

# Measuring Code Level Determinants of Software Sustainability: A Systematic Literature Review

Zoya Mahboob

National University of Computer and Emerging Sciences

Atif Jilani

[atif.jilani@nu.edu.pk](mailto:atif.jilani@nu.edu.pk)

National University of Computer and Emerging Sciences

Qurrat Ul Ain

National University of Computer and Emerging Sciences

---

## Research Article

**Keywords:** Sustainable Software Engineering, Green Software Development, Code Level Determinants, Energy Efficiency, Carbon Emissions, Systematic Literature Review

**Posted Date:** October 22nd, 2026

**DOI:** <https://doi.org/10.21203/rs.3.rs-10357156/v1>

**License:**   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

**Additional Declarations:** No competing interests reported.

---

# Measuring Code Level Determinants of Software Sustainability: A Systematic Literature Review

Zoya Mahboob<sup>1</sup>, Atif Aftab Ahmed Jilani<sup>1\*</sup>, Qurrat Ul Ain<sup>1</sup>

<sup>1\*</sup>NCRA UAV Dependability Lab, Department of Software Engineering,  
National University of Computer and Emerging Sciences, Islamabad,  
Pakistan.

\*Corresponding author(s). E-mail(s): [atif.jilani@nu.edu.pk](mailto:atif.jilani@nu.edu.pk);  
Contributing authors: [zoya.mahboob@isb.nu.edu.pk](mailto:zoya.mahboob@isb.nu.edu.pk);  
[quratul.ain.v@isb.nu.edu.pk](mailto:quratul.ain.v@isb.nu.edu.pk);

## Abstract

The energy a software system consumes and the carbon emitted in supplying it are shaped directly by decisions made at the level of code. Studies of these effects have multiplied, yet their evidence remains scattered, and the field lacks a consolidated account of the determinants developers control, the metrics that quantify their impact, and how the two relate. To provide such an account, we conducted a systematic literature review of 137 peer reviewed studies published between December 2015 and January 2026, retrieved from nine academic databases and extended through backward and forward snowballing. From the included studies we derived a taxonomy that organises code level determinants and impact metrics and maps them onto the implementation, testing, and runtime phases of development. We then analysed the evidence for each metric, tracing it to the studies that report it and their findings. This synthesis showed that energy is measured in 131 studies while carbon is measured directly in only three. Building on our findings, we proposed a literature grounded decision framework and a compact set of actionable guidelines for developers. The review, thus, offers both a consolidated map of code level sustainability research and a practical basis for building greener software.

**Keywords:** Sustainable Software Engineering, Green Software Development, Code Level Determinants, Energy Efficiency, Carbon Emissions, Systematic Literature Review

# 1 Introduction

Software now underpins large scale digital infrastructure, from cloud services to mobile and embedded systems. This reach, however, comes at a considerable environmental cost. Global data centres consume approximately 500 terawatt hours of electricity annually, a figure that is projected to grow as artificial intelligence workloads, streaming services, cloud computing, and connected devices continue to expand ([International Energy Agency 2025](#)). Beyond the data centre, the software running on countless personal and embedded devices collectively places a substantial further demand on global electrical infrastructure. Estimates place the information and communications technology sector’s share of global greenhouse gas emissions at a level comparable to that of the aviation industry ([Freitag et al. 2021](#)), and unlike many other sectors, software can address a meaningful share of this footprint through deliberate engineering decisions.

Software design and implementation choices directly influence the energy consumed by this infrastructure. The way code is written, the algorithms developers choose, the programming languages they employ, and the data structures they rely upon all shape how much energy a software system consumes, and by extension, its environmental impact and carbon footprint. A program written in Python may consume twenty times more energy than its equivalent in C ([Pereira et al. 2017a](#)), and a memory access pattern that lacks spatial locality may incur substantial cache miss penalties that translate into wasted energy. Prior empirical studies have shown that such choices can measurably affect energy consumption. Yet, this evidence remains poorly integrated into mainstream software engineering practice and education.

Recognising this reality, the field of sustainable software engineering has emerged over the past decade to investigate how software can be developed, operated, and maintained in ways that reduce its environmental footprint without sacrificing functionality, quality, or performance. This emergence has been catalysed by multiple concurrent developments including the 2015 Sustainable Development Goals adopted by the United Nations, the establishment of the IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems, the founding of the Green Software Foundation in 2021 by major technology companies (including Microsoft, GitHub, Accenture, and others), and increasingly stringent regulatory frameworks such as the EU Green Deal, the EU Corporate Sustainability Reporting Directive (CSRD), and national carbon accounting mandates. Major technology companies including Google, Microsoft, and Amazon have committed to achieving carbon neutrality or carbon negativity, placing pressure on their internal software development practices to demonstrate measurable progress toward green software development.

In response, academia has approached the sustainability challenge from multiple angles. Some have studied sustainability at the architectural and systems level, examining how software systems are deployed and operated. Others have studied it at the process level, investigating how development methodologies, DevOps practices, and testing strategies can be made more sustainable. More crucially, a substantial body of work has studied sustainability at the level of the code itself, examining the concrete, implementable properties of source code that govern how much energy a system consumes. It is this last vantage point that has proven both productive and fragmented in

the literature. Studies have examined how loops are written and optimised (Procacianti et al. 2016), how algorithms are selected and their energy implications compared (Pereira et al. 2017a), how refactoring changes energy consumption (Gottschalk et al. 2016), how programming language choice shapes power use across orders of magnitude (Pereira et al. 2017a), and how data structure selection affects both cache behaviour and energy efficiency (Pereira et al. 2023). Yet this evidence remains scattered, and the determinants and impact metrics through which these effects are studied and quantified have never been collected, organised, and synthesised into a single coherent account that lets developers and researchers obtain a consolidated view of the evidence.

This gap is consequential in several ways. Developers, to begin with, lack actionable guidance at the very point where their decisions are most tractable, namely when the code is being written, because there is little structured indicative evidence of the determinants that actually make a difference to the net consumption of resources of software. Researchers face a related problem. Absent a mapping of those determinants to the phases of development, it is hard to say where intervention would be most impactful. There is also no clear census of the impact metrics the community uses in practice, and without one, judging the comparability of findings is difficult, measurement gaps cannot be exposed, and standardisation cannot sensibly be called for. Underlying all of this is a lack of connection between a commonly observed sustainability symptom and the determinant most likely to be its cause. Taken together, these shortcomings motivate a review organised around code level determinants and their impact metrics.

In this paper we distinguish two kinds of quantifiable properties used in the literature to characterise the environmental impact of code. *Impact metrics* are the dependent variables through which that impact is measured, either directly (such as energy consumed or carbon emitted) or by proxy (such as execution time). *Code level determinants* are the independent variables that developers choose or influence (such as the choice of programming language or algorithm) and whose effect on those impact metrics has been measured. Our review synthesises 137 peer reviewed studies published between December 2015 and January 2026, retrieved from nine academic databases (ACM Digital Library, IEEE Xplore, Google Scholar, Springer Link, ScienceDirect, Semantic Scholar, Scopus, Wiley Online Library, and Web of Science) and augmented by backward and forward snowballing, to answer five research questions spanning demographic analysis, metric identification, development phase mapping, impact metric characterisation, and future research directions.

Systematic reviews adjacent to this topic do exist, and our work is positioned deliberately against them. Recent reviews have examined carbon efficient design and development (Danushi et al. 2025), the energy impact of design patterns, code smells, and refactoring (Poy et al. 2025), the influence of software design on energy performance (Connolly Bree and Ó Cinnéide 2025), and the tooling used to evaluate the energy consumption of machine learning (Rodriguez et al. 2024). Most recently, a conceptual overview has revisited green source code and the sustainability of software output (Toosi 2026), and a research roadmap has charted the augmentation of software engineering processes and products with generative AI (Amalfitano et al. 2026).

Each is valuable, but each is also bounded. Some restrict attention to a single family of code constructs, some operate at the design rather than the code level, some target a single application domain, and some remain conceptual or engage sustainability only peripherally. None organises code level determinants and impact metrics into a taxonomy, maps them to development phases, and couples the synthesis with a practitioner facing decision aid. Table 1 summarises this comparison and situates our review with respect to scope, granularity, and contribution. We engage these reviews and build on their foundations, while noting that the consolidated, metric centred, and developer oriented synthesis we provide is, to our knowledge, not yet available.

