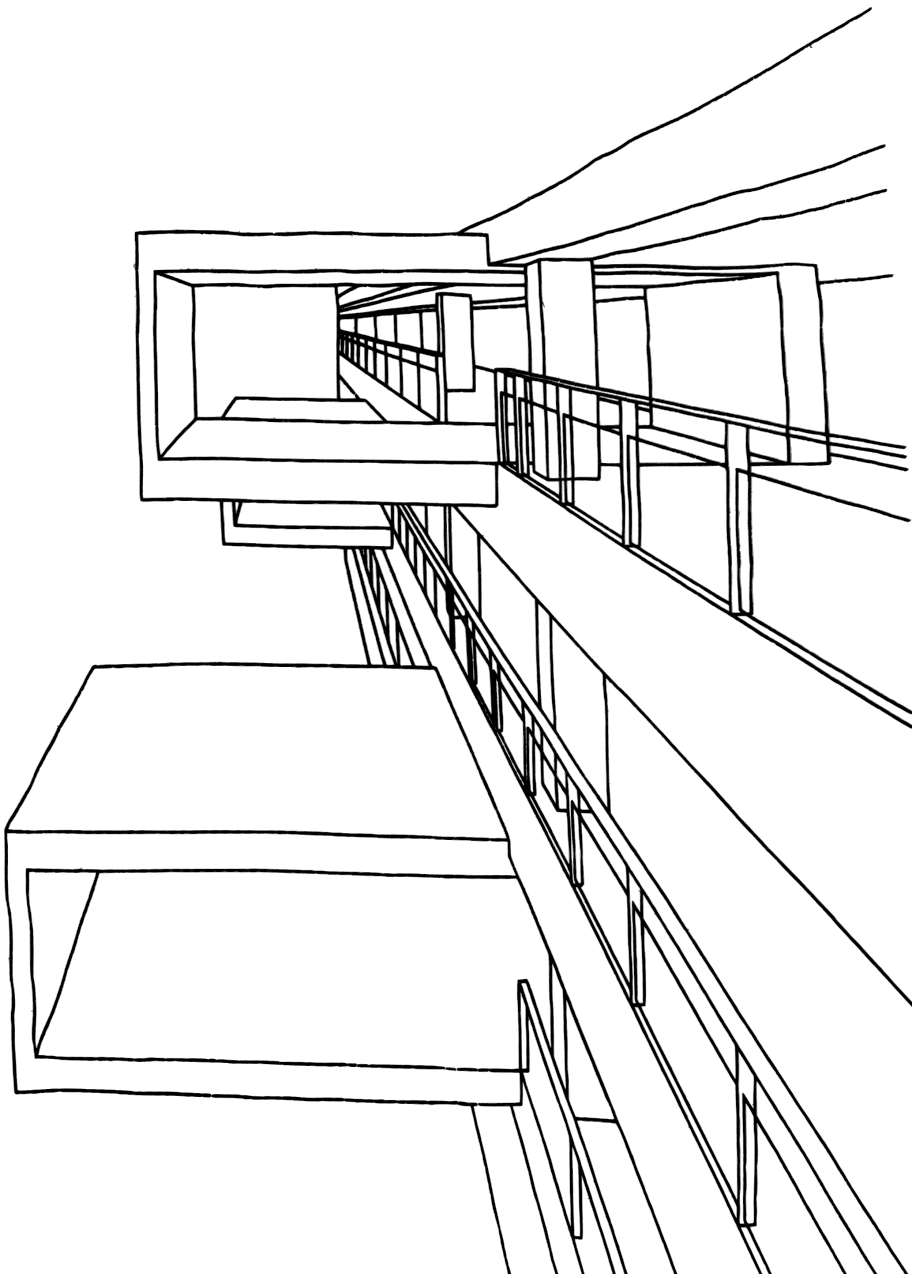
IT UNIVERSITY OF COPENHAGEN

Computer Science MSc

Research Project

Oskar Breindahl (osbr@itu.dk)

Course Code: KIREPRO1PE
December 2024



Contents

1 Introduction

Python is a popular programming language used in many applications worldwide, and was the fastest growing programming language in 2020[13]. Python is known for bad performance compared to other languages, and consequently also bad energy efficiency[10]. While the developers of Python work to improve the performance of the language, the Python Enhancement Proposal (PEP) index currently contains no proposal to improve energy efficiency[4].

