Open Source Software Efficiency Project

Rebecca Meißner,[1,*] Michael Gatchell,[2] Yevhen Fatieiev,[3] Enrique Rozas Garcia,[4] Jose David Bermudez Perez,[5] Nicola Cerutti,[6] José Mejía,[7] David Paulovics,[8] Philine van Vliet,[9] Brian Gitahi,[10] and Johannes Postler[11]

[1] *Leopold-Franzens Universität Innsbruck, Technikerstrasse 25, 6020 Innsbruck, Austria*
[2] *Department of Physics, Stockholm University, 106 91 Stockholm, Sweden*
[3] *King Abdullah University of Science and Technology (KAUST), 23955-6900 Thuwal, Saudi Arabia*
[4] *Department of Physics, Gothenburg University, 41296 Gothenburg, Sweden*
[5] *School of Engineering, Science and Technology, Universidad del Rosario, Bogotá 111711, Colombia*
[6] *Oasis Loss Modelling Framework*
[7] *Departamento de Física, Universidad de los Andes, Cra 1 No 18A-12, Bogotá, Colombia*
[8] *Université Côte d'Azur, CNRS, Institut de Physique de Nice, 06200 Nice, France*
[9] *Laboratoire de Physique Théorique de l'École Normale Supérieure, 75231 Paris Cedex 05, France*
[10] *Allen Institute, 615 Westlake Ave North, Seattle, USA*
[11] *Txture Gmbh, 6020 Innsbruck, Austria, (not a Lindau alumni but listed due to his input as co-founder of the initial project idea)*

(Dated: April 21, 2024)

Countless open source libraries build the very foundation of modern information societies. Despite their role as fundamental components within software ecosystems, they are hidden from users and often overlooked. In this article, we show that a coordinated effort to improve their performance could achieve significant energy savings. A way of finding targets for optimization is shown and potential strategies to achieve the performance gains are laid out and backed up by two proofs of concept. Finally, an outlook on how to establish this project long-term is given.

I. INTRODUCTION

With the ever-growing concern of environmental sustainability, global energy conservation is crucial. The Information Technology (IT) sector is responsible for as much as 4% of total worldwide greenhouse gas emissions and energy consumption [1, 2]. This value is increasing by a 6% mean annual growth rate [3]. Computational hardware is continuously improving according to Moore's law, thus consuming less resources for the same computational power. However, it has already been identified [4] that efforts on the software development-side are necessary to further improve computational efficiency. Software development occurs through two distinct approaches. When a program's source code is openly accessible, allowing collaborative development and modification by anyone willing to contribute, it is termed Open Source. In contrast, Non-Open Source software employs proprietary code, limiting access and modifications, typically tailored for industrial and commercial applications. However, according to the 2024 Open Source Security and Risk Analysis Report [5], 77% of the total codebase in industry is originated from open-source code. Open source code is often developed by individual developers for no financial benefits. Fig. 1 illustrates a comic representation of this situation. Understandably, these developers cannot always put the sufficient amount of work into the source code to make it energy-efficient.

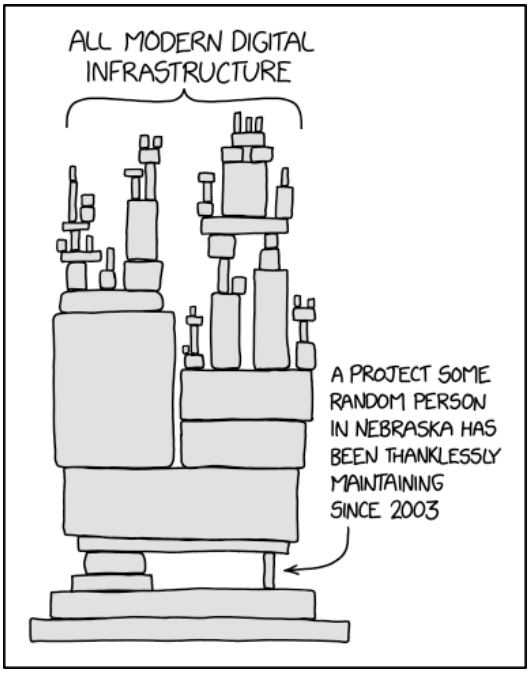

FIG. 1. XKCD's rendition of modern IT infrastructure [6]

In this paper we propose an outlook on how to a) give tools for Open Source and Non-Open Source developers to create energy-efficient software; b) provide information and education to developers for developing efficient code; c) determine which actual Open Source software repositories are worthwhile targets for optimization.