**Table 1** Comparison of this review with recent related reviews, overviews, and roadmaps in software engineering (from 2024 to 2026). CL = code level granularity, TAX = provides a taxonomy, PHASE = maps to development phases, DEC = provides a decision aid or actionable guidelines for practitioners.

| Review                              | Year             | Primary focus                                      | CL         | TAX        | PHASE      | DEC        |
|-------------------------------------|------------------|----------------------------------------------------|------------|------------|------------|------------|
| Toosi (2026)                        | January 2026     | Green source code and sustainable output           | Yes        | Partial    | No         | Partial    |
| Amalfitano et al. (2026)            | January 2026     | Generative AI augmentation of software engineering | Partial    | Yes        | Yes        | No         |
| Danushi et al. (2025)               | 2025             | Carbon efficient design and development            | Partial    | Yes        | Yes        | Partial    |
| Poy et al. (2025)                   | 2025             | Design patterns, code smells, refactoring          | Yes        | No         | No         | Partial    |
| Connolly Bree and Ó Cinnéide (2025) | 2025             | Software design and energy performance             | Partial    | No         | No         | No         |
| Rodriguez et al. (2024)             | 2024             | Energy evaluation tooling for machine learning     | Partial    | Partial    | No         | No         |
| <b>This review</b>                  | <b>July 2026</b> | <b>Code level determinants and impact metrics</b>  | <b>Yes</b> | <b>Yes</b> | <b>Yes</b> | <b>Yes</b> |

This paper makes three contributions. The first is a taxonomy of the metrics and determinants of code level sustainability, grounded in the 137 included studies, organised by what each metric quantifies, and mapped onto the implementation, testing, and runtime phases of development. The second is a per metric evidence synthesis that records, for every metric, the exact studies reporting it and the analytical substance of their findings, and that exposes a pronounced imbalance between the almost ubiquitous measurement of energy and the rare measurement of carbon, a gap whose significance grows as carbon accounting regulation matures. The third translates that synthesis into practice by providing a literature grounded decision framework, presented as a flowchart, alongside a compact set of actionable developer guidelines and explicit implications for researchers and industry practitioners.

The remainder of this paper is structured as follows. Section 2 provides background on sustainability in software engineering and presents the taxonomy of code level determinants and impact metrics, defining and exemplifying each. Section 3 details our systematic review methodology in accordance with the guidelines of Kitchenham and Charters (Kitchenham and Charters 2007), including the search process, inclusion and exclusion criteria, the PRISMA flow diagram, the quality assessment framework, and the data extraction procedure. Section 4 presents the results and discussion for each research question. Section 5 presents the decision framework and flowchart for practitioners. Section 6 presents the per metric evidence synthesis. Section 7 presents the developer guidelines. Section 8 presents the implications for researchers and industry professionals. Section 9 addresses threats to validity, and Section 10 concludes this review. The complete list of included studies, together with their full bibliographic references, is provided as supplementary material in Supplementary Material (SLR on Sustainability of Software), and is also available at the given repository link.

## 2 Background

### 2.1 Sustainability in Software Engineering

The notion of sustainability has long been anchored in the Brundtland Report of 1987, which defined sustainable development as development that “*meets the needs of the present without compromising the ability of future generations to meet their own needs*” (Brundtland et al. 1987). This foundational definition emerged from deliberations by the World Commission on Environment and Development and has shaped environmental policy discourse for nearly four decades. The Brundtland Report’s conceptualization of sustainability as inter generational equity, the idea that present development choices should not preclude future generations from enjoying comparable quality of life and access to natural resources, provides a powerful ethical foundation for the field of sustainable software engineering.

The application of the Brundtland framework to software engineering requires translating abstract inter generational concerns into concrete technical properties and metrics. In the context of software engineering, a universally accepted definition of sustainability has not yet emerged, though the literature converges on several recurring framings. Penzenstadler (2013) characterises software sustainability as “*preserving the function of a system over a defined time span*”, emphasising continuity, resilience, and long term utility as hallmarks of a sustainable software system. A more expansive and multi dimensional perspective is offered by Lago et al. (2015), who decompose software sustainability into four interacting quality dimensions. The *Environmental* (E) dimension addresses the software system’s impact on natural resources, ecosystems, and planetary boundaries. The *Economic* (EC) dimension concerns the financial viability and cost effectiveness of developing, maintaining, and operating the software over time. The *Social* (S) dimension encompasses the software’s impact on human well being, equity, accessibility, and social justice. The *Technical* (T) dimension relates to the system’s internal quality attributes such as maintainability, modifiability, testability, and compatibility with evolving technological landscapes.

These four dimensions interact synergistically. Technical debt may lead to increased maintenance effort and energy consumption, which in turn increases operational costs and may marginalise communities unable to afford high energy software. Within this four dimensional framework, the determinants and impact metrics addressed in this review have implications across all four dimensions, though our focus here is primarily on the Environmental dimension.

Lastly, an important recent development in the field is the Software Carbon Intensity (SCI) specification, developed by the Green Software Foundation in collaboration with the Linux Foundation and standardised as ISO/IEC 21031:2024 (Green Software Foundation 2024). The SCI formula quantifies software carbon intensity as  $SCI = (E \times I) + M$ , where  $E$  represents energy consumed (measured in kilowatt hours),  $I$  represents the carbon intensity of the electricity grid (in grams of CO<sub>2</sub> per kilowatt hour, which varies by region and time), and  $M$  represents embodied emissions (carbon embedded in the manufacture of hardware). The SCI specification is significant for this review because it makes carbon, rather than energy alone, the unit of account, and because it expresses carbon as a function of an energy term that the bulk of the code level literature already measures. As we show in Section 4, the literature reports energy almost universally but carbon only rarely, and the SCI formula is precisely the bridge that would let the field convert its existing energy evidence into the carbon terms that regulators increasingly demand (Danushi et al. 2025).

For the purposes of this systematic literature review, we adopt a focused operational definition grounded in the environmental dimension while remaining aware of these interdependencies. We define *sustainable software engineering* as the discipline concerned with the *technical, code level properties of software that determine a software system’s environmental impact*, particularly in terms of energy consumption and carbon emissions. This definition is grounded in the Brundtland Report’s inter generational equity framework and is consistent with the direction taken by influential researchers such as Patricia Lago (Lago et al. 2015) and Birgit Penzenstadler (Penzenstadler 2013), while deliberately narrowing the scope to code level properties so as to make the review directly actionable for software developers and researchers.

## 2.2 Code level Determinants and Impact Metrics of Software Sustainability

A recurring difficulty in the code level sustainability literature is that the properties through which environmental impact is quantified are heterogeneous in kind. Some are direct measures of consumption, such as energy in joules. Others are temporal proxies, such as execution time. Many more are code level determinants whose effect on consumption has been measured, such as the choice of programming language or the presence of a code smell. Treating these as a flat, undifferentiated list, obscures the relationships among them and makes the body of evidence harder to navigate. To remedy this, we organise the properties identified across the 137 included studies into a taxonomy with two top level groups, structured by the role each entry plays in the evidence. *Impact metrics* are the dependent variables through which environmental impact is measured, and *code level determinants* are the independent variables

that developers choose or influence and whose effect on those metrics has been measured. The taxonomy comprises six categories beneath these two groups, shown in Figure 1. The impact metrics group contains the consumption metrics and the temporal and performance metrics, while the code level determinants group contains the code structural determinants, the code quality and anti pattern determinants, the machine learning specific determinants, and an other category for emerging implementation specific determinants. Throughout the rest of the paper we refer to individual determinants and impact metrics by their category in this taxonomy. For example, energy usage is a consumption metric as defined in this section and depicted in Figure 1.

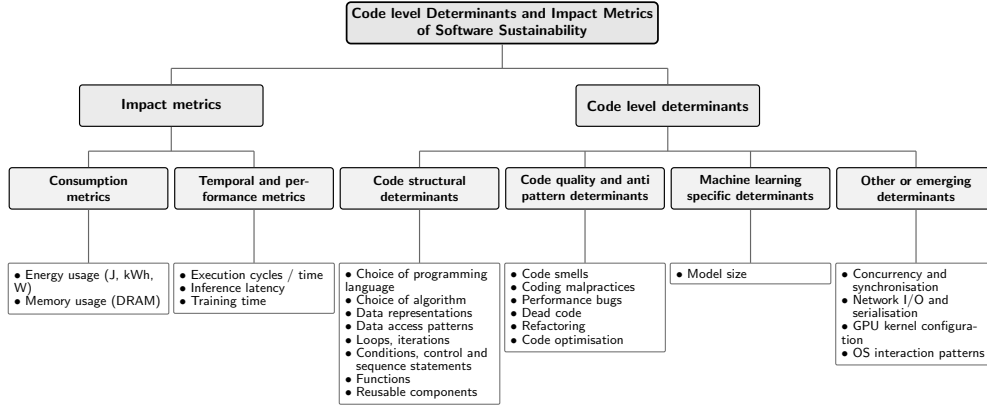


**Fig. 1** Taxonomy of the code level determinants and impact metrics of software sustainability, derived from the 137 included studies. The impact metrics (the consumption and temporal categories) measure environmental impact directly or by proxy, while the code level determinants (the structural, quality and anti pattern, machine learning specific, and other categories) are code level properties whose effect on that impact has been measured. Inference latency and training time are temporal impact metrics that recur specifically in machine learning contexts.