The goal of this research project is to develop an experimental setup capable of benchmarking Python's energy efficiency, and documenting initial results. The contributions of this project are as follows:

- Design and setup of experiment to benchmark Python energy efficiency.
- Initial results of running aforementioned experiment.
- Initial analysis of aforementioned results.

These contributions will be used in a future thesis that aims to enable Python developers to test and improve the energy efficiency of the Python programming language alongside its performance.

This effort is based on *Pyperformance*, which is an open-source benchmarking tool used by developers of the Python language. When the developers report performance improvements, as seen in the release notes for Python 3.12[14], Pyperformance has been used to measure the scale of those improvements. The source code for Pyperformance is available on GitHub[1].

As stated in the tool's documentation, the purpose of Pyperformance is to provide real-world benchmarks based on actual Python applications, to ensure results that are as relevant as possible[2]. These same reasons makes running Pyperformance an ideal scenario in which to measure device energy consumption. The tool runs on the standard implementation of Python, known as *CPython*. In this report, "Python" refers to CPython.

To measure and compare energy consumption, a lab experiment is set up. The experiment consists of a laptop computer connected to a power meter, which is connected to a single-board computer known as a Raspberry Pi running a number of Pyperformance benchmarks. To make the measurements relevant to the improvement of Python's energy performance, benchmarks are run on three different versions of Python, on two different Raspberry Pis. The results and their implications are then presented and discussed.

2 Related Work

The energy efficiency of programming languages in general is a well studied field. When comparing languages to each other, Python ranks in the bottom in terms of performance and energy consumption, as presented by Georgiou et al.[5], Pereira et al.[9, 10] and Koedijk and Oprescu[7]. Methods of measuring energy consumption in these studies range from external monitoring tools, akin to the one used in this

project, to software based ones that queries functions provided by the onboard processor. While Python generally performed worse than other languages, there was not a direct link between performance and energy consumption in all cases; interpreted languages such as Java performed worse but had better energy efficiency than compiled languages such as C and C++ in some cases.

Research investigating the energy efficiency of Python specifically is also prevalent. These papers often focus on specific use-cases, rather than benchmarking. Pfeiffer, which inspired this project, compares the energy consumption of different versions of Python in an experimental setup very similar to the one used in this project[11]. The paper finds that, in the given setup, newer versions of Python perform better, with Python 3.12 being fastest. Lamprakos et al. investigates how altering Python's memory allocation schemes can improve energy efficiency, and find that it can be improved quite significantly[8]. Holm et al. attempts to replace C++ with Python in GPU computing scenarios, finding that they are able to match the energy effieciency of C++[6]. They note that Python eases implementation due to the high level nature of the language, motivating the switch from C++. While these studies all shed light on different aspects of Python's energy efficiency, they do not provide a platform for benchmarking Python versions across diverse use-cases and hardware as is done in this project.

