect resource usage data for a command to execute and to output the data in a CSV format". We want to gather energy metrics per test. This is not possible when you have to provide a command each time when you want to measure a test. Furthermore, the tool requires admin rights to read out the CPU sensor values. For cloud solutions this is not possible.

3.2 SPECPower Model

The SPECPower model[2] is a tool which provides an estimation of the used energy based on provided input parameters. The model can even provide estimations with only the CPU utilization as input. This means that it can provide estimations on virtually all systems (no privileged system rights are needed to get this data). The documentation states that "Its use is the estimation of the current power draw of the whole machine in

[1] https://github.com/tdurieux/EnergiBridge
[2] https://github.com/green-coding-solutions/spec-power-model

Watts". Normally this would mean that on shared hardware it is not possible to retrieve energy estimation for your software. However, the model exposes extra parameters that enable you to specify a 'VHost Ratio' which can be used to estimate the energy of only your part of the shared hardware.

3.3 Intel Performance Counter Monitor

Intel Performance Counter Monitor (PCM)[3] is the successor of Intel Power Gadget. It provides energy metrics for Intel's Intel Core, Xeon, Atom and Xeon Phi processors. Although it runs cross-platform it only provides data for Intel processors. As we want to reach a broad audience of developers, it is not a viable option.

Conclusion

The main trade-off in choosing the right EP is 'accurate results' versus 'portability'. Since Intel PCM can only determine metrics for Intel CPU's it can be discarded. Comparing EnergiBridge and SPECPower Model, the latter is the most portable solution, as it can run in any non-administrative environment. Since we want to support as many different types of hardware as possible in as many environments as possible, we pick the SPECPower Model as our tool of choice. A drawback of this choice is that it does not provide the most accurate results. However, we favor portability over accurate results.

4 Design and Implementation of the Energy Reporter

We have designed and implemented a custom energy measurement tool[4] to measure the energy and power consumption of functions and parts of code using an energy profiler. As a proof of concept to demonstrate how this tool can be integrated into existing tests, we have designed and implemented a plugin[5] for the popular Python test framework pytest[6] [9] utilizing this tool to add energy metrics to the test reports.

4.1 Energy Measurement Tool

The energy measurement tool utilizes the SPECPower estimation model [10] to generate reports on the energy consumption of a specific function. This is achieved by sampling the CPU utilization during the execution of a function, and then estimating the power and energy consumption by employing the SPECPower model.

To utilize the SPECPower model, we first need to detect the parameters for of the system. These parameters will be used to select the correct data within the SPECPower data set. This data is then used to train the model to predict as accurate values as possible for the current system.

We have extended and improved the existing automatic detection module to work on more machines and architectures, specifically Windows systems. It also utilizes the TDP list provided by CodeCarbon[7] to detect the correct TDP value for the current system.

Next, we expose an energy tester module, which provides multiple ways to test the energy consumption of a single method. When called, it starts a new measurement process, which continuously samples the CPU utilization of the process that called it. Each sample is then given to the the model to estimate power and energy usage. In order to get a relative energy consumption, we optionally allow subtracting the background power defined by a utilization of zero. Once the measurement process is started, the energy tester module calls the function under investigation. When the tested function terminates, it signals the measurement process to finish measuring and waits for it to return its measurements. The energy tester module then saves and returns these results in a report. Optionally, the current state of the report can be saved to disk, to make it available for later analysis. The report could also be directly showcased in the terminal, either in conjunction with or in place of being stored in a file.

The default measuring model is the SPECPower model, but the tool allows for different energy profilers to be integrated into the measuring process. The whole package is made available as a Python package[8], and can be used for different purposes then our own plugin.

[3] https://github.com/intel/pcm
[4] https://github.com/aron-hoogeveen/energy-consumption-reporter
[5] https://github.com/delanoflipse/pytest-energy-reporter
[6] `https://pytest.org/`
[7] https://github.com/mlco2/codecarbon/blob/master/codecarbon/data/hardware/cpu_power.csv
[8] `https://pypi.org/project/energy_consumption_reporter/`

4.1.1 Defining the Measurement Output Report Format

In order to allow the convenient integration of the energy measurement tool in other applications, we define a clear and reusable output format.