The remainder of this subsection defines each determinant and impact metric and grounds it in concrete findings from the included studies, organised by the six categories of the taxonomy.

### ***Consumption metrics.***

These metrics measure the resources a system consumes during operation and are the most direct expression of environmental impact at the code level. *Energy usage* refers to the total electrical energy consumed by a software system during a defined period of operation, typically reported in joules, kilowatt hours, or watts. It is the single most pervasive metric across the studies, and it is most often measured using hardware performance counters such as Intel’s Running Average Power Limit (RAPL) interface, which has become the de facto standard because it offers fine granularity, low overhead, and broad hardware support. [Maquoi et al. \(2025\)](#) link RAPL measured consumption to individual lines of code through unit test execution, while [Pereira et al. \(2016\)](#) use RAPL to localise energy inefficient fragments. *Memory usage* refers to the memory consumed during execution. Although not synonymous with energy, the two

are strongly correlated in DRAM intensive workloads because DRAM access consumes considerably more energy per bit than on chip cache access, a relationship quantified directly in the cross language study of [Pereira et al. \(2017a\)](#) and the collections study of [Hasan et al. \(2016\)](#).

#### ***Temporal and performance metrics.***

These metrics quantify the time a computation takes and serve as proxies for, or determinants of, energy. *Execution cycles*, or execution time, represent the computational effort expended to complete a task. Since energy is the integral of power over time, execution time is a strong, if imperfect, proxy for energy, as [Cunha et al. \(2024\)](#) demonstrate when they show that runtime and energy can diverge under power caps, and as [Georgiou et al. \(2018\)](#) examine through the energy delay product. *Inference latency* is the time a machine learning model takes to produce a prediction, which directly drives the energy consumed by deployed models ([Duran et al. 2024](#); [Ben Chaaben et al. 2025](#); [Öztürk 2020](#)), and *training time* is the resource consumed in fitting a model, an increasingly significant cost for large models ([Cheung et al. 2025](#); [Shanbhag et al. 2022](#); [McIntosh et al. 2019](#)).

#### ***Structural determinants.***

These are properties of how code is structured whose effect on consumption has been measured. The *choice of programming language* is among the most impactful, since compiled languages such as C, C++, and Rust are consistently far more energy efficient than interpreted languages such as Python, Ruby, and Perl, with the seminal benchmark of [Pereira et al. \(2017a\)](#) reporting compiled language averages near 120 J against roughly 2365 J for interpreted languages on identical tasks, a gap of up to twentyfold ([Pereira et al. 2021](#)), a result reproduced and extended across mobile and virtual machine settings ([Cunha et al. 2024](#); [Georgiou et al. 2018](#); [Kholmatova 2020](#); [Ournani et al. 2021](#)). The *choice of algorithm* can change time and space complexity by polynomial or exponential factors, with correspondingly large energy differences at scale ([Pereira et al. 2017a](#); [Felix and Mohankumar 2024](#); [Aljabri et al. 2019](#)). *Data representations and structures* exhibit different cache locality properties that produce energy differences of an order of magnitude for equivalent tasks, as shown for Java collections ([Pereira et al. 2023, 2018](#); [Oliveira et al. 2021](#); [Michanan et al. 2015](#)). *Data access patterns* determine cache miss rates and therefore DRAM activity and energy ([Liu et al. 2015](#); [Rocha et al. 2019](#)). Finer grained structural determinants include *loops* and their optimisations such as fusion, tiling, and unrolling ([Procaccianti et al. 2016](#); [Alessi et al. 2015](#)), *iterations*, which is the count of repeated execution that multiplies whatever lives in the loop body ([Pereira et al. 2017a](#); [Procaccianti et al. 2016](#)), *conditions* and *control statements*, where branch misprediction wastes computational work in hot paths ([Stokke et al. 2016](#); [Gottschalk et al. 2016](#); [Corral-Garcia et al. 2019](#)), *sequence statements*, whose arrangement affects instruction level parallelism and vectorisability ([Jalili and Faghieh 2022](#); [Anistyasari et al. 2018](#); [Gottschalk et al. 2016](#)), *functions*, where call overhead and inlining behaviour matter inside hot

loops (Roque et al. 2025; Babakol et al. 2020), and the *use of reusable software components and libraries*, where mature, well tuned libraries usually outperform bespoke code but can introduce bloat (Balanza-Martinez et al. 2024; Kim et al. 2017).

#### ***Quality and anti pattern determinants.***

These determinants concern deficiencies in code quality and the transformations that address them. *Code smells* are structural deficiencies that signal design problems and are linked to elevated consumption, especially in the Android context, where smells such as Slow Loop, Leaking Thread, and Inefficient Data Structure raise energy use measurably (Fatima et al. 2020; Palomba et al. 2019; Carette et al. 2017; Dhaka and Singh 2016). *Coding malpractices* are functionally correct but resource suboptimal patterns, such as unnecessary object instantiation, redundant copies, or busy waiting, are frequent root causes of energy waste (Pathania et al. 2024; Pereira et al. 2016, 2017b). *Performance bugs* degrade performance and consumption without causing functional failure and often surface only under load (Banerjee et al. 2018; Song et al. 2019). *Dead code* inflates binary size, cache pressure, and compilation cost (Gottschalk et al. 2016; van Gastel et al. 2018; Verdecchia et al. 2018). The two transformational determinants are *refactoring*, a double edged intervention that can reduce or increase energy depending on the change and the surrounding code (Gottschalk et al. 2016; Manikyala 2024; Verdecchia et al. 2018; Anwar et al. 2019; Couto et al. 2020; İbrahim Şanlıalp and Öztürk 2020), and *code optimisation*, the deliberate transformation of code for efficiency, whose outcome is context dependent and occasionally regresses energy while improving runtime (Munoz et al. 2017; Roberts et al. 2015; Corral-Garcia et al. 2019).

#### ***Machine learning specific determinants.***

A small but rapidly growing cluster of the literature is specific to machine learning systems. *Model size*, typically the parameter count or memory footprint, is a design choice made when a model is selected or architected, and it is a linear or worse driver of both training and inference energy (Duran et al. 2024; Ben Chaaben et al. 2025). Its effects are observed chiefly through the temporal impact metrics *inference latency* and *training time*, introduced above, which recur in machine learning contexts as the dominant energy contributors at deployment scale and at training time respectively (Shanbhag et al. 2022; Öztürk 2020).

#### ***Other and emerging determinants.***

Finally, a sizeable *other* category captures implementation specific determinants that do not map cleanly onto a single canonical category but recur across the studies. Concretely, these include concurrency and synchronisation primitives, where lock contention and thread management affect energy in multithreaded applications (Babakol et al. 2020), network input/output and serialisation patterns, including the energy of HTTP request methods and protocol choices (Chowdhury et al. 2016; Anwar et al. 2018), logging behaviour, whose energy cost has been measured directly on Android (Chowdhury et al. 2018), GPU kernel configuration in multimedia and heterogeneous workloads (Stokke et al. 2016; Hayduk et al. 2016), and operating system interaction

patterns (Taherikhonakdar and Fazlollahta 2023). We retain this category because its prevalence, present in 60 of the 137 studies, is itself a finding and signals the breadth of code level sustainability research and the steady emergence of new determinants as the field expands into new domains.

### 3 Methodology

This systematic literature review was conducted in accordance with the guidelines proposed by Kitchenham and Charters (2007) for performing systematic literature reviews in software engineering. Our methodology adheres to the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta Analyses) standards (Page et al. 2021) to ensure transparency, reproducibility, and completeness. The review process comprised five main stages, namely protocol definition, comprehensive search execution, study selection, rigorous quality assessment, and systematic data extraction and synthesis, as summarised in Figure 2. This process resulted in a final set of 137 studies. The complete set of filled data extraction forms and the complete list of included studies with their identifiers, are publicly available in a project repository at [https://github.com/ZoyaMahboob27/Sustainability\\_SLR](https://github.com/ZoyaMahboob27/Sustainability_SLR).