Software developers are generally trained to write faster algorithms without focusing on energy efficiency [7]. An illustrative example is parallel processing, which reduces execution time, but can end up consum-

* rebecca.meissner@rwth-aachen.de

ing more energy compared to sequential processing of individual threads [8, 9]. A study from Pang et al.[7] shows that energy-efficient software development is generally not present in the training of programmers.

Smartphone app development gained in popularity in recent years. Indeed in 2023 2.5M applications were available on Google Play Store alone [10]. Easy accessibility to mobile app development leads to potentially poorly developed applications. It is very easy for new developers to test their algorithms for functionality with available testing tools, but methods for testing energy efficiency are not readily available and Plug-and-Play.

Nowadays, with the evolution of Artificial Intelligence (such as Chat-GPT, Microsoft Copilot and Jasper), operations such as image recognition and generation, text generation among others, have exponentially increased the need for resources [11], especially during the training of such algorithms. Indeed, the training and development of a single machine learning model on a GPU cluster can have an environmental impact equivalent to that of 6 average cars in their whole lifetime with fuel consumption, including the impact of their production [12]. Models need to be regularly trained to keep up with the fast-evolving concurrence. Such neural networks rely on Open Source libraries such as PyTorch, JAX, Triton and TensorFlow. The slightest improvement in energy efficiency can lead to significant energy savings due to the very high rate of use of these libraries.

A. Related Initiatives

Over the last years, there have been multiple efforts to improve open source software efficiency and reduce energy consumption, as well as efforts to make the IT sector more energy aware. Perhaps the most well-known example is the Faster CPython project [13], initiated by Microsoft developer Mark Shannon. Microsoft employed a team of six developers, among whom is also Python founder Guido van Rossum, to speed up Python over the course of several releases. The release of Python 3.11 has already shown an improvement of $10 - 60\%$ in speed, depending on the workload, with an average increase of 25%, with respect to the previous version Python 3.10 [14]. This shows that employing a small number of dedicated developers to improve open source code gives promising results. The Faster CPython project immediately provides a new avenue for improvement: ensuring that other open source software and libraries are compatible with the improved Python releases and run at maximal speed and energy efficiency. This has already been done with Fedora 41, which looks to use the `-O3` compiler optimization flag also used by CPython for its Python package instead of the default `-O2`, which is found to increase performance by 4% .

A related initiative is the Codon project [15]. Codon is a compiler and Domain Specific Language (DSL) framework that has a Pythonic base and uses Python's intuitive interface but can match the performance of languages such as C and C++. In this way, it forms a bridge between a large group of Python users while optimizing and improving on runtime.

While the above examples have in common that they focus on one or two particular examples of open source software, initiatives that have a broader scope have also been launched. The Free and Open Source Software (FOSS) Energy Efficiency Project (FEEP) [16], supported by KDE Eco, aims to collect energy consumption measurements of open source software, and to integrate measuring energy consumption into the development of open source software. They use criteria formulated by the label "Blauer Engel" [17], a product of the German ministery of Environment (Umweltbundesamt).

Lastly, the Green Software Foundation [18], an alliance of Microsoft, GitHub, the Linux Foundation, Accenture and more, hosts several projects that focus on green coding and minimizing energy consumption in the IT sector. Green coding is a programming practice aiming to create software systems that are as environmentally friendly as possible. The efforts range from providing databases with tools to make software more energy efficient, to developing software to measure energy consumption. They also provide education for software practitioners.

These already existing initiatives and others that have not been listed here show that there is engagement and attention for making open source software greener. However, the total effort will require a lot more involvement than what is currently the case. While the goals of the Open Source Software Efficiency Project overlap with the initiatives above, its scope aims to build a bridge between already existing efforts and solutions, as well as to come up with new ways to optimize open source software.