To allow for the processing of the output data by both machines and humans, we choose the well established human-readable ECMA-404 JSON syntax[9]. The basic structure is inspired by a JSON data standard from Tesults[10]. This basic structure can be found in Appendix A.1.

The per-test data fields are derived from the energy metrics from Section 2.2. The developer might choose to run the tests multiple times in order to increase the reliability of the data. The number of runs will be saved in the appropriate data field.

For determining the high-level data fields we look at the needs of the software developers. Software developers working on improving the energy consumption of a specific part of the code need to compare the energy consumption of the original code with the new code. They might even want to try different programming approaches and compare the resulting energy consumption differences. Therefore, they need a way to differentiate the different reports for each of these scenarios.

The report generated will have the date and time embedded in its filename. Additionally, since developers commonly use version control systems (e.g. git) to track their different versions of the code. If they do, the commit current commit hash will be embedded within the file to facilitate tracking of code changes. Lastly, the energy consumption of software will differ from hardware to hardware and, therefore, the system information is saved within the report. The resulting JSON template can be found in Appendix A.2.

4.2 Pytest Plugin

On top of the measurement tool, we built a plugin for the pytest framework. The plugin allows tests to be marked as “energy tests”. Once the tests are finished, it provides a summary of the energy consumption of these “energy tests” to identify potential sources of excessive energy usage. Furthermore, it can save a detailed report on energy consumption. This report that can be used to provide a history of energy consumption for the same set of tests.

The plugin acts as a bridge between the pytest framework, the measurement tool, and the estimation model. Figure 1 highlights the interaction between the different elements. First, we initialize the plugin, and the model (using the auto detection module). Then, the plugin hooks into the call to every test method. Instead of running the test once, the plugin now runs the test a configurable number of times. For each iteration it calls the measurement tool to measure the test method. It then stores the results. After this is done for each test, for each iteration, the plugin will create a summary.

[9] https://ecma-international.org/publications-and-standards/standards/ecma-404/
[10] https://www.tesults.com/docs/tesults-json-data-standard