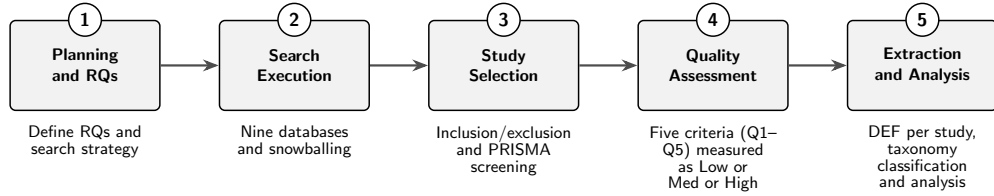


**Fig. 2** Overview of the five stage systematic review process, from protocol definition to data extraction and synthesis. The search execution stage draws on nine databases complemented by backward and forward snowballing, and the study selection stage is detailed in the PRISMA flow of Figure 3.

#### 3.1 Research Questions

The review is guided by five research questions designed to explore the landscape of code level sustainable software engineering and its determinants and impact metrics.

**RQ1:** What are the demographic statistics across the code level sustainable software engineering domain?

*RQ1.1:* Which authors have contributed the most to the field?

*RQ1.2:* Which countries have contributed the most to the field?

*RQ1.3:* How many papers in the domain have been published between December 2015 and January 2026?

*RQ1.4:* In what venues have papers in the domain been published?

**RQ2:** Which code level determinants are reported to affect software sustainability, and which metrics are used to measure their effects?

**RQ3:** In which development phase is each code level determinant decided, and in which phase are its effects measured?

**RQ4:** What impact metrics have been used to determine the effect of code level determinants across the literature?

*RQ4.1:* How many studies use a measure of energy usage as an impact metric?

*RQ4.2:* How many studies use a measure of carbon emissions as an impact metric?

**RQ5:** What research perspectives can be further explored in the domain of code level sustainable software engineering?

## 3.2 Search Process

In order to find the most relevant research papers for this review, we constructed our search strategy around three conceptual pillars that jointly define the scope of code level sustainability in software engineering. The first pillar captures sustainability in software engineering (for example, *sustainable software engineering*, *green software development*, and *energy efficient programming*). The second captures the coding or implementation artefact (for example, *code*, *algorithm*, and *implementation*). The third captures the sustainability measure of interest (for example, *energy consumption*, *carbon emission*, and *environmental impact*). Combining the three pillars with Boolean conjunction yields a search string that is broad enough to capture the field yet specific enough to remain succinct.

The base string was adapted to each database’s syntax, controlled vocabulary, and operator limits. Two databases required deviations from the general form. ScienceDirect imposes a limit on the number of Boolean connectors permitted in a single query, so a reduced string was used, and Semantic Scholar supports a native year filter, which we applied directly in the query. To present these variations transparently, Table 2 reports the exact string applied to each database. The search was conducted across nine academic databases, namely ACM Digital Library, IEEE Xplore, Google Scholar, Springer Link, ScienceDirect, Semantic Scholar, Scopus, Wiley Online Library, and Web of Science, each selected because it indexes peer reviewed software engineering research and because preliminary searches confirmed it contained relevant literature. Google Scholar was included as a supplementary search engine to broaden coverage and its results were deduplicated against the indexed databases.

The rationale behind this structure is that the first pillar scopes the work to sustainability in software engineering, the second restricts it to coding and coding artefacts, and the third ties it to concrete sustainability measures such as energy consumption and carbon emissions. To complement the database search and reduce the risk of missing seminal work, we performed systematic backward and forward snowballing on the set of papers retained after full text screening, following the guidelines of Wohlin (2014). Snowballing yielded an additional 79 studies that are reported alongside the database results throughout this review. Database searches were executed in January 2026, and snowballing concluded in April 2026. Coverage of 2026 is therefore partial.

The search string is deliberately precision oriented. We intentionally excluded broad terms such as power, battery, profiling, or performance, because in isolation these retrieve very large volumes of literature from hardware design, power electronics, and general performance engineering outside the scope of this review, making screening infeasible. Instead, we adopted a hybrid search strategy in which the precise string

**Table 2** Search strings applied to each database. The general string was used for all databases except ScienceDirect (reduced owing to a limit on Boolean connectors) and Semantic Scholar (which supports a native year filter).

| Database                                                                                                          | Search string                                                                                                                                                                                                                                             |
|-------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| General string<br>(ACM, IEEE<br>Xplore, Google<br>Scholar, Springer<br>Link, Scopus,<br>Wiley, Web of<br>Science) | ("sustainable software engineering" OR "green software development"<br>OR "energy-efficient programming") AND ("code" OR "algorithm" OR<br>"implementation") AND ("energy consumption" OR "carbon emission*"<br>OR "environmental impact")                |
| ScienceDirect                                                                                                     | ("sustainable software engineering" OR "green software development")<br>AND ("code" OR "implementation") AND ("energy consumption" OR<br>"carbon emission")                                                                                               |
| Semantic Scholar                                                                                                  | ("sustainable software engineering" OR "green software development"<br>OR "energy-efficient programming") AND ("code" OR "algorithm" OR<br>"implementation") AND ("energy consumption" OR "carbon emission*"<br>OR "environmental impact") year:2015–2026 |

identifies a citation connected core set and systematic backward and forward snowballing serves as the recall mechanism. Hybrid strategies have been shown to achieve a precision and recall balance comparable to broad database searches at substantially lower screening effort (Mourão et al. 2020; Wohlin et al. 2022). The effectiveness of this design is visible in our final set of included studies. Snowballing contributed 79 studies, including work framed as power, battery drain, energy hotspots, and profiling (for example SB-009, SB-013, SB-023, SB-040, SB-055, SB-074). It also resulted in the inclusion of several of the field’s most cited studies, including the programming language energy benchmarks, thus demonstrating the effectiveness of our search string.

### 3.3 Inclusion and Exclusion Criteria

The following inclusion and exclusion criteria were applied during study selection to ensure consistent and objective decision making. The criteria operationalise the three search pillars and additionally require that each study identify both the development phase of its code level determinant and a measure of its impact, to enable the phase mapping of RQ3 and the metric characterisation of RQ4.

#### Inclusion Criteria.

- I1** The paper is published in a peer reviewed conference, workshop, or journal.
- I2** The paper was published between December 2015 and January 2026.
- I3** The full text of the paper is available to the researchers.
- I4** The paper is written in English.
- I5** The paper discusses one or more code level determinants within the context of sustainable software engineering.

- I6** The paper identifies the software development phase to which one or more determinants pertain.
- I7** The paper reports a measure of the impact of one or more determinants in terms of energy or carbon emissions.

### **Exclusion Criteria.**

- E1** The paper is a book chapter or can be categorised as grey literature.
- E2** The paper was published before 2015.
- E3** The full text of the paper is unavailable to the researchers.
- E4** The paper is in a language other than English.
- E5** The paper does not discuss any code level determinant within the context of sustainable software engineering.
- E6** The paper does not identify the development phase to which a determinant pertains.
- E7** The paper does not report a measure of impact in terms of energy or carbon emissions.

Criteria I6 and I7 shape the selected studies by construction, ensuring that every included study necessarily reports an energy or carbon measure and an identifiable development phase. RQ3 and RQ4 therefore characterise distributions within this eligible population rather than the field at large.

## **3.4 Study Selection and PRISMA Flow**

Our study selection followed a multi stage process to minimise bias and ensure rigorous screening. The database search identified 1,269 records across the nine databases. Before screening, 555 records were removed by a first pass application of the record level exclusion criteria, namely document type (book chapters and grey literature) and language, and a further 119 duplicate records were removed through deduplication on DOI and exact title, supplemented by manual review of near duplicates, leaving 595 records for screening. Title screening excluded 503 records, and abstract screening of the remaining 92 excluded a further 32. In both stages two independent reviewers assessed each paper against the criteria, with a third reviewer resolving disagreements. At the eligibility stage, the 60 remaining candidates were assessed in full text against criteria I1 to I7, and 2 records whose full text could not be retrieved were excluded, yielding 58 studies from the database search. We then conducted systematic backward and forward snowballing on this set, applying the same staged screening. Of 817 records identified through backward and forward snowballing, 495 were removed by the record level exclusion criteria and 7 as duplicates, title screening excluded 209 and abstract screening a further 26, and all 80 remaining full texts were retrievable. One record was subsequently merged as an earlier version of another snowballed study, so snowballing contributed 79 additional studies satisfying the same criteria. The final list of studies therefore consists of 58 plus 79 studies which results in a total of 137 studies. Figure 3 documents the number of records identified, screened, and included at each stage.