3 Experimental Setup

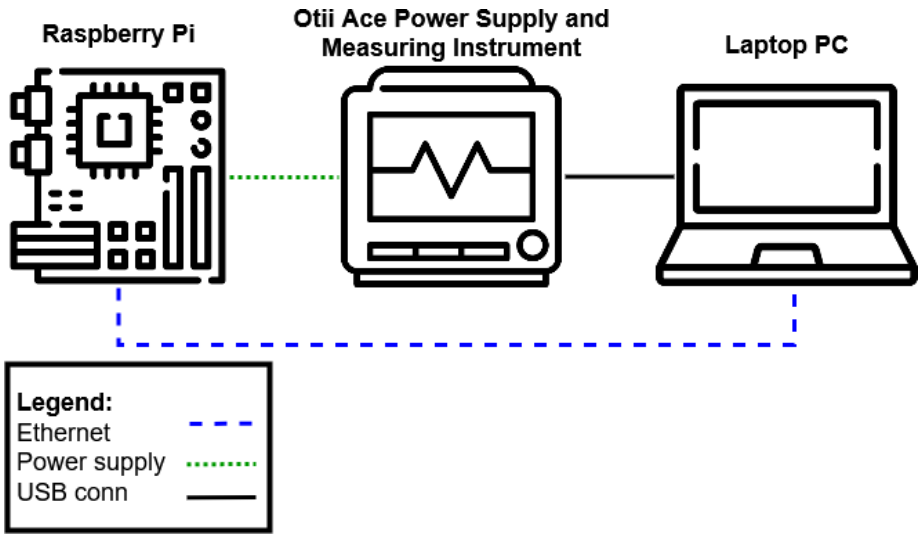


Figure 1: Experimental setup for measuring Python energy efficiency

A basic figure of the experimental setup can be seen in Figure 1. The setup consists of three main parts: A laptop PC, a *Raspberry Pi* (RPi) single-board computer and an *Otii Ace Pro* power-meter. The laptop is connected to the power-meter via USB

cable for reading data and ethernet for network access. The power-meter is powered by a 25 volt power supply, and in turn powers the RPi via USB. The RPi is also connected to ethernet for network access. The laptop uses Otii 3 software to control the power supply from power-meter to RPi and record data.

The RPi's software has been configured beforehand to include Python 3.12 and 3.13. Additionally, it contains Python 3.9 natively in the operating system (OS) Raspbian. Raspbian is an OS based on Debian, and is chosen because it is optimized for Raspberry Pi hardware[12]. Pyperformance has also been individually installed for each version of Python, as is required. For this experiment, the benchmark "mako" is used. While Pyperformance has many benchmarks to chose from, mako is chosen because of relatively short running time during testing, and perceived relevance; mako is a widely used HTML template builder[3] and as such is understood as a relevant use case for Python.
The control flow for running a benchmark is as follows:

1. Python script (`otii_config.py`) starts recording on power-meter
2. Python script connects to RPi via SSH and executes bash script (`pypower.sh`) with pre-written parameters
3. Bash script runs benchmark
4. Python script stops recording on power-meter when bash script finishes execution
5. Python script closes SSH connections and finishes execution

Benchmarks are run on a Raspberry Pi 3 Model B+ and a Raspberry Pi 4 Model B to compare performance across different hardware setups. The main difference between these is the CPU; an ARMv8, 64-bit at 1.4GHz on the RPi 3 and an ARM v8, 64-bit at 1.8GHz on the RPi 4. On each RPi, the process listed above is repeated ten times for each of the three versions of Python to ensure a robust dataset for analysis. Recordings are done for the channels Main Current (ampere), Main Power (watts) and Main Voltage (volts) simultaneously.

The resulting recordings are downsampled from 1000 to 10 samples per second (sps) to make the export files more digestible. They are then exported as CSV-files, cleaned and combined to enable analysis. Analysis is done in Python with the packages Pandas, Matplotlib and Seaborn. All code for the entire experiment can be found at `https://github.com/oskarbreindahl/reproj2024`.

4 Results

In this section, the results of the experiment are presented in the form of three graphs[1]. While three channels were measured for each benchmark, voltage remained

[1]Cleaned data used to produce the graphs can be found in the previously linked repository. Uncleaned data can be found at `https://docs.google.com/spreadsheets/d/1XaP2RWFwa6S0qfyKfYrWt6t61CpLrjeAYE2wIeVtvo4/edit?usp=sharing` and `https://docs.google.com/spreadsheets/d/1EdKtvsTGAhpcNRMoIiMDc1yl6cQ5HMrwO4pfMMfItOE/edit?usp=sharing`

constant at 5V throughout, and thus was deemed insignificant. Additionally wattage was observed to be directly and proportionately linked to amperage, and so only the graphs for amperage are presented, since the corresponding ones for wattage are visually identical.

