



International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.935

Volume 15, Issue 8, August 2026

CarbonWise: An AI-Driven Framework for Carbon Footprint Estimation and Code Quality

Sapana Choudhari^{1*}, Divya Desai², Shruti Wankhede³, Mohit Salodkar⁴, Prathmesh Shukla⁵,
Swati Dronkar⁶

U.G. Student, Department of Computer Science Engineering, Priyadarshini College of Engineering, Nagpur,
Maharashtra, India

U.G. Student, Department of Computer Science Engineering, Priyadarshini College of Engineering, Nagpur,
Maharashtra, India

U.G. Student, Department of Computer Science Engineering, Priyadarshini College of Engineering, Nagpur,
Maharashtra, India

U.G. Student, Department of Computer Science Engineering, Priyadarshini College of Engineering, Nagpur,
Maharashtra, India

U.G. Student, Department of Computer Science Engineering, Priyadarshini College of Engineering, Nagpur,
Maharashtra, India

Assistant Professor, Department of Computer Science Engineering, Priyadarshini College of Engineering, Nagpur,
Maharashtra, India

ABSTRACT: The current software consists of tenable tools that mainly focuses on code quality, it enhance the energy footprint calculation which usually offers less support for extensive carbon footprint analyzation and AI-driven optimization. By recalling this gap the CarbonWise is being introduced it is an integrated AI-driven framework designed to advance and develop sustainable green software of engineering practices. CarbonWise unites multiple dimensions of greenDevOps sustainability assessment by combining and analyzing gitHub repository, carbon footprint estimation, code quality evaluation, and sustainability scoring within a same platform.

Carbonwise includes Green DevOps principles and utilize AI to generate optimization recommendations which are beign tailored to developer's workflows. Moreover CarbonWise will provide interactive,logical,analytic and automated sustainably evaluated reports, which enables teams to monitor, evaluate and improve the environmental impact of their software systems. By linking technical property metrics with environmental considerations, CarbonWise empowers developers to measure, understand and reduce impact enviornmentally to build energy-efficient and responsible software solutions thus contributing to the wider goals of carbon-perceptive computing and viable digital infrastructure.

KEYWORDS: Sustainable Software Engineering, Energy Efficiency,Green Software Code Quality Analysis, Green DevOps, Carbon-Aware Computing, Artificial Intelligence, Carbon Footprint Estimation.

I. INTRODUCTION

As the existing software applications are growing rapidly, AI applications and systems and all other cloud computing services has changed the view of the computerized ecosystem which is permitting the incomparable innovation and extensibility. Even though the rapid and extreme growth of digital technologies and systems has driven up the power requirements of computing structure and it is being contributed to an increase carbon emissions with the Constant spread of software applications and digital services, the intensified focus is now being given to the environmental footprint which is associated with software development and administration; deployment has become a critical and challenging aspect for modern software engineering. While confronting this challenge requires not only technical

power but also a prototype shift in the direction of carbon-aware computing and environmentally responsible development practices.

present approaches to software sustainability remain shattered. Many tools focus predominantly on code quality assessment, emphasizing maintainability, reliability, and performance of system while others concentrate on energy consumption or carbon estimation. While these methods may provide the valuable and important points they fail to hold and address the holistic nature of green practices in software engineering. Specifically they often lack in unification with developers' workflows and mainly offer narrow applicable guidance, leaving the professionals without a unified framework to develop, estimate and centralized the environmental impact of their projects. that is the gap which emphasize the need for the widespread solutions that combine professional quality metrics with ecological accountability, while also offering developer-centric insights and AI-driven recommendations.