II. FINDING TARGETS TO IMPROVE

To efficiently allocate the resources of our project, we propose a set of strategies to identify the sections of a software project whose improvement would have the greatest impact on global efficiency. We use the language of complex networks, where different pieces of the software system (functions, modules, packages, etc.) are interpreted as nodes in a network and they are joined by vertices representing their call structure and dependencies. Fig. 2 shows an example of such a graph, where nodes of the graph are the 360 most used python libraries with some of them seeing several millions of downloads daily [19]. The vertices represent dependencies of these packages to other ones. Different centrality and clustering measures can then be used to find the most used or most connected nodes, to maximize the impact of optimizing a small amount of nodes. In the course of the weekend, we managed to identify the 10 most central Python packages according to various metrics (betweeness, Page Rank, etc). Among which, 2 are not implemented to work with the new, more efficient version of

Python. We suggest that these packages should be optimized in order to maximize the efficiency gains compared to efforts.

There are many instances in the stages of design, development, operation and use of software where optimization and improvements can be implemented according to [20], which can be extrapolated to improve the open-source software efficiency. The first way to improve the software is in its early stages, in particular in the design; changing the way different components interact with each other can vary significantly in the use of energy. As mentioned throughout the paper, in open-source coding these early stages are often overlooked and there is not a clear way as to how to measure the energy efficiency in this early stage. However, some improvement tactics could be implemented in the early stages, one of the most efficient is to identify and use *design patterns*, or recurrent structures which are called several times throughout the code as noted by [21] in the design of Android applications.

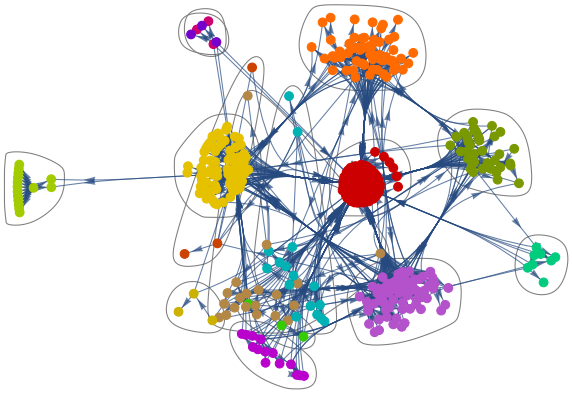

FIG. 2. Dependency network of the 360 most used Python modules [22], i.e., sets of nodes more interconnected among themselves than with the rest of the network.

One set of useful recommendations to which can be implemented are in Bitkom's manual [23] for resource-efficient programming, among them are compatibility, low hardware stress and the use of the right data structure which can be evaluated using the tool GreenC5 [24].

III. IMPROVEMENT STRATEGIES

We can break this section into **high level strategies** that can guide the general long term progress and **specific, actionable tactics** that can be implemented immediately even by individual developers.

A. High level strategies

There are many possible strategies to guide long term progress including the following examples.

First, if there is a choice of programming language to be used for a project, a first step would be to encourage the use of compiled languages like C++ or Rust (instead of interpreted languages like Python) for faster execution and lower run-time overhead [25] (see figure 3 for an actual distribution). This is in the case that the flexibility and convenience of a high-level language is not necessary. The second is educating people on energy-aware practices. In the same way that people are educating others about the harmful effects of bias in AI, one can also integrate energy aware coding practices into programming courses, and software development projects [26]. A perfect example of this is the previously mentioned Green Software Foundation. Another strategy to use is to offer incentives such as grants, awards, or recognition for projects that demonstrate significant improvements in efficiency. Examples of this are the Google Summer of Code (GSoC) Sustainable Computing Track and the Institute of Electrical and Electronics Engineers (IEEE) Green ICT Initiative.