The graphs portray the average amperage for the duration of the benchmark. This average is calculated by averaging the amperage for each individual decisecond across all ten benchmarks for a given Python version.

It is worth noting that the averages run for as long as there is data. Therefore, all graphs run as long as the longest benchmark among the ten, and thus their length is not representative of average running time. Additionally, data from one of the Python 3.9 benchmarks on the RPi 4 was corrupted, and so only nine benchmarks are used to inform the corresponding graph.

Standard deviation was observed to be very low between benchmarks on a given version, indicating reliable measurements.

Figure 2 shows the results from the Raspberry Pi 3 Model B+, Figure 3 shows the results from the Raspberry Pi 4 Model B, while Figure 4 shows the results from Figure 2 and Figure 3 in the same graph for comparative purposes.

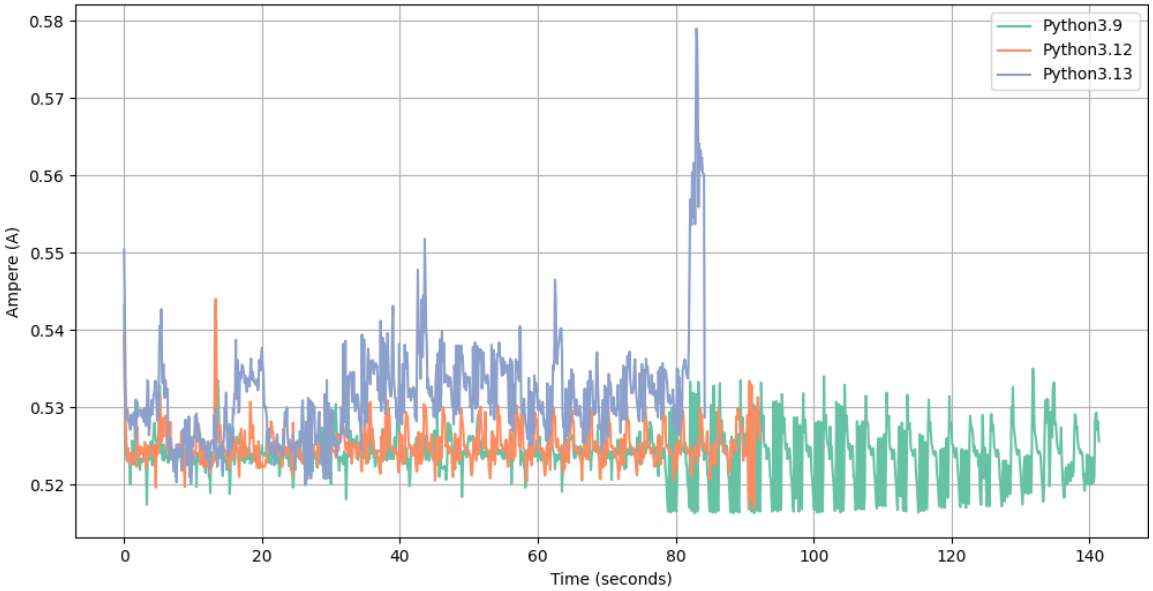


Figure 2: Average amperage while running "mako" on RasberryPi 3 Model B+

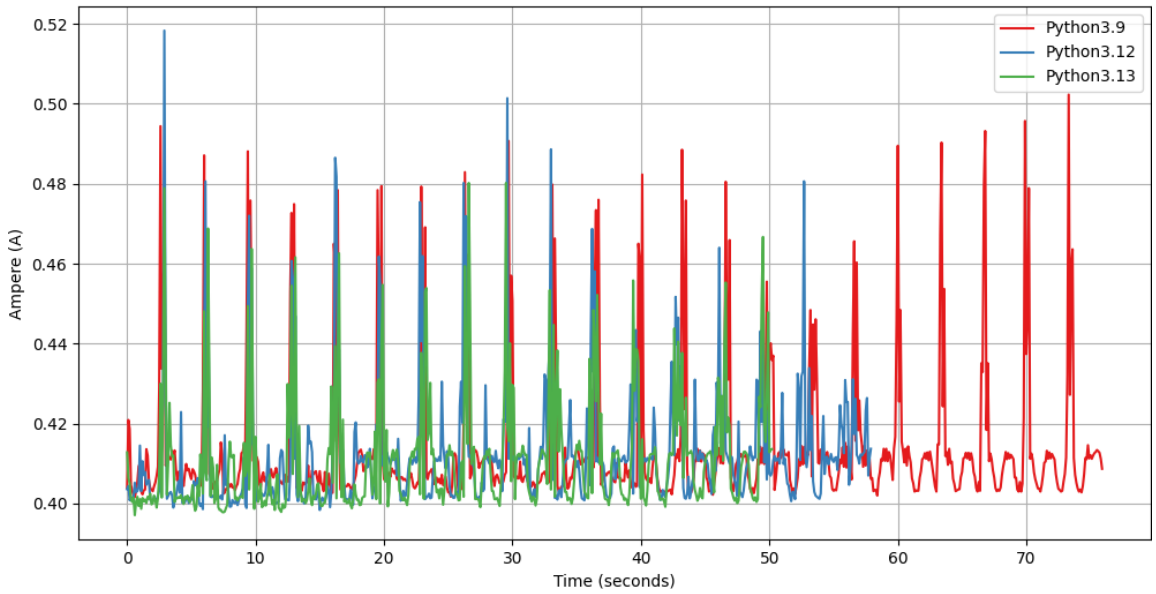


Figure 3: Average amperage while running "mako" on RasberryPi 4 Model B

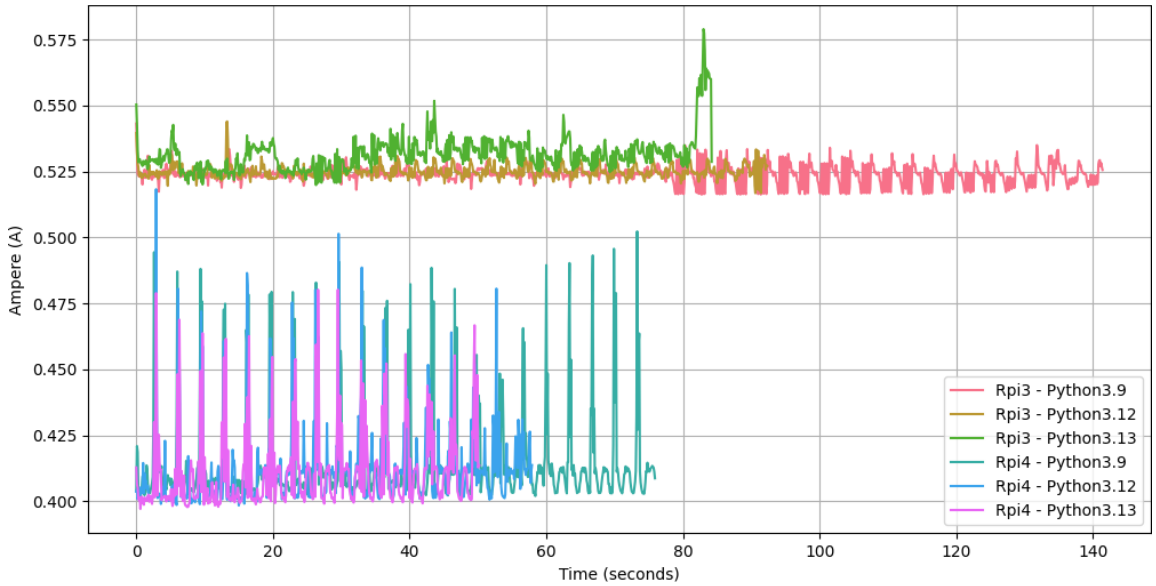


Figure 4: Comparative average amperage while running "mako"