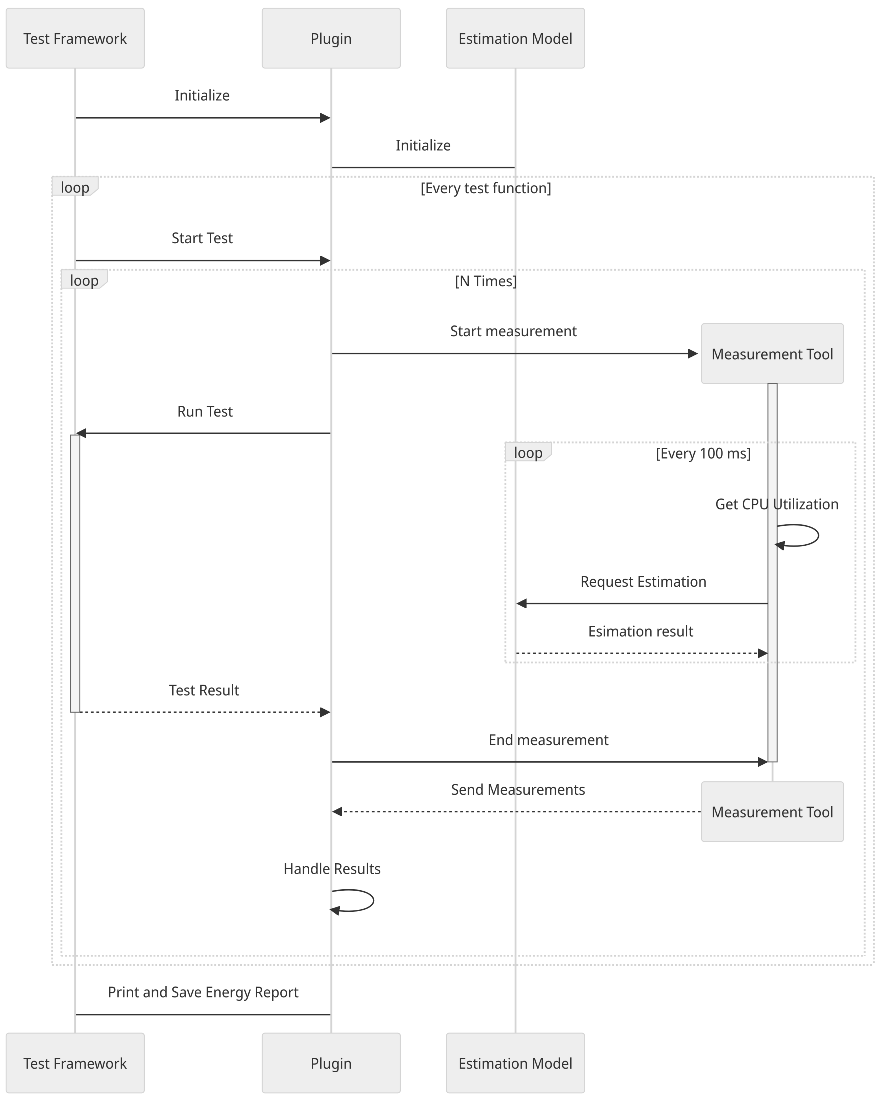


Figure 1: A sequence diagram of the interaction between the test framework, the plugin, the estimation model and the measurement tool.

Although simplistic in nature, the plugin provides a flexible integration into existing projects by working as an installable Python package[11]. Since it uses an approximation model, it can also be run in a CI/CD pipeline, allowing it to be integrated in the modern development process.

5 Validating the tool

To validate the effectiveness of our plugin and tool, we will create a test suite to represent and validate assumptions about the energy consumption of software, and compare these with the reporting of our tool and plugin. In addition to this, it would be beneficial to examine realistic scenarios and usages, but due to time constraints we are limited to a constrained validation.

5.1 Methodology

To reason about the effectiveness of our plugin and tool, we to be able to define:

- The effect of more iterations on the quality of the energy test, as we want to know how many iterations yield a reliable output.

[11]https://pypi.org/project/pytest_energy_reporter/

- The effect of running the same computation multiple times. In this scenario one would expect a linear increase in energy consumption. This can give us confidence in the quality of the measurements.
- The effect of doing no computations. We expect to see a decrease in energy consumption compared to test that actually compute something, but we don't expect to see zero energy usage for doing no computation, as there always is overhead and background energy usage.
- The effect of introducing a delay into a computation, so its average power usages is lower. We expect to see a decrease in power and an increase in EDP.

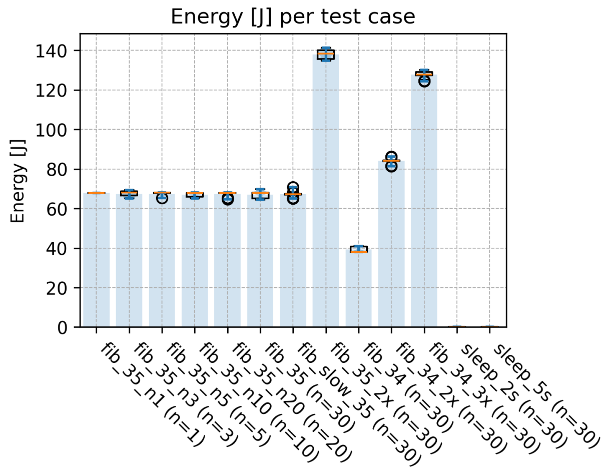
Listing 1: An inefficient implementation of calculating the nth Fibonacci number.

```python
def fib(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
    return fib(n-1) + fib(n-2)
```