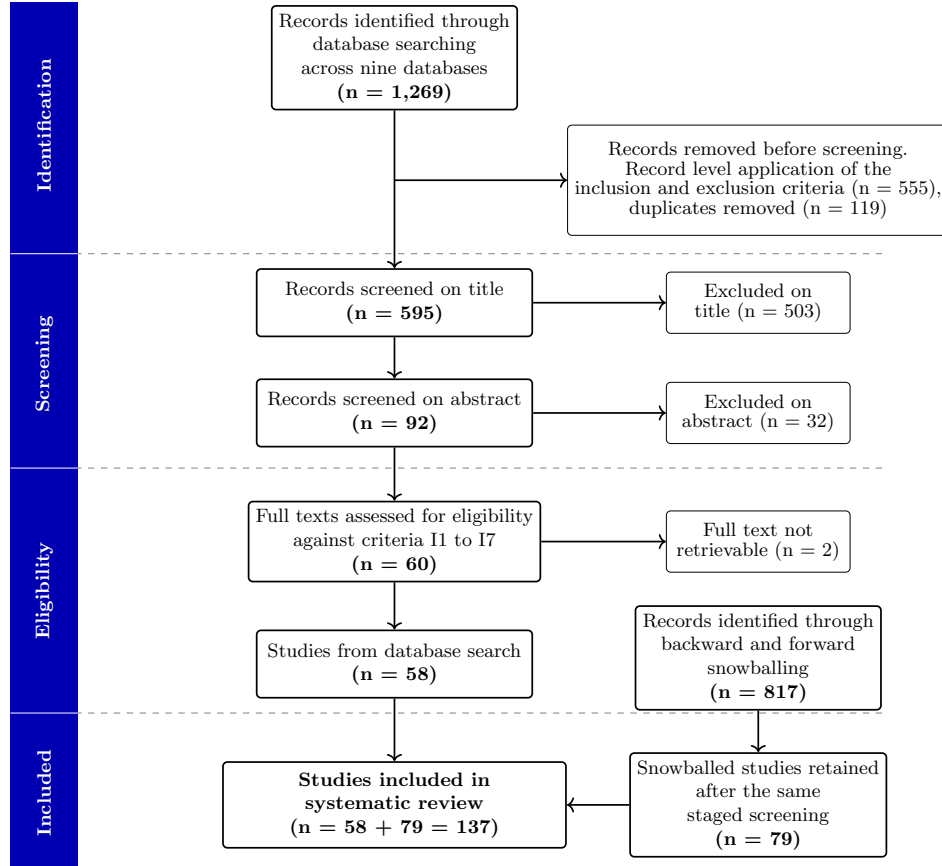


**Fig. 3** PRISMA flow diagram of the study selection process. The right hand column summarises the snowballing arm, whose staged screening (record level criteria, deduplication, title, and abstract screening) is detailed in the text.

### 3.5 Quality Assessment

All 137 included studies underwent a rigorous quality assessment using quality assessment (QA) scores. The criteria are adapted for primary empirical studies following the quality instruments commonly used in software engineering reviews (Dybå and Dingsøyr 2008), deliberately folding in the review specific dimensions of relevance, measurement, and phase. Each study was scored on five criteria, each assigned 1 (fully met), 0.5 (partially met), or 0 (not met):

- Q1** *Selection*. Did the researchers clearly describe how the studied systems, benchmarks, subjects, or workloads were selected, and are those choices appropriate?
- Q2** *Rigour*. Did the researchers evaluate the quality or validity of the studies or systems they examined?
- Q3** *Relevance*. Did the researchers explicitly address one or more code level determinants related to sustainable software engineering?

- Q4** *Measurement*. Did the researchers describe an appropriate measure of the impact of one or more determinants on the sustainability of the software system?
- Q5** *Phase*. Did the researchers discuss the development phase in which a determinant is found?

Each paper then received a Quality Assessment Score (QAS) equal to the sum of the five criteria scores, ranging from 0 to 5. We categorise papers into three quality tiers based on their scores. Low (QAS 0.5 to 2), Medium (QAS 2.5 to 3), and High (QAS 3.5 to 5). Applying this scheme, the literature skews towards higher quality with 111 studies being rated High, 18 Medium, and 8 Low. All 137 papers were independently assessed by two reviewers, with disagreements resolved by discussion and a third reviewer available as a tiebreaker, so that quality judgements are rigorous and not unduly influenced by individual reviewer bias.

### 3.6 Data Extraction

A standardised data extraction form (DEF) was developed to systematically extract information from each included paper. Prior to full scale extraction, a pilot was conducted on five papers to ensure that the DEF captured all relevant information and that field definitions were unambiguous. Data extraction was performed by the first and third authors independently, with the corresponding author serving as final reviewer, and disagreements resolved by discussion. Figure 4 shows the blank form, and Table 3 presents its fields and allowed values as a reusable template that future researchers can adopt or adapt. The form captures bibliographic metadata, the research questions addressed, the five quality assessment criteria, the determinants and impact metrics covered (recorded against the taxonomy of Section 2.2, each treated as a binary checklist item), the development phase addressed, the impact metrics used, the application domain, code artefact availability, and a structured summary.

## 4 Results and Discussion

This section discusses the five research questions presented in Section 3. It includes the answers to each of the research questions, and further details and analysis regarding each RQ.

**RQ1: What are the demographic statistics across the code level sustainable software engineering domain?**

*In brief.* The field is concentrated and maturing. The University of Minho group (Saraiva, Fernandes, Pereira, Cunha) leads in output. The United States (29 studies), Portugal (16), and Spain (15) lead by country. Publications grow steadily, with a 2019 peak and a renewed 2024 to 2025 rise driven by AI focused work, and ACM venues.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |            |                                                                                          |                                           |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|------------------------------------------------------------------------------------------|-------------------------------------------|
| <b>Study Identifier:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |            | <b>Code Artefact Available:</b> <input type="checkbox"/> Yes <input type="checkbox"/> No |                                           |
| <b>Title</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |            |                                                                                          |                                           |
| <b>Year</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |            |                                                                                          |                                           |
| <b>Author(s)</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |            |                                                                                          |                                           |
| <b>Publication</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |            |                                                                                          |                                           |
| <b>Research Questions Covered</b> (tick all that apply)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |            |                                                                                          |                                           |
| <input type="checkbox"/> RQ1.1 <input type="checkbox"/> RQ1.2 <input type="checkbox"/> RQ1.3 <input type="checkbox"/> RQ1.4 <input type="checkbox"/> RQ2 <input type="checkbox"/> RQ3 <input type="checkbox"/> RQ4.1 <input type="checkbox"/> RQ4.2 <input type="checkbox"/> RQ5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |            |                                                                                          |                                           |
| <b>Quality Assessment Score</b> (each $\in \{0, 0.5, 1\}$ ; $QAS = \sum Q_i \in [0, 5]$ )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |            |                                                                                          |                                           |
| <b>Q1:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | <b>Q2:</b> | <b>Q3:</b>                                                                               | <b>Q4:</b>                                |
| <b>Determinants and Impact Metrics Covered</b> (tick all that apply, from the taxonomy of Section 2.2)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |            |                                                                                          |                                           |
| <input type="checkbox"/> Energy usage <input type="checkbox"/> Memory usage <input type="checkbox"/> Execution cycles <input type="checkbox"/> Inference latency <input type="checkbox"/> Training time<br><input type="checkbox"/> Choice of prog. language <input type="checkbox"/> Choice of algorithm <input type="checkbox"/> Data representations <input type="checkbox"/> Data access patterns<br><input type="checkbox"/> Types of loops <input type="checkbox"/> Iterations <input type="checkbox"/> Conditions <input type="checkbox"/> Control statements <input type="checkbox"/> Sequence statements<br><input type="checkbox"/> Functions <input type="checkbox"/> Reusable components <input type="checkbox"/> Code smells <input type="checkbox"/> Coding malpractices<br><input type="checkbox"/> Performance bugs <input type="checkbox"/> Dead code <input type="checkbox"/> Refactoring <input type="checkbox"/> Code optimisation <input type="checkbox"/> Model size<br><input type="checkbox"/> Other |            |                                                                                          |                                           |
| <b>Development Phase</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |            |                                                                                          |                                           |
| <input type="checkbox"/> Implementation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |            | <input type="checkbox"/> Testing                                                         | <input type="checkbox"/> Runtime          |
| <b>Impact Metrics</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |            |                                                                                          |                                           |
| <input type="checkbox"/> Energy                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |            | <input type="checkbox"/> Carbon                                                          | <input type="checkbox"/> Other (specify): |
| <b>Application Domain</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |            |                                                                                          |                                           |
| <input type="checkbox"/> Web <input type="checkbox"/> Mobile                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |            | <input type="checkbox"/> Desktop                                                         | <input type="checkbox"/> Other (specify): |
| <b>Summary of Paper</b> ( $\leq 250$ words)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |            |                                                                                          |                                           |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |            |                                                                                          |                                           |

**Fig. 4** The blank data extraction form completed for each of the 137 primary studies. The *Determinants and Impact Metrics Covered* field is recorded against the taxonomy of Section 2.2, with every metric treated as a binary checklist item. The field definitions and allowed values are given in Table 3, and all 137 completed forms are available in the project repository.