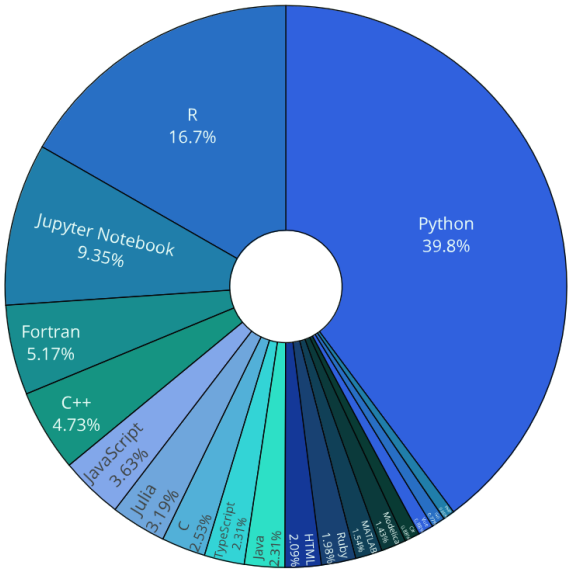

FIG. 3. Distribution of programming languages used in sustainability software [27]

B. Actionable tactics

Implementing green coding implies the adoption of good coding practices. For instance, using or enabling newer language versions or simply rewriting code in another language. These languages can be tested through various benchmarks such as binary-trees, fannkuch-redux, and fasta. In 2021, according to the Computer Language Benchmarks Game (CLBG), on average, C was the most efficient language in terms of energy consumption and execution time for compiled languages, while Fortran consumed on average 2.52 times more energy

and took 3.40 times longer to execute than C. Another interesting example is choosing the best loop statement between for, do-while, and while, which reduces CPU resources when running the same code lines multiple times[28].

To effectively incorporate green coding, developers can adopt several practices that can potentially enhance the efficiency of their code and contribute to reducing its environmental impact. Here are some of the steps that have been highlighted by the scientific literature:

Early decisions and benchmarking Considering energy efficiency early on helps avoid potential lock-ins down the line. The choice of both hardware and software and of the appropriate monitoring tools (such as websitecarbon[29] or ecograder[30] for websites, powerapi[31] or scaphandre[32] for software, or directly on premise with a power meter) can help gauge the current energy footprint and set benchmarks for improvement.

Integrating monitoring and optimization tools Performance benchmarking tools (e.g., bencher[33]) can be introduced, along with static code analysis tools (e.g., sonar[34], codiga[35]), to ensure that code quality does not degrade over time, leading to efficiency losses.

Platform as a service (PaaS) In some cases using PaaS can improve scalability and performance management, potentially lowering energy use[36, 37].

Algorithmic and data optimizations, hardcoding energy conservation Optimizations on the software side and on the data on which the software relies can lead to energy savings, as partly already specified in the high-level strategies. While some of these are heavily language- and data-specific, some adjustments could be made to adjust applications' behavior based on usage, such as reducing functionality or switching off non-critical features to conserve energy when possible [38, 39].

IV. PROOFS OF CONCEPT

During the Sciathon, two small proofs of concept were attempted.

A. Measuring Power Consumption

The ultimate outcome of performance improvement efforts is always a reduced power consumption. This is rather hard to measure as a single optimized process can not be completely isolated from other tasks an operating system is handling. Nevertheless, these measurements are crucial and during the Sciathon a small proof of concept was undertaken. The goal was to identify a computational task that is both widely used and easily benchmarkable in order to be able to achieve reliable numbers.

We chose to measure the power consumption of the serialization of Java objects to the BSON (Binary JSON) format as it represents a common task in programming.

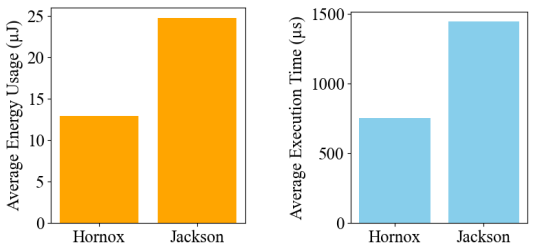

FIG. 4. Comparison of execution time and energy used per operation by the Hornox and Jackson implementations.

Two serialization libraries were benchmarked and their power usage was measured using Scaphandre [32] and a small Python evaluation script. The two libraries were Jackson [40], a widely used library with a lots of functionalities and Hornox [41], a purposefully simple library for serialization that focuses on speed rather than feature richness. By executing the provided benchmark of Hornox, a simple task of serializing a Java object was executed 1 million times, thereby enabling us to actually measure the energy consumption of an individual execution. The numbers are subject to multiple external variables (hardware, OS, Java version etc), but they are comparable within the configuration used (Lenovo T14s Gen3, Fedora 39, OpenJDK 17).