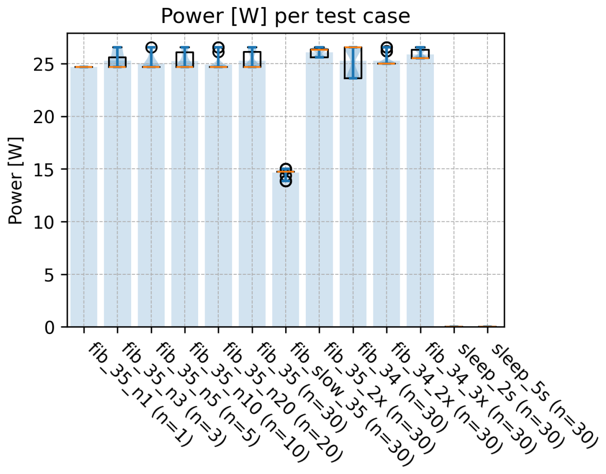
To measure these scenarios, we built a test suite which we measure using our plugin. To provide a computationally expensive method, we (inefficiently) compute the Fibonacci number as defined in Listing 1. We then define test cases to calculate the 35th Fibonacci number (which takes a reasonable amount of time) with increasingly higher iterations, calculate it multiple times, and calculate it with or without artificial delays. We further have test cases that calculate the 34th Fibonacci number (thus doing less work), and test cases that make the process sleep for a number of seconds.

5.2 Results

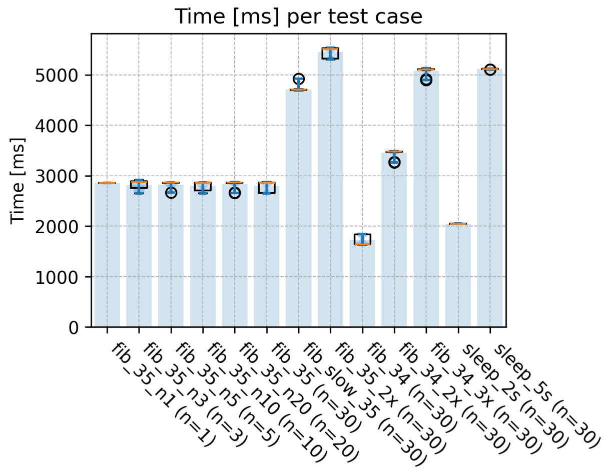
Given the test cases, we run the test suite with our plugin enabled. Figure 2 shows the results for the test cases and their measured (average) energy, power, EDP and duration. The results were made by running them inside a github action, to ensure a reproducible setup. The measurement tool is set to remove the background power.



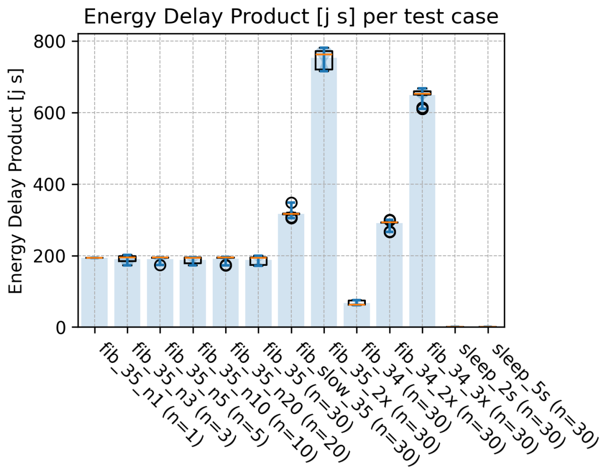
(a) Energy consumption for the different test.



(b) Power consumption for the different test cases.



(c) Time duration for the different test cases.



(d) Energy Delay Product for the different test cases.

Figure 2: Energy and power consumption for the different test cases that either calculate Fibonacci numbers (one or more times), call sleep for a given duration, or a combination of both. The bars shows the mean values for all iterations, and the box plots the variation between iterations.