#### 4.1 RQ1.1: Most Contributing Authors

The 137 included studies reveal a concentrated research community. The leading contributors are João Saraiva and João Paulo Fernandes, each appearing as co author on four distinct included papers, followed closely by Rui Pereira, Marco Couto, and Jácome Cunha, each with three publications. A second cluster of prolific contributors includes Silverio Martínez-Fernández, Vibhu Saujanya Sharma, and Vikrant Kaulgud, each contributing to three papers. The Portuguese group from the University of Minho, led by Saraiva, Fernandes, Pereira, and Cunha, has been particularly impactful in establishing energy efficiency across programming languages as a standardized benchmark problem in the field (Pereira et al. 2017a, 2023, 2021; Lima et al. 2016, 2019). The Indian research community, represented by Sharma, Kaulgud, and their collaborators, has focused particularly on mobile and enterprise software sustainability. Figure 5 presents the top contributing authors.

#### 4.2 RQ1.2: Most Contributing Countries

Figure 6 presents the distribution of included papers by the country of affiliation. Where a paper has co authors from multiple countries, each country is credited. The United States leads with 29 papers, followed by Portugal with 16, Spain with 15, Brazil

**Table 3** Data extraction form (DEF) template used for each of the 137 primary studies.

| Field                                   | Content / Allowed Values                                                                      |
|-----------------------------------------|-----------------------------------------------------------------------------------------------|
| Study Identifier                        | ID1 to ID58 (database search) or SB-001 to SB-079 (snowballed)                                |
| Title                                   | Full title of the study                                                                       |
| Year                                    | Publication year (2015 to 2026)                                                               |
| Author(s)                               | Full author list                                                                              |
| Publication                             | Venue name and type (Journal or Conference)                                                   |
| Research Question Coverage              | RQ1.1, RQ1.2, RQ1.3, RQ1.4, RQ2, RQ3, RQ4.1, RQ4.2, RQ5                                       |
| Quality Assessment Score                | Q1–Q5, each $\in \{0, 0.5, 1\}$ ; $QAS = \sum Q_i \in [0, 5]$                                 |
| QAS Level                               | Low (0.5 to 2) or Medium (2.5 to 3) or High (3.5 to 5)                                        |
| Determinants and Impact Metrics Covered | Tick all that apply from the taxonomy in Section 2.2                                          |
| Development Phase                       | Implementation or Testing or Runtime, multiple phases are allowed                             |
| Impact Metrics                          | Energy or Carbon or Execution time or Memory or Other (specify)                               |
| Application Domain                      | Web or Mobile or Desktop or Other (specify if other)                                          |
| Code Artefact Availability              | Yes or No                                                                                     |
| Summary of Paper                        | Structured narrative ( $\leq 250$ words) capturing objective, method, findings, and relevance |

with 12, India with 11, Canada with 10, the Netherlands with 9, France with 8, Pakistan with 7, the United Kingdom with 6, Germany, Italy, and Malaysia with 5 each, Estonia, Turkey, and Russia with 4 each, and Saudi Arabia and Austria with 3 each. The remaining contributing countries include Iran, Algeria, China, Indonesia, Norway, Switzerland, the UAE, and others, each with one or two papers. The strong showing of the United States reflects its broad academic output across software engineering, while Portugal’s high count is disproportionately driven by the Saraiva and Fernandes group at the University of Minho. Spain’s contributions cluster around automated program repair and tool studies, particularly from the UPC group. Brazil and India have emerged as substantial contributors, with Brazil hosting a strong empirical software engineering community and India contributing significantly through its mobile development research. Pakistan’s seven papers reflect a growing research interest in sustainable software at South Asian institutions.

Notably absent or severely underrepresented are several major geographic regions. Africa contributes no papers in our set of studies, and Southeast Asia is represented only marginally through Malaysia. This geographic skew has important implications for the universality and applicability of the field’s findings. Most benchmarking studies are conducted on North American and European hardware, with electricity grids characteristic of developed nations. The absence of research from African and Southeast Asian contexts means that the field has limited understanding of software energy efficiency in resource constrained environments, in contexts with unreliable power

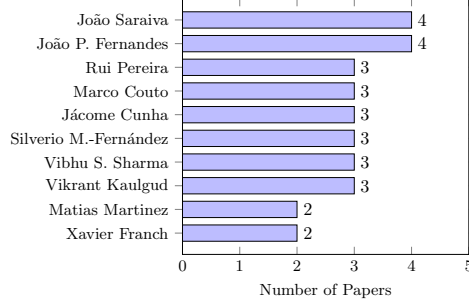


**Fig. 5** Top contributing authors by number of included papers (RQ1.1).

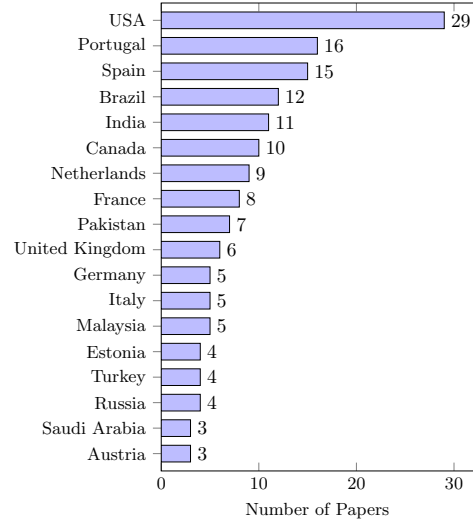


**Fig. 6** Top contributing countries by number of included papers (RQ1.2). Multi country papers contribute one count per country.

infrastructure, or in hardware ecosystems that dominate in developing economies. This is an important gap for future research to address.

### 4.3 RQ1.3: Publication Trends Over Time

Figure 7 presents the distribution of included studies by publication year. The temporal distribution reveals a clear growth trend, with 13 papers in 2015, 16 in 2016, 16 in 2017, 17 in 2018, 21 in 2019, 11 in 2020, 11 in 2021, 3 in 2022, 5 in 2023, 12 in 2024, and 12 in 2025. A 2026 figure is not included, since the 2026 studies did not pass the inclusion criteria (Section 3). The 2015 to 2019 cluster is dominated by snowballing era foundational work, particularly the energy across programming languages line of research, while the 2024 to 2025 cluster is driven by AI focused sustainability work and by the response to the EU Green Deal and the founding of the Green Software Foundation.

### 4.4 RQ1.4: Publication Venues

The 137 included studies were retrieved from nine databases plus snowballing. Figure 8 shows the source level distribution. Snowballing contributed 79 papers, ACM Digital Library 24, Google Scholar 8, Springer Link 7, ScienceDirect 6, IEEE Xplore 5, Scopus 4, Wiley 3, and Web of Science 1. Semantic Scholar is not shown because no included study had it as its retrieval source once duplicates across databases were removed. The dominance of snowballing reflects the fact that the field has matured enough to have well connected citation neighbourhoods, and that backward and forward citation tracing successfully led to the inclusion of foundational work that the keyword search

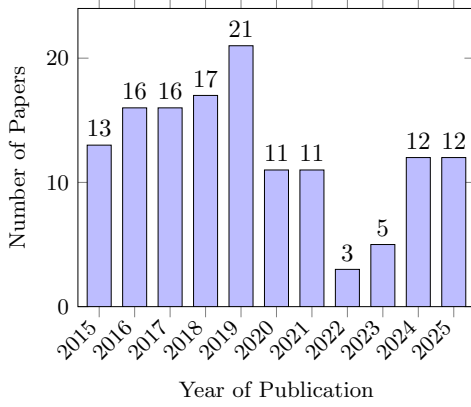


**Fig. 7** Distribution of the included studies by publication year (RQ1.3).

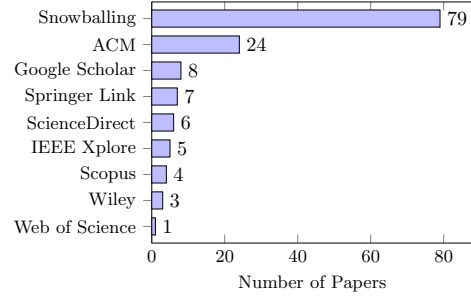


**Fig. 8** Distribution of the included studies by retrieval source (RQ1.4).

alone did not retrieve. Among non snowballing sources, ACM Digital Library dominates, reflecting the receptivity of ACM venues, including ICSE, SAC, ESEM, and TOSEM, to empirical research on code level software engineering concerns.