The energy consumed by an individual serialization operation is in the range of micro Joules on the test setup. Figure 4 illustrates how using a faster implementation (Hornox) compared to standard libraries (Jackson) can reduce the power consumption in the range of 50%.

B. Gains in Python performance

Given the recent efforts to optimize the CPython interpreter for greater performance, developers can with relatively little effort improve the performance of their Python libraries by making them compatible with the latest versions of the language. The latest stable release of Python, version 3.12, was released in October 2023. As of April 2024, nearly half (173) of the 360 most accessed packages on the PyPI repository are still not verified as being compatible with Python 3.12 [22]. As a practical demonstration of the performance gains that can be expected by updating packages for Python 3.12, we have carried out performance benchmarks of 87 Python libraries using the `pyperformance` suite [42], spanning a wide range of use cases from web tools to scientific packages, running on three recent versions of the Python interpreter (3.9, 3.11, and 3.12) with identical hardware. Running the same code on different versions of Python, we see a significant improvement in performance going from Python 3.9 to 3.11 and 3.12. Compared to the latter, Python 3.9 takes, on average, 37% longer to complete the same tasks (Figure 5). From 3.11 to 3.12, the

improvement is still significant at 6% on average. However, in certain areas, like in input/output that are particularly important for network and database tasks, the recent gains in performance are significantly larger with Python 3.12 being 2.04 times faster than 3.9 on average and 1.45 times faster than Python 3.11. With Python being a ubiquitous language in a plethora of areas, from internet backends to high-performance scientific computing, significant optimization gains can be achieved with relatively little effort simply by optimizing code for recent versions of the CPython interpreter.

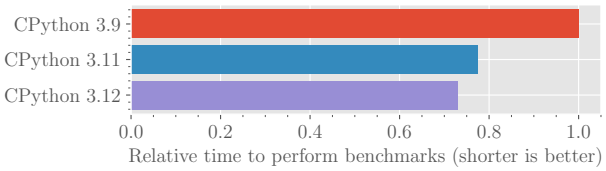

FIG. 5. Relative performance of CPython interpreters running the pyperformance benchmark suite shown as the geometric mean of time to complete the benchmark tasks.

V. WHAT'S NEXT?

The exploration of possible open source software to be targeted along with finding related efforts, possible improvement strategies and best practices is only the first step in the Open Source Software Efficiency Project. Measuring the power consumption in the example of the Hornox BSON and in the case study has shown potential, but for this project to be a success in the long run, funding possibilities and human resources will be necessary.

For financing, several possibilities exist for gathering donations and sponsorships, such as GitHub Sponsors [43], Open Collective [44], but also corporate sponsors. Another option is crowdfunding, as with Kickstarter [45], Indiegogo [46] or GoFundMe [47]. Finally, once the general idea of making open source software more energy efficient is broken down into more concrete individual efforts, grants and awards are available. Unfortunately, only a few funds support FOSS [48]. Out of those, most focus on new features (UKRI's Infrastructure Fund, UK [49], Horizon, Europe [50], Pathways to Enable Open-Source Ecosystems (POSE), US [51]). In recent years, new funding agencies popped up supporting sustainability in FOSS. Although here sustainability refers mostly to the improvement and maintenance of the software, one could argue that energy efficiency follows the same ideals and is eligible to apply. Relevant funds include the Sovereign Tech Fund [52], Apache Software Foundation [53], FOSS Sustainability Fund (Open Tech Fund) [54], NumFOCUS [55] and netidee [56].

Besides the financial aspect, it is crucial to have a community actually improving the FOSS code. Public outreach in the form of published articles, e.g. at Phoronix, Heise, HackerNews, QuantaMagazine both inform a wider audience of people about the need and impact of energy optimisation in code and it can attract further volunteers to join. Once we have a project team together, co-operations and partnerships with other groups following similar aims but with different approaches, like Open Source in Environmental Sustainability [27], could help both to gain further visibility and profit from each other's expertise. Projects might even be combined.

The long-term aim is to build a diverse team which can offer consulting and training to both commercial and open source software developers to make their code more energy-efficient. After having the experience of several projects being executed and the tools to measure the difference, this is a way to make many more developers aware of how to write their software in an energy-efficient way, ideally from the beginning on.