To address these limitations, we propose CarbonWise, an AI-driven framework designed to embed sustainability into the core of software engineering practices. CarbonWise integrates multiple dimensions of sustainability analysis, including GitHub repository evaluation, carbon footprint estimation, code quality assessment, and sustainability scoring, within a single platform. By incorporating Green DevOps principles, the framework ensures that sustainability checkpoints are embedded into continuous integration and deployment pipelines, making ecological responsibility an integral part of modern development workflows. Furthermore, CarbonWise leverages machine learning models to provide intelligent optimization recommendations, guiding developers toward energy-efficient coding practices, resource-aware architectural decisions, and environmentally responsible deployment strategies.

II. RELATED WORK

Pathania et al. (2025) presented a comprehensive survey of 21 tools for calculating software energy use and carbon emissions, introducing a taxonomy of Monitoring, Estimation, and Black-Box approaches. Monitoring tools include hardware and software meters such as RAPL, Intel Power Gadget and nvidia-smi, which directly measure energy consumption. Estimation approaches rely on coefficients, similarity analysis, analytical models, or custom model training to approximate energy usage. Recorder approaches are patented calculators provided by cloud service providers, such as AWS's Customer Carbon Footprint Tool, Microsoft's emissions impact dashboard, and Google Cloud's Carbon Footprint, that offer aggregated insights but lack of transparency. The survey compared tools across attributes like granularity, component coverage, carbon estimation methods, and usability, resulting in a Tool Matrix for developers. A componentwise consolidation was also provided, mapping existing approaches to specific hardware domains.

Key challenges identified include: (1) lack of process-level visibility, as most tools operate at system-level granularity. (2) reliance on static averages for grid carbon intensity, despite daily fluctuations of over 70%. (3) neglect of embodied emissions, which increasingly dominate computing's carbon footprint. (4) dependence on approximate proxies like TDP, which often misrepresent actual energy use. The contribution of this work lies in offering a structured taxonomy and comparative framework that enables practitioners to select appropriate tools for their use cases, while highlighting research gaps such as real-time carbon intensity integration, embodied emissions accounting, and finer-grained process-level measurement.

Krishna Murthy et al. (2025) have introduced to Sustainable Green Coding as a structure cross-language framework to insert carbon-conscious practices directly into software development. Unlike traditional approaches that emphasize hardware efficiency or renewable energy, SGC mainly focuses on source level involvement such as algorithmic and infrastructural optimisations, and resource oriented programming strategies for AI/ML frameworks. The framework starts with core foundational analysis by using tools like Code Carbon, Carbon Tracker and Intel Power Gadget to measure CPU/GPU energy draw, memory usage, and carbon intensity. This analysis identifies bottlenecks such as inefficient loops, memory mismanagement or redundant I/O operations. Empirical evaluations demonstrated that deliberate code-level interventions can reduce energy consumption till 63% without compromising performance. Case studies included replacing bubble sort with (std::sort) in C++, loop fusion in C#, early termination in primality tests in Python, and inference caching in AI workflows to avoid redundant computations. Resource-conscious practices such as using StringBuilder instead of string concatenation in C#, reserving memory upfront in C++, batching API calls, and throttling background logging further reduced runtime overhead and emissions. For AI workloads, Green AI interventions like model pruning, quantisation, and efficient inference strategies were shown to lower computational

intensity while maintaining accuracy. The contribution of this work lies in reframing sustainability as a first-class software engineering objective, positioning carbon efficiency alongside performance and scalability.

Danushi et al. (2025) have introduced a literature review of 65 base studies on carbon-efficient software design and advancement, aiming to consolidate scattered research into a structured taxonomy. This analyzes the 5W1H framework (Who, What, Why, Where, When, How) which classifies current proposals. Stakeholders include developers, architects, IT operators, product owners, and sustainability engineers. The studies were grouped into four categories: sustainability guidelines (best practices and experience reports), reference models, measurement solutions (tooling and estimation techniques), and software improvement techniques. The review indicates that existing approaches mainly focus on assessing and lowering carbon emissions, with many relying on improvements in energy efficiency. However only a limited number of studies directly incorporate carbon awareness into the software design and development lifecycle. These approaches have been explored across various application areas, including cloud computing, blockchain, mobile platforms, and machine learning and HPC. Lifecycle stages addressed included requirements, design, implementation, testing, and rollout, though design and implementation were identified as the most critical leverage points. Key gaps highlighted were the lack of regularized metrics and methodologies, limited tool maturity and insufficient integration of sustainability into mainstream development pipelines. The authors concluded that future research should focus on sustainable-by-design approaches, embedding carbon efficiency into architectural decisions, data structures, and orchestration strategies, rather than treating sustainability as an afterthought.