5 Analysis

A visual inspection of the graphs yield several takeaways. Looking at Figure 2, it is readily apparent that significant improvements have been made between Python versions regarding performance. While Python 3.9's worst-case running time is just over 140 seconds, 3.12 is much faster at just over 90 seconds, while 3.13 improves this further to just over 80 seconds. Looking at the profiles of the individual graphs for each version, it is interesting that 3.13 seems to have a more erratic amperage, with spikes and drops that are significantly more prominent than the other versions. It is worth noting that the large spike in the end of the graph for 3.13 is informed by just two of the ten benchmarks for that version. These two benchmarks were the only ones that exceeded 82 seconds, and both had a sudden surge in amperage in the extra two seconds they ran. Determining whether this behaviour is anomalous or symptomatic would require a larger sample size.

Looking at Figure 3, one can also see a marked improvement in worst-case running time between versions. The spikes in amperage here seem very pronounced, indicating perhaps a more effective power utilization than on the RPi 3. Overall, running time is also much shorter, maxing out at just over 75 seconds. The different Python versions also seem very close to each other in amperage, both on average and during surges, than was the case on the RPi 3.

In the comparative view provided by Figure 4, the large difference between hardware setups becomes apparent. The RPi 3 runs at a rather constant, 0.525 A amperage, and does not generally spike more than 0.025 A during benchmarking. Conversely, the RPi 4 runs at around 0.4-0.41 A current, but spikes with more than 0.1 A during execution. This would indicate that the newer processor allows for much more careful power management, only drawing larger amounts of power when needed, while the older processor maintains a relatively high amperage with very little variation. The overall shorter running time is also obvious, with the RPi 4 almost halving worst-case running time for Python 3.9 and significantly shortening it for 3.12 and 3.13.

6 Discussion

The findings of the experiment in this report, while somewhat limited in scope, suggest several interesting things regarding the impact of Python versions and hardware setup on performance and energy efficiency.

The differences in running time between versions are significant, but mostly speak to increased performance. Intuitively, an improvement in performance is also an improvement in energy efficiency, since the device running the code will need to be powered for a shorter amount of time. However, it would have more complicated implications if average amperage during specific execution points varied across Python versions. This would especially be true if newer versions of Python had higher energy consumption at certain points than older versions. While such behaviour is not definitively proven in this experiment, there is a suggestion of it in the generally higher amperage on Python 3.13 than the other tested versions on the RPi 3. Here,

Python 3.13 executes faster, but can be observed to generally have an up to 0.01 A higher amperage than 3.9 and 3.12. This behaviour is not observed on the RPi 4, leading to questions as to Python 3.13's energy efficiency on older hardware, and whether improved performance can come at the expense of backwards compatibility.

It is also interesting to observe the markedly different behaviour of Python between hardware setups. As noted in section 5, the amperage is rather constant with small deviations on the RPi 3, while having large spikes and being generally much lower on the RPi 4. It is uncertain whether this difference is the result of a more efficient execution regiment on the CPU itself, or whether the CPU allows Python to use tricks of implementation to control energy consumption more tightly. Regardless of the cause of this difference, the implications could be quite significant; if the the difference lies in Python's programming, there could be large improvements made with regards to its energy efficiency when being run on older CPUs. If the difference lies in the CPU architecture, the choice of hardware becomes one of the most important factors for benchmarking Python's energy efficiency accurately.

6.1 Threats to validity

Internal threats

As the experiment relies on the specific hardware of the two RPis and only runs one benchmark, results may be hardware or benchmark specific. This could constitute a threat of confounding variables. Additionally, the use of an external power meter, along with the use of Python and bash scripts using SSH with unknown overhead, constitutes potential for measurement error.

External threats

Using only one benchmark may not reflect Python's energy performance across a broader range of benchmarks, and so threatens the generalizability of the results. Additionally, results from Raspberry Pi hardware may not generalize to other platforms or devices where Python is used.

7 Future work

The experiment documented in this report is mainly concerned with proving the experimental setup, and so many different avenues of future work present themselves.