VI. ACKNOWLEDGEMENT

We thank the committee of the 4$^{\text{th}}$ Lindau Online Sciathon for the organisation and opportunity provided to participate with the Open Source Software Efficiency Project.

[1] Green IT Factsheet, University of Michigan, Center for Sustainable Systems **Pub. No. CSS09-07** (2023).
[2] B. Knowles, *ACM TechBrief: Computing and Climate Change*, Tech. Rep. (New York, NY, USA, 2021).
[3] H. Ferreboeuf, M. Efoui-Hess, and X. Verne, *Environmental impacts of digital technology: 5-year trends and 5G governance*, Tech. Rep. (The Shift Project, 2021).
[4] M. Morisio, P. Lago, N. Meyer, H. A. Müller, and G. Scanniello, 4th International Workshop on Green and Sustainable Software (GREENS 2015), in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2 (2015) pp. 981–982.
[5] F. Bals, *Open Source Security and Risk Analysis Report*, Tech. Rep. (Synopsis, 2024).
[6] R. Munroe, Dependency, `https://xkcd.com/2347/`.
[7] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, What do programmers know about software energy consumption?, IEEE Software **33**, 83 (2015).
[8] B. Goetz, *Java concurrency in practice* (Pearson Education, 2006).
[9] S. Abdulsalam, Z. Zong, Q. Gu, and M. Qiu, Using the greenup, powerup, and speedup metrics to evaluate software energy efficiency, in *2015 Sixth International Green and Sustainable Computing Conference (IGSC)* (IEEE, 2015) pp. 1–8.

[10] Statista, Number of available applications in the google play store from december 2009 to december 2023, https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-\play-store/, accessed on 21.04.2024.

[11] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, Green AI, Communications of the ACM **63**, 54 (2020).

[12] E. Strubell, A. Ganesh, and A. McCallum, Energy and policy considerations for deep learning in nlp, arXiv preprint arXiv:1906.02243 (2019).

[13] M. Shannon, Implementation plan for speeding up CPython, https://github.com/markshannon/faster-cpython/blob/master/plan.md, accessed on 21.04.2024.

[14] P. G. Salgado, What's new in python 3.11, https://docs.python.org/3/whatsnew/3.11.html, accessed on 21.04.2024.

[15] A. Shajii, G. Ramirez, H. Smajlović, J. Ray, B. Berger, S. Amarasinghe, and I. Numanagić, Codon: A compiler for high-performance pythonic applications and dsls, in *Proceedings of the 32nd ACM SIGPLAN International Conference on Compiler Construction*, CC 2023 (Association for Computing Machinery, New York, NY, USA, 2023) p. 191–202.

[16] K. Eco, Free and Open Source Software (FOSS) Energy Efficiency Project (FEEP), https://invent.kde.org/teams/eco/feep, accessed on 21.04.2024.

[17] J. Groeger, *Guide on Green Public Procurement of Software* (German Environment Agency, 2020).

[18] Green Software Solutions, Green Software, https://greensoftware.foundation/, accessed on 21.04.2024.

[19] Analytics for PyPI packages, https://pypistats.org/(), accessed on 21.04.2024.

[20] V. Majuntke and F. Obermaier, Towards sustainable software, in *INFORMATIK 2023 - Designing Futures: Zukünfte gestalten* (Gesellschaft für Informatik e.V., Bonn, 2023) pp. 149–164.

[21] C. Bunse and S. Stiemer, On the energy consumption of design patterns, Softwaretechnik-Trends **33**, 7 (2013).

[22] Python 3.12 Readiness, https://pyreadiness.org/3.12/, accessed on 21.04.2024.

[23] Bitkom, *Ressourceneffiziente Programmierung*, Tech. Rep. (Bundesverband Informationswirtschaft, 2021).

[24] J. Michanan, R. Dewri, and M. J. Rutherford, Greenc5: An adaptive, energy-aware collection for green software development, Sustainable Computing: Informatics and Systems **13**, 42 (2017).

[25] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva, Ranking programming languages by energy efficiency, Science of Computer Programming **205**, 102609 (2021).