Junger et al. (2024) presented an extended literature analysis on the Potentials of Green Coding, focusing on sustainable software development practices since 2010. The study addressed three research questions (1) does an environmentally friendly concept exist? (2) does practice are organized and deployed in industries ?(3) How can Green Coding be deployed into present processes and higher level of education curriculum?. The authors defined Green Coding as organizing, developing, structure and utilize software systems in ways that reduce energy and resource consumption. Their expertise showed that while physical level sustainability has long been studied, software-induced power consumption that has only gained attention in the past few years mainly after 2015. Reports highlights the practices such as energy-aware coding guidelines, byte-code transformations, code clouding, and containerization, noting that these can significantly affect energy efficiency. Case studies revealed both positive and negative impacts of such techniques, underscoring the complexity of software sustainability. The study emphasized the lack of formal standards for sustainable software development, though frameworks like Greensoft and sustainability labels have been proposed. Importantly, the authors advocated embedding sustainability as a non-functional requirement in software processes and curricula. They argued that since over 75% of developers have higher education backgrounds, integrating Green Coding into academic programs is crucial for shaping future professionals. The contribution of this work lies in bridging industry practices, academic research, and education, offering orientation for stakeholders and stressing that sustainable software engineering must evolve from isolated experiments into mainstream development and training.

Castor (2024) provided a primer on estimating the energy footprint of software systems, distinguishing between two main approaches: measurement and modeling. Measurement involves using external equipment or processor-level counters to directly quantify energy consumption. This method is non-invasive and precise but often limited to device- or component-level granularity, making it difficult to attribute energy use to specific software elements. Representing this on the other hand uses analytical or pretense based techniques to assess power consumption at superior levels such as methods, data structures and also lines of code. Though conceptual modeling is valuable testing continues after deployment and offers versatility it endures from reduced accuracy due to simplifying assumptions and latent latency from instrumentation. The paper explained these trade-offs through an experiment using the Fannkuch Redux benchmark on a MacBook Air (M1 chip). Results showed that execution consumed 64.835 J per run, with CPU energy usage more than 140 times higher under load compared to idle states, reinforcing the principle of “race to idle.” The study also highlighted how sampling rates affect accuracy: a 10 Hz rate produced stable results, while a 1 kHz rate introduced overhead, increasing energy consumption by 5.6%. Castor outlined three concrete approaches: (1) hardware performance counters combined with software tools, (2) specialized external measurement hardware, and (3) analytical models. The contribution of this work lies in offering a structured methodology for experiment design, workload characterization, and reproducibility, making it a practical guide for researchers and practitioners in green software engineering.

Radovanović et al (2023) have introduced to Google's Carbon Intelligent Compute Management the first global, in-production, carbon-aware scheduling system for datacenters. The system directs the problem where the carbon intensity of electricity varies significantly by time and zone means datacenters can unintentionally when workload scheduling does not account for changes in grid carbon intensity, the same computing tasks may result in higher CO₂ emissions. CICS uses Virtual Capacity Curves to handle the amount of computing resources assigned to flexible workloads during different hours while organizing the required daily capacity. As a result non-urgent tasks, including machine learning model training, data compaction, and video processing, can be shifted to periods when electricity generation is relatively cleaner. This scheduling strategy reduces the environmental impact without disrupting essential user-facing applications such as Search, Maps, and YouTube.