In terms of publication type, 92 studies (67.2%) were published at conferences and 44 (32.1%) in journals. The preponderance of conference publications is typical of empirically active software engineering sub fields. Among individual venues, the ACM Symposium on Applied Computing (SAC) appears frequently, as does the International Conference on Software Engineering (ICSE), the International Conference on ICT for Sustainability (ICT4S), and the International Workshop on Green and Sustainable Software (GREENS). Journals such as the Journal of Systems and Software, Empirical Software Engineering, and Information and Software Technology are the primary journal outlets.

**RQ2: Which code level determinants are reported to affect software sustainability, and which metrics are used to measure their effects?**

*In brief.* The answer falls into two groups, following the taxonomy of Section 2.2 (Figure 1). Among the impact metrics, energy usage is near universal (131 studies), followed by the temporal metric execution time (88) and memory usage (46). Among the code level determinants, code optimisation leads (70 studies), while structural determinants such as the choice of algorithm (45) and of programming language (41) are less frequent but, as RQ4 and Section 6 show, carry the largest reported effects.

Figure 9 presents the frequency with which each code level determinant and impact metric in the taxonomy of Section 2.2 was identified across the 137 included studies, with the bars ordered by group. Among the impact metrics, energy usage is by far the most pervasive, identified in 131 studies, followed by execution cycles and execution time in 88 studies, memory usage in 46, training time in 8, and inference latency in 5. Among the code level determinants, code optimisation appears in 70 studies, the

“other” category in 60, choice of algorithm in 45, choice of programming language in 41, and coding malpractices in 39. The mid frequency cluster covers data representations (38), use of reusable software components (37), functions (32), refactoring (30), and types of loops (27). The lower frequency tail covers data access patterns (21), code smells (20), iterations (16), performance bugs (13), control statements (12), conditions (8), dead code (7), model size (5), and sequence statements (3).

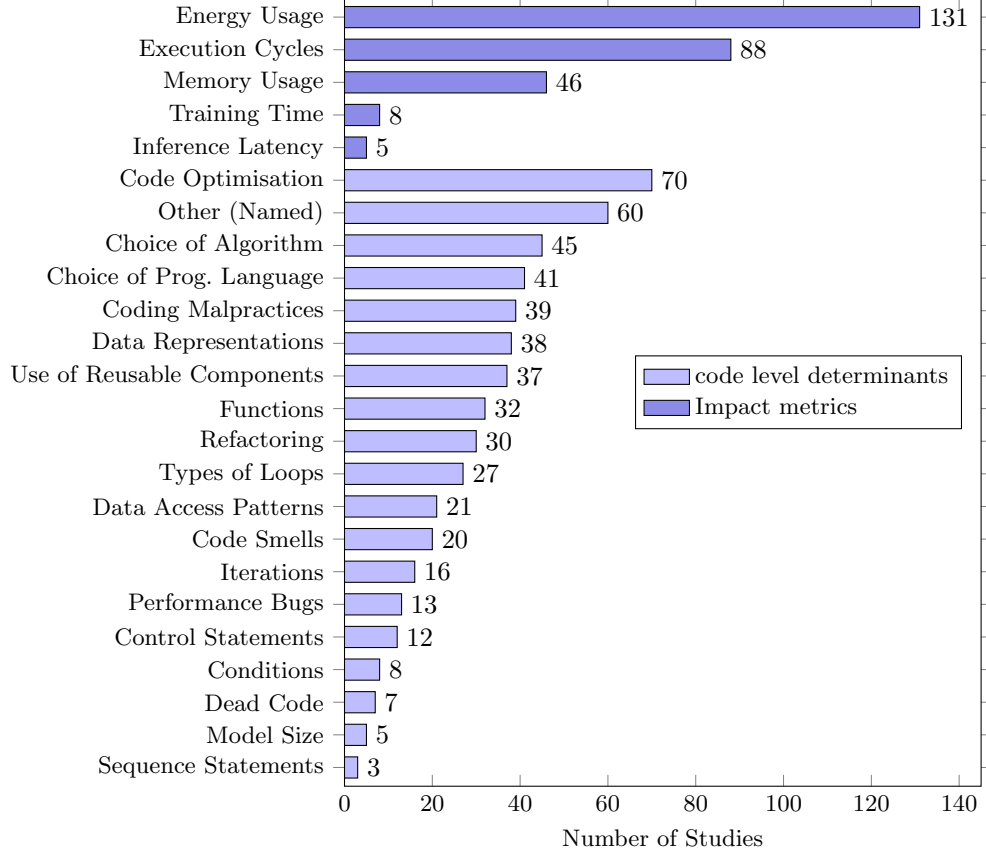


**Fig. 9** Frequency of each code level determinant and impact metric (from the taxonomy of Section 2.2) across the 137 included studies (RQ2). The bars are ordered by group, with the impact metrics (dark) shown together at the top and the code level determinants (light) below.

The prominence of programming language choice deserves substantial attention. A methodologically rigorous cluster of studies has established, through controlled benchmarking on identical hardware with identical workloads, that the choice of programming language can result in energy consumption differences spanning one to two orders of magnitude. Compiled languages such as C, C++, and Rust consistently outperform interpreted languages such as Python, Perl, and Ruby in terms of energy efficiency, with studies showing that C is roughly 20 times more energy efficient than

Python for the same algorithm (Pereira et al. 2017a; Cunha et al. 2024; Hamizi et al. 2021; Mahadevappa and Figueira 2021; Oliveira et al. 2017; Pereira et al. 2021). The mid tier determinants, such as the use of reusable software components, functions, and data representations, form a rich body of literature around the structural and compositional decisions that shape software energy profiles. At the lower frequency end, code smells, refactoring, and data access patterns are important but comparatively understudied. The machine learning specific entries, namely the determinant model size and the impact metrics inference latency and training time, are the least represented in the literature, but their trajectory is clearly upward given the 2024 to 2025 surge in AI focused papers.

**RQ3: In which development phase is each code level determinant decided, and in which phase are its effects measured?**

*In brief.* We separate the phase in which a determinant is decided from the phase in which its effects are measured, using three phases. *Implementation* (writing code), *Testing* (evaluating code), and *Runtime* (executing code). Decisions concentrate in the implementation phase ( $\approx 86\%$  of studies), measurement happens almost exclusively at runtime ( $\approx 95\%$ ), and the testing phase remains markedly underrepresented ( $\approx 12\%$ ), exposing a clear gap in energy aware software testing.

This review distinguishes two roles that a development phase can play for each code level determinant, namely the phase in which the determinant is decided and the phase in which its effects on the impact metrics are measured. We adopt three developer facing phases that map directly onto what a developer is doing. First we have the *Implementation phase* (when code is being written), next is the *Testing phase* (when code is being evaluated), and lastly we have the *Runtime phase* (when code is being executed). A determinant such as the choice of programming language is decided during implementation while its effect is observed at runtime. The two columns of Table 4 separate these roles.

**Implementation phase.** The largest cluster of determinants is decided during the active writing of code. These include the choice of programming language, the selection of algorithms and data structures, the use of reusable libraries, the structuring of functions and control flow, the avoidance of dead code and coding malpractices, loop formulation, and code optimisation strategies. Studies addressing determinants decided at implementation tend to use static analysis, code review methodologies, or controlled experimental benchmarks (Pathania et al. 2024; Munoz et al. 2017; Moises et al. 2018; Gottschalk et al. 2016; Georgiou et al. 2019; Corral-Garcia et al. 2019). Approximately 86% of included studies address at least one determinant decided in the implementation phase, making this the phase in which the literature concentrates its decisions.

**Testing phase.** Determinants rooted in the testing phase include performance bugs, whose energy impact only becomes measurable under load, and test code smells, which introduce unnecessary overhead into automated test suites. These are decided when tests are designed or when the bug is introduced during implementation, and

their effects are measured during testing and at runtime under load. Continuous integration pipeline configuration, test selection strategies, and test infrastructure all affect the energy consumed by continuous integration. Testing phase determinants are the least represented, appearing in approximately 12% of included studies (Misu et al. 2025; van Gastel et al. 2018; Jabbarvand et al. 2016; Jabbarvand and Malek 2017; Chowdhury et al. 2019; Jabbarvand et al. 2019, 2020; Mancebo et al. 2021), signalling a meaningful research gap in energy aware software testing.