[26] A.-M. Oprescu, I. Kokken, K. Maat, and F. de Geus, Introducing green thinking into cs bachelor curriculum, in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 2* (2023) pp. 667–667.

[27] T. Augspurger, E. Malliaraki, J. Hopkins, and D. Brown, *The Open Source Sustainability Ecosystem*, Tech. Rep. (The Linux Foundation, 2023).

[28] M. Al-Hashimi, M. Saleh, O. Abulnaja, and N. Aljabri, Evaluation of control loop statements power efficiency: An experimental study, in *2014 9th International Conference on Informatics and Systems* (IEEE, 2014) pp. PDC–45.

[29] Wholegrain Digital, Website carbon calculator, https://www.websitecarbon.com/, accessed on 21.04.2024.

[30] Mightybytes, Ecograder, https://www.ecograder.com/, accessed on 21.04.2024.

[31] Inria, University of Lille, Powerapi, https://github.com/powerapi-ng/powerapi (2024).

[32] Hubblo, Scaphandre, https://github.com/hubblo-org/scaphandre (2024).

[33] Bencher.dev, Bencher, https://github.com/bencherdev/bencher (2024).

[34] SonarSource, sonar, https://www.sonarsource.com/, accessed on 21.04.2024.

[35] codiga, codiga, https://www.codiga.io/, accessed on 21.04.2024.

[36] J. Weber and B. Rauch, Sustainable software design: Background and best practices, https://www.iese.fraunhofer.de/blog/sustainable-software-design/ (2023).

[37] X. Fan, W.-D. Weber, and L. A. Barroso, Power provisioning for a warehouse-sized computer, ACM SIGARCH computer architecture news **35**, 13 (2007).

[38] L. Ardito, G. Procaccianti, M. Torchiano, and A. Vetro, Understanding green software development: A conceptual framework, IT professional **17**, 44 (2015).

[39] Green Software Foundation, The carbon monkey, https://greensoftware.foundation/articles/the-carbon-monkey, accessed on 21.04.2024.

[40] Faster XML, Jackson, https://github.com/FasterXML/jackson (2024).

[41] Txture, Hornox BSON, https://github.com/Txture/hornox-bson (2022).

[42] The Python Performance Benchmark Suite, https://pyperformance.readthedocs.io (), accessed on 21.04.2024.

[43] GitHub, Github sponsors, https://github.com/sponsors, accessed on 21.04.2024.

[44] Open Collective, Open collective docs, https://opencollective.com/, accessed on 21.04.2024.

[45] Kickstarter, Bring a creative project to life, https://www.kickstarter.com/, accessed on 21.04.2024.

[46] Indiegogo, Crowdfund innovations & support entrepreneurs, https://www.indiegogo.com/, accessed on 21.04.2024.

[47] GoFundMe, Leading crowdfunding platform - your home for help, https://www.gofundme.com/, accessed on 21.04.2024.

[48] Deloitte Consulting and Offerman Consulting, *Development of a Funding Mechanism for Sustaining Open Source Software for European Public Services*, Tech. Rep. (European Commission, 2022).

[49] UK Research and Innovation, Ukri - what we do, https://www.ukri.org/what-we-do/, accessed on 21.04.2024.

[50] European Commission, Horizon europe, https://research-and-innovation.ec.europa.eu/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-europe_en, accessed on 21.04.2024.

[51] U.S. National Science Foundation, Pathways to enable open-source ecosystems (pose), https://new.nsf.gov/funding/opportunities/pathways-enable-open-source-ecosystems-pose, accessed on 21.04.2024.

[52] Sovereign Tech Fund, Applying for investment from the sovereign tech fund, https://www.sovereigntechfund.de/programs/applications, accessed on 21.04.2024.

[53] Apache Software Foundation, Software for the public good, https://www.apache.org/, accessed on 21.04.2024.

[54] Open Tech Fund, Free and open source software sustainability fund, https://www.opentech.fund/funds/free-and-open-source-software-sustainability-fund/, accessed on 21.04.2024.

[55] NumFOCUS, Numfocus project support, https://numfocus.org/projects-overview, accessed on 21.04.2024.

[56] netidee, Österreichs große Internet Förderaktion, https://www.netidee.at/, accessed on 21.04.2024.