For each of the different cases we can see the following results:

- Performing more iterations has minimal effect on this specific test case. The maximal coefficient of variance is low with a value of only 0.058.
- Performing the same work twice leads to an average energy increase of 106.7%.
- Performing no computation caused a decrease of 29% in power consumption.
- Performing the same work with a delay creates the expected result of consuming 0.23% more energy but having a 68.8% increase in EDP, when compared to a case without a delay.

The results correspond to our expectations of energy usage. Therefore, we conclude that our plugin and measurement tool work as expected, under the given conditions. We also conclude that one iteration can already be useful, but it is more accurate to use at least three iterations.

6 Limitations and Gained Insights

The choice of energy profiler has a large impact on the limitations of the tool. The profiler was not made to estimate the energy of only a single process but of the whole system[12]. However, by only providing the CPU utilization for the process under test, instead of the total CPU utilization, we make the results more accurate for estimating a single process.

The model is trained on a subset of the SPECPower dataset, which comprises system-specific information. However, it's important to note that this dataset may not cover every hardware configuration. To address this limitation, default values are assigned when the hardware is not supported. However, this approach can lead to inaccuracies in energy consumption estimations, especially when dealing with hardware not represented in the dataset.

Furthermore, the SPECPower Model has not been trained on data with GPU usage. This greatly reduces the accuracy for programs under test that make use of GPUs. Ideally, we want to support measurements for both CPU and GPU heavy programs.

Due to time constraints, the tool has not been thoroughly tested on all platforms. Only three different pieces of hardware were available to develop and test on. The tool has been tested on a limited set of Windows and Linux (Ubuntu) systems. For this tool to get a large foothold in the programming community it must first be further developed and tested on more hardware.

7 Conclusion and Future Recommendations

We have developed a tool to measure energy and power consumption, aligning with the research questions outlined in the introduction. This tool provides detailed reports on energy metrics necessary for developers to make informed decisions regarding their software's energy efficiency. Utilising the SPECPower model as its backbone, the tool measures the energy consumption of a given function using the CPU utilisation of the process. Furthermore, we have introduced a proof of concept capable of measuring energy consumption at the test level. Leveraging the pytest framework, this proof of concept enriches test results by incorporating the energy metrics. By seamlessly integrating energy consumption analysis into the testing workflow, we aim to equip developers with the tools necessary for thorough energy-aware software testing and optimization.

The results show that our tool does have realistic properties of energy consumption. We believe that the current and future iteration of a this tool hold potential in cultivating energy awareness among software developers, as well as making energy consumption considerations part of the software development process.

To maximize the impact of this tool, we propose the following steps:

- **Leverage Energy Reports for Feedback:** Incorporate energy reports into the review process to provide feedback on energy consumption changes. This integration will enable developers to identify and address energy inefficiencies during code reviews, fostering a culture of energy-aware software development.
- **Demonstrate Preventive Potential:** Integrate the plugin into existing projects known to have resolved energy bugs. By showcasing how the tool can prevent such issues from occurring in the future, developers gain confidence in its effectiveness and are motivated to adopt it as a proactive measure against energy-related issues.
- **Enhance Platform Compatibility:** Extend the functionality of the tool to support relevant major platforms and hardware configurations. This expansion ensures broader accessibility and applicability, enabling developers to assess energy consumption across diverse environments accurately.
- **Evaluate Model Accuracy:** Conduct a comparative analysis of alternative energy approximation models to determine the most accurate model. By evaluating different models' performance against real-world data, developers can select the model that best aligns with their specific requirements and use cases.

[12]https://github.com/green-coding-solutions/spec-power-model/blob/main/README.md

References

[1] M. Koot and F. Wijnhoven, "Usage impact on data center electricity needs: A system dynamic forecasting model," *Applied Energy*, vol. 291, p. 116798, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261921003019

[2] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about the energy consumption of software?" 07 2015.