**Runtime phase.** Runtime is where measurement happens. The impact metrics, namely energy usage measured with hardware counters, execution cycles measured through profiling, memory consumption during operation, and inference latency and training time in machine learning, are all observed while the code executes. Intel RAPL is the de facto standard for CPU and DRAM energy measurement, alongside JoularMX, the Android Battery Manager API, Perf, and external power measurement hardware. Approximately 95% of included studies measure at runtime, making runtime measurement the dominant paradigm. Table 4 summarises, for each determinant and impact metric, the phase in which it is decided and the phase in which its effects are measured.

**Table 4** Code level determinants and impact metrics mapped to the development phase in which each is decided and the phase in which its effects are measured (RQ3).

| Determinant or metric                                                                                                                                                                                                                                   | Decision phase                                           | Measurement phase              |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|--------------------------------|
| Structural determinants (choice of programming language, choice of algorithm, data representations or structures, data access patterns, types of loops, iterations, conditions, control and sequence statements, functions, use of reusable components) | Implementation                                           | Runtime                        |
| Quality and anti pattern determinants (code smells, coding malpractices, dead code)                                                                                                                                                                     | Implementation                                           | Runtime                        |
| Refactoring and code optimisation                                                                                                                                                                                                                       | Implementation (maintenance)                             | Runtime                        |
| Test code smells and performance bugs                                                                                                                                                                                                                   | Testing (test design) or Implementation (bug introduced) | Testing and Runtime under load |
| Model size                                                                                                                                                                                                                                              | Implementation                                           | Runtime                        |
| Impact metrics (energy usage, execution cycles or execution time, memory usage, inference latency, training time)                                                                                                                                       | Not applicable (it is an outcome variable)               | Runtime                        |

**RQ4: What impact metrics have been used to determine the effect of code level determinants across the literature?**

*In brief.* Energy is the near universal impact metric, used in 131 of 137 studies and measured predominantly with Intel RAPL. Carbon is reported as a primary metric in only three studies. This energy to carbon imbalance is a significant finding. Section 2.2 and the SCI specification together potentially point to how the field’s abundant energy evidence could be converted to the carbon terms that regulation increasingly demands.

#### 4.4.1 RQ4.1: Energy Usage as an Impact Metric

Energy consumption is by far the dominant impact metric employed across the literature, used in 131 of the 137 included studies. The measurement of energy is most often accomplished using Intel’s Running Average Power Limit (RAPL) interface (Babakol et al. 2020; Pereira et al. 2017a, 2016), which provides software accessible hardware performance counters for CPU and DRAM power consumption on x86 architecture processors. The advantages of RAPL include minimal measurement overhead, standard availability across modern x86 processors, and accessibility through standard Linux perf tools. Other tools employed in the literature include dedicated power monitors and smart plugs for whole system measurements, the Android Battery Manager API for mobile applications, JoularMX for Java applications, and external hardware instruments such as Monsoon power monitors. The unit of measurement is most commonly joules, kilowatt hours, watts, or normalised consumption per unit of work. This methodological diversity is itself a challenge of the field. Direct comparison of energy measurements across studies is complicated by differences in hardware, ambient conditions, workload definitions, and profiling instruments, which limits meta synthesisability of quantitative findings (Balanza-Martinez et al. 2024).

#### 4.4.2 RQ4.2: Carbon Emissions as an Impact Metric

Carbon emissions are an explicitly stated impact metric in a substantially smaller subset of the field. Only three studies treat “carbon” or “emissions” as a primary focus. Gupta and Gupta (2025) develop carbon efficient software development practices, Cheung et al. (2025) compare the carbon footprint of manual versus LLM assisted code development and report that LLM assisted development produced on average 32.7 times higher carbon emissions, and Vemuri (2021) reduces web carbon footprints through front end optimisation. A further small set, such as the cloud scheduling study of Uddin et al. (2015), reports carbon alongside energy. The rarity of carbon as a direct metric is, we argue, the most consequential gap this review finds, and it carries a significance disproportionate to its three study footprint for two reasons. To begin with, carbon rather than energy is the quantity that regulation targets. The EU Corporate Sustainability Reporting Directive, climate disclosure rules, and procurement mandates are all framed in carbon terms, so an energy only literature is misaligned with the obligations its findings must eventually inform. The gap, moreover, is closable

rather than fundamental. The Software Carbon Intensity specification (Green Software Foundation 2024) expresses carbon as  $SCI = (E \times I) + M$ , in which the energy term  $E$  is exactly what 131 of our 137 studies already measure, and what is missing is only the disciplined recording of the grid intensity term  $I$  and the embodied term  $M$ . In other words, the field has built an extensive evidence base in energy that could be re expressed in carbon almost immediately if studies recorded grid intensity context alongside their energy measurements, an observation echoed by the carbon efficiency review of Danushi et al. (2025). We return to this as a concrete recommendation for both researchers and industry in Section 8. Figure 10 illustrates the energy versus carbon comparison.

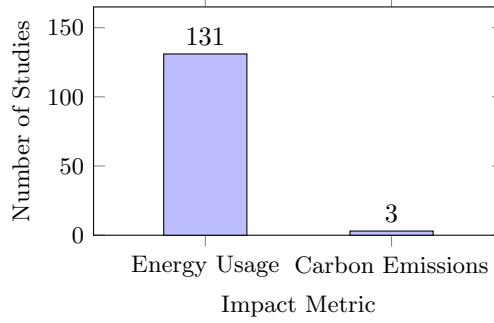


**Fig. 10** Comparison of primary impact metrics across the 137 included studies (RQ4).

**RQ5: What research perspectives can be further explored in the domain of code level sustainable software engineering?**

*In brief.* Six directions stand out. Namely, measurement driven definitions of code level sustainability, low overhead assessment tools and IDE plugins, energy aware testing and sustainable DevOps, domain specific studies beyond Java based web and mobile applications, explicit trade off models such as energy debt, and the sustainability of machine learning. Cutting across all six is the need to report carbon alongside energy and to standardise measurement so findings can be pooled.

Our synthesis of the primary studies reveals a rich landscape of underexplored research directions, organised into six thematic areas.

**Definitional and Conceptual Foundations.** There is no universally accepted definition of software sustainability that maps cleanly to measurable code level properties. Future work could develop rigorous, measurement driven definitions of sustainability at the code level, grounded in ISO software quality models such as ISO/IEC 25010, and aligned with the UN Sustainable Development Goals, particularly SDG 12 and SDG 13 (König et al. 2025).

**Assessment Tools and Methodologies.** A recurring limitation is the absence of standardised, low overhead tools for assessing the energy sustainability of code during

development (Maquoi et al. 2025; Pereira et al. 2016). Future research could develop static analysis approaches for energy estimation that provide fast feedback without execution overhead, or IDE integrated plugins that provide real time feedback on the estimated energy impact of code as it is written.

**Sustainable DevOps and Continuous Integration.** The intersection of sustainability and DevOps practices remains almost entirely unexplored. SusDevOps (David 2025), green software testing, and energy aware CI/CD pipelines are all open research directions. The energy consumed by test suites across an entire organisation, with thousands of CI runs per day, can easily eclipse the energy consumed by production code.

**Domain Specific Sustainability Studies.** The current literature is heavily dominated by studies on Java based web and mobile applications. Domain specific sustainability studies are needed for game development, cloud native applications, microservices, container and serverless architectures, real time embedded systems, autonomous vehicles, and self adaptive systems.

**Trade off Analysis and Optimisation.** Improvements in energy efficiency often come at the cost of maintainability, readability, or performance, along with other dimensions such as correctness and developer productivity (Balanza-Martinez et al. 2024). Therefore the concept of “energy debt” (Maia et al. 2020; Manikyala 2024) deserves further empirical development as a framework for prioritising and communicating about energy sustainability investments.

**Machine Learning and AI Sustainability.** The rapid growth of machine learning in software systems has created a new frontier for code level sustainability research. The energy cost of training, fine tuning, and prompting large language models deserves continued investigation, as does the sustainability of different inference strategies (Cheung et al. 2025; Ben Chaaben et al. 2025).

## 5 A Decision Framework for Determinant Selection

For practitioners and researchers to decide on determinants to focus on, we have provided a decision framework they can act on. The framework maps commonly reported sustainability problems to the code level determinants that the literature identifies as their primary cause and the most productive focus for remediation. It is grounded in the included studies and the structured summaries captured in our data extraction forms. It is not prescriptive, since individual contexts always require judgement, but it provides a literature backed default for where to look first when a particular symptom is observed. The framework is a literature grounded, hypothesis generating triage aid. It has not yet been validated