The methodology integrates several analytical pipelines: (1) carbon supervised estimation from framework data (2) Mathematical power designs plotting CPU usage to power consumption all around the heterogeneous hardware, (3) forecasting the load for flexible and inflexible workloads, and (4) risk-aware optimization that balances environmental goals with service-level objectives and infrastructure constraints. A monitoring and feedback loop ensures reliability and adapts to workload changes. Evaluation across 20+ datacenters on four continents, consuming more than 15.5 TWh annually, demonstrated that VCCs effectively reduce peak CPU and power usage while shifting workloads to low-carbon periods. Compared to prior research, which relied on theoretical models or small-scale prototypes, CICS represents the first hyperscale, realworld deployment of carbon-aware scheduling. Its strengths lie in scalability, modularity, and independence from real-time schedulers, while its limitation is applicability only to temporally flexible workloads. Nonetheless, it sets a benchmark for industry-wide adoption of carbon-aware computing, showing that datacenter operations can simultaneously reduce emissions, improve efficiency, and lower costs.

Verdecchia et al. (2021) have discussed Green IT and Green Software as a paradigm shift in software engineering, emphasizing the urgent need to reduce the ecological footprint of IT systems, which already consume about 10% of global electricity and are projected to reach one-third by 2030. The authors proposed a temporal horizon framework: H1 (shortterm) solutions such as cloud migration, renewable energy adoption, shutting down idle servers, and energy-aware optimizations; H2 (near-future) solutions including distributed energy landscapes, federated learning, affordable AI chips, and lifecycle management practices; and H3 (long-term) breakthroughs like integrated photonics, neuromorphic computing, and novel non-von Neumann architectures. The paper gives practical guidance for practitioners by highlighting strategies such as efficient data management, serverless computing, and Rooting versatility in software design. It also highlights the importance of user experience in power consumption, promoting flexible features like reduced resolution modes and energy usage indicators. Additionally, the authors warned against energy-intensive technologies such as blockchain, advocating alternative consensus mechanisms. The contribution of this work lies in offering practical, multi-level strategies for developers, testers, and IT administrators to embed sustainability into everyday practice, while framing ecology not as a cost but as a business opportunity. By combining technical, social, and paradigm-shift solutions, the paper positions green IT as the next major wave of innovation after digital transformation.

Lannelongue et al. (2021) introduced Green Algorithms, a methodological framework and freely available online tool (www.green-algorithms.org) to estimate the carbon footprint of computational tasks. The framework considers hardware requirements, runtime, data centre efficiency (PUE), and regional carbon intensity (CI), and contextualises emissions by equating them to car travel, flights, or tree sequestration. Case studies demonstrated its applicability across diverse domains: particle physics simulations (GEANT4-DNA) emitted $\approx 544,115$ gCO_{2e}, equivalent to a transcontinental flight; weather forecasting models (ICON and IFS) emitted ≈ 2.3 million gCO_{2e} and $\approx 298,915$ gCO_{2e} per day, respectively; and NLP models such as BERT and Meena training emitted $\approx 754,407$ gCO_{2e} and ≈ 164 million gCO_{2e}, respectively. The study highlighted that parallelization does not always reduce emissions—beyond an optimal number of cores, energy costs outweigh runtime gains. It also showed how relocating data centres can paradoxically increase emissions if regional CI is higher, even when PUE improves. Limitations include reliance on average CI values, exclusion of hardware lifecycle impacts, and assumptions about usage factors. The contribution of this work lies in providing a transparent, generalizable, and easy-to-use tool that enables scientists to quantify and report emissions consistently, raising awareness of the environmental costs of computation and offering practical recommendations such as algorithm optimization, reducing redundant runs (lowering the pragmatic scaling factor), and mindful choice of data centre location.