[3] G. Pinto and F. Castor, "Energy efficiency: a new concern for application software developers," *Commun. ACM*, vol. 60, no. 12, p. 68–75, nov 2017. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3154384

[4] "2024 | SustainableSE," Mar. 2024, [Online; accessed 27. Mar. 2024]. [Online]. Available: https://luiscruz.github.io/course_sustainableSE/2024

[5] "Green Software Practitioner," Mar. 2024, [Online; accessed 27. Mar. 2024]. [Online]. Available: https://learn.greensoftware.foundation/energy-efficiency#energy-measurement

[6] N. Scarlat, M. Prussi, and M. Padella, "Quantification of the carbon intensity of electricity produced and used in europe," *Applied Energy*, vol. 305, p. 117901, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261921012149

[7] L. Cruz, "Tools to measure software energy consumption from your computer," http://luiscruz.github.io/2021/07/20/measuring-energy.html, 2021, blog post.

[8] J. Sallou, L. Cruz, and T. Durieux, "Energibridge: Empowering software sustainability through cross-platform energy measurement," 2023.

[9] "PyPI Download Stats," Mar. 2024, [Online; accessed 27. Mar. 2024]. [Online]. Available: https://pypistats.org/packages/pytest

[10] "spec-power-model," Mar. 2024, [Online; accessed 27. Mar. 2024]. [Online]. Available: https://github.com/green-coding-solutions/spec-power-model

A Appendix A

A.1 Basic JSON template

The original design document can be found in the docs of the repository[13].

Listing 2: Basic JSON template.

```
{
    "results": {
        "name": "<user defined descriptive name>",
        <!-- other high-level information -->,
        "cases": [
            {
                "name": "<some test method name>",
                "result": "pass",
                "reason": "",
                <!-- other test-level information -->,
                "_userDefinedField": "<some value>"
            },
            {
                "name": "<another test method name>",
                "result": "fail",
                "reason": "AssertionError: Invalid Operation",
                <!-- other test-level information -->,
                "_userDefinedField": "<some value>"
```

[13] https://github.com/aron-hoogeveen/energy-consumption-reporter/blob/1ff727b05227f2bf9b736ddbaee7dac0465f35c3/docs/report-file-format.md

```
            }
        ]
    }
}
```

A.2 Final JSON template

Listing 3: "Full JSON template."

```
{
    "results": {
        "name": "Hello World!",
        "description": "This is some extra description that can be displayed
            for convenience.\nDo whatever you need.",
        "version": 1,
        "software_version": "v1.2BETA",
        "commit": "755f02b971d59acd85d3aa727baa0e822efcd73f",
        "date": "2024-03-11T16:50:00",
        "model": "https://github.com/green-coding-solutions/spec-power-model
            ",
        "hardware": {
            "PC_name": "<value>",
            "CPU_name": "<value>",
            "CPU_temp": <value_in_degrees_celcius>,
            "CPU_freq": <value_in_MHz>
        },
        "cases": [
            {
                "name": "test_user_supplies_incomplete_information",
                "result": "fail",
                "reason": "AssertionError: Invalid Operation",
                "N": 3,
                "execution_time": [
                    <milliseconds>,
                    <milliseconds>,
                    <milliseconds>
                ],
                "energy": [
                    <joules>,
                    <joules>,
                    <joules>
                ],
                "power": [
                    <Watts>,
                    <Watts>,
                    <Watts>
                ],
                "edp": [
                    <joules-second>,
                    <joules-second>,
                    <joules-second>
                ],
                "_my_custom_field": "This is so epic!",
                "_test_params": {
                    "param1": "<value1>",
                    "param2": "<value2>"
                }
            },
            {
                "name": "test_user_supplies_complete_information",
                "result": "pass",
                "reason": "",
                "N": 3,
                "execution_time": [
                    <milliseconds>,
                    <milliseconds>,
                    <milliseconds>
                ],
                "energy": [
                    <joules>,
                    <joules>,
                    <joules>
                ],
                "power": [
                    <Watts>,
                    <Watts>,
                    <Watts>
                ],
                "edp": [
                    <joules-second>,
                    <joules-second>,
                    <joules-second>
                ],
                "_my_custom_field": "This is so epic!",
                "_test_params": {
                    "param1": "<value1>",
                    "param2": "<value2>"
                }
            }
        ]
    }
}
```