As for the experimental design itself, there are some obvious improvements to make. Testing more versions of Python, testing on more devices with more diverse hardware configurations and running more benchmarks will provide a much more solid statistical foundation for further analysis. As the downsampling, exporting and cleaning of data from the Otii software is a quite time consuming and manual process, automation of this process could allow for much more experimentation in a shorter amount of time.

Regarding analysis of results, there a many other relevant measurements to look at than average amperage during execution. Analysing average running time, average

total power consumption and other observations could provide many more insights than presented in this report. These metrics are not easily exportable from the Otii software however, so significant time would be needed to properly extract and document them.

When looking at the analysis and takeaways of this experiment, it is clear that closer investigation of the actual implementation of Python and how it interacts with hardware could yield significant findings. This area of research is what the thesis following this research project will primarily focus on.

8 Conclusion

In this report, an experimental setup was presented for measuring the energy efficiency of a Raspberry Pi running a Python benchmark. The purpose of the experimental setup was to document and prove a method of combining the Python benchmarking tool Pyperformance with a power meter to allow for benchmarking of different Python versions' energy efficiency alongside performance on different hardware configurations.

Initial results of using this experimental setup was also presented to show how the resulting measurements could be used. While no final conclusions were presented based on the measurements, the analysis and discussion prove that the results of the experiment can be used to make certain observations regarding Python energy efficiency and performance.

To yield substantial results and insight into these areas, the experimental setup will be used in a future thesis project, which will further investigate what causes differences in energy efficiency between Python versions. This thesis will also attempt to answer how Python developers can test and improve the language's energy efficiency going forward.

References

[1] Pyperformance github repository. `https://github.com/python/pyperformance`. [Accessed on 15/12/202].

[2] Python performance benchmark suite (documentation). `https://pyperformance.readthedocs.io/`. [Accessed on 5/12/2024].

[3] Michael Bayer. Mako templates for python. `https://www.makotemplates.org/`. [Accessed on 15/12/202].

[4] The PEP Editors. Index of python enhancement proposals. `https://peps.python.org/#`. [Accessed on 5/12/2024].

[5] Stefanos Georgiou, Maria Kechagia, and Diomidis Spinellis. Analyzing programming languages' energy consumption: An empirical study. In *Proceedings of the 21st Pan-Hellenic Conference on Informatics*, pages 1–6, 2017.

[6] Håvard H Holm, André R Brodtkorb, and Martin L Sætra. Gpu computing with python: Performance, energy efficiency and usability. *Computation*, 8(1):4, 2020.

[7] Lukas Koedijk and Ana Oprescu. Finding significant differences in the energy consumption when comparing programming languages and programs. In *2022 International Conference on ICT for Sustainability (ICT4S)*, pages 1–12. IEEE, 2022.

[8] Christos P. Lamprakos, Lazaros Papadopoulos, Francky Catthoor, and Dimitrios Soudris. The impact of dynamic storage allocation on cpython execution time, memory footprint and energy consumption: An empirical study. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*. Springer International Publishing, 2022.

[9] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. Energy efficiency across programming languages: how do energy, time, and memory relate? In *Proceedings of the 10th ACM SIGPLAN international conference on software language engineering*, pages 256–267, 2017.

[10] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. Ranking programming languages by energy efficiency. *Science of Computer Programming*, 205, 2021.

[11] Rolf-Helge Pfeiffer. On the energy consumption of cpython. In *International Conference on the Quality of Information and Communications Technology*, pages 194–209. Springer, 2024.

[12] Raspbian. Welcome to raspbian. `https://www.raspbian.org/FrontPage`. [Accessed on 15/12/202].

[13] Akshansh Sharma, Firoj Khan, Deepak Sharma, Sunil Gupta, and FY Student. Python: the programming language of future. *Int. J. Innovative Res. Technol*, 6, 2020.

[14] Adam Turner. What's new in python 3.12. `https://docs.python.org/3/whatsnew/3.12.html`. [Accessed on 15/12/202].