Ergasheva et al. (2020) carried out a systematic review of existing studies focused on measuring energy consumption in software systems. Out of 504 identified studies, 124 were selected for an in-depth analysis. The review resulted in the identification of 169 different energy-related metrics that can be applied across mobile, embedded, and cloud-based computing environments. The metrics were classified into six categories: hard metrics (physical measurements), code metrics (source-level analysis such as cyclomatic complexity, loops, coupling), run-time metrics (dynamic analysis of bytecode and execution), indirect metrics (model-based estimations), process metrics (development process analysis), and others (specific operational measures). The review revealed that mobile systems dominated research (49.2%), followed by embedded (27.5%) and cloud systems (23.3%), highlighting the immaturity of energy efficiency studies in cloud and embedded domains. The authors emphasised that while hardware-level metrics are well explored, software-level process metrics remain underdeveloped, with limited tools for non-invasive monitoring during development. Several tools were identified, including PETrA (Android power estimation), JADE (energy-aware offloading), GEMMA, GreenAdvisor (system-call profiling), and ANEPROF (function-level power distribution). The study concluded that current approaches are fragmented, with no evolutionary framework to assess energy efficiency across the entire software lifecycle. The authors called for the development of integrated frameworks combining developer behaviour patterns with code metrics, enabling precise in-situ analysis of energy efficiency. Their contribution lies in mapping the landscape of metrics, identifying gaps, and stressing the need for standardized, developer-friendly tools to insert stability into software engineering practice.

III. CONCLUSION

CarbonWise focus to project an ideal improvement and development in sustainable software engineering by integrating GitHub repository analysis, carbon footprint estimation, code quality testing and sustainability scoring into a unified AI-driven platform. Unlike fragmented existing approaches, CarbonWise embeds Green DevOps practices directly into development pipelines, ensuring that ecological responsibility becomes a core design principle rather than a peripheral concern. Through its machine learning-powered optimization recommendations, interactive analytics dashboards, and automated sustainability reports, the framework empowers developers to adopt energy-efficient coding practices, resource-aware architectures, and environmentally responsible deployment strategies. By bridging technical excellence with ecological accountability, CarbonWise not only supports the creation of high-quality, carbon-aware software systems but also contributes to the broader vision of sustainable digital infrastructure, demonstrating the transformative potential of AI in addressing global climate challenges.

IV. RESEARCH GAP

- Existing studies are fragmented, focusing on isolated aspects such as energy consumption, code optimization, or carbon-aware computing.
- There is no unified framework that integrates multiple sustainability dimensions into a single solution.
- GitHub repository analysis is rarely incorporated, limiting seamless integration with developer workflows.
- AI-driven recommendations for sustainability optimization are largely absent, leaving developers without actionable guidance.
- Automated sustainability reporting is underdeveloped, reducing transparency and organizational accountability.
- Tools provide limited developer-centric insights, often presenting raw data without translating it into practical strategies.
- Green DevOps practices are insufficiently supported, with few solutions embedding sustainability checkpoints into CI/CD pipelines.
- Existing approaches show limited scalability across diverse projects, cloud environments, and AI-driven applications.
- There is a disconnect between technical sustainability metrics and organizational carbon reduction goals, restricting alignment with broader ecological strategies.

REFERENCES

- [1] O. Danushi, S. Forti, and J. Soldani, “Carbon-efficient software design and development: A systematic literature review,” *ACM Computing Surveys*, 2025.
- [2] S. M. K. Murthy, U. Airsang, K. Rajamani, and B. M. Schuller, “From code to carbon: Being energy efficient while coding,” in *Proc. Int. Conf. Sustainable & Resilient Futures (ICSRF)*, 2025.
- [3] Verdecchia et al. discuss how software engineering practices can be considered from an environmental sustainability perspective, particularly in relation to the energy and resource demands of software systems.
- [4] Castor presents an introductory approach for understanding and assessing the energy consumption associated with software execution.
- [5] S. Podder, H. Date, and S. Murthy Green prompt engineering for sustainable generative AI, *Environmental Science and Ecotechnology*, vol. 30, 2026.
- [6] J. Rashid, T. Mahmood, and M. W. Nisar A study on software metrics and its impact on software quality, *Technical Journal, University of Engineering and Technology (UET) Taxila*, vol. 24, no. 1, 2019.
- [7] R. Baghdadi, M. Merouani, M. H. Leghettas, K. Benatchba, S. Amarasinghe, and T. Arbaoui, A deep learning-based cost model for automatic code optimization, in *Proc. 4th MLSys Conference*, 2021.
- [8] S. Ergasheva, I. Khomyakov, A. Kruglov, and G. Scucci, Metrics of energy consumption in software systems: A systematic literature review, *IOP Conference Series: Earth and Environmental Science*, 2020.
- [9] Y. G. Kim, U. Gupta, A. McCrabb, Y. Son, V. Bertacco, D. Brooks, and C.-J. Wu, GreenScale: Carbon-aware systems for edge computing, *arXiv preprint arXiv:2304.00404*, 2023.
- [10] A. Radovanović, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, S. Talukdar, E. Mullen, K. Smith, M. Cottman, and W. Cirne, Carbon-aware computing for datacenters.
- [11] S. S. Shenoy and R. Eeratta, Green software development model: An approach towards sustainable software development in *Proc. 2011 Annual IEEE India Conference (INDICON)*, Hyderabad, India.
- [12] Sehgal, R., Mehrotra, D., Nagpal, R., & Sharma, R. (2022). Green software: Refactoring approach. *Journal of King Saud University – Computer and Information Sciences*, 34(10), 4635–4643.
- [13] Gupta, U., Kim, Y. G., Lee, S., Tse, J., Lee, Wei, G. Y., Brooks, D., & Wu, C. J. (2020). Chasing carbon: The elusive environmental footprint of computing.
- [14] Kern, E., Hilty, L. M., Guldner, A., Maksimov, Y. V., Filler, A., Groger, J., & Naumann, S. (2018). Sustainable software products – Towards assessment criteria for resource and energy efficiency. *Future Generation Computer Systems*, 86, 199–210.
- [15] Pathania, P., Bamby, N., Mehra, R., Sikand, S., Sharma, V. S., Kaulgud, V., Podder, S., & Burden, A. P. (2025). Calculating software’s energy use and carbon emissions: A survey of the state of art, challenges, and the way ahead.
- [16] Amsel, N., Ibrahim, Z., Malik, A., & Tomlinson, B. (2011). Toward sustainable software engineering (NIER track). In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)* (pp. 976–979). ACM.
- [17] Capra, E., Francalanci, C., & Slaughter, S. A. (2012). Is software “green”? Application development environments and energy efficiency in open source applications. *Information and Software Technology*, 54(1), 60–71.
- [18] Ournani, Z., Rust, P., Rouvoy, R., & Penhoat, J. (2020). On reducing the energy consumption of software: From hurdles to requirements. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '20)* (pp. 1–12). ACM.
- [19] Hindle, A. (2016). Green software engineering: The curse of methodology. *Proceedings of the 2016 IEEE/ACM International Conference on Software Engineering Companion (ICSE-C)*, 507–510. IEEE.
- [20] Guldner, A., Bender, R., Calero, C., Fernando, G. S., Funke, M., Gröger, J., Hilty, L. M., Hörschemeyer, J., Hoffmann, G.-D., Junger, D., Kennes, T., Kreten, S., Lago, P., Mai, F., Malavolta, I., Murach, J., Obergöcker, K., Schmidt, B., Tarara, A., Naumann, S. (2024). Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components – Green Software Measurement Model (GSMM). *Future Generation Computer Systems*, 155, 402–418.



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

 9940 572 462  6381 907 438  ijirset@gmail.com



www.ijirset.com

Scan to save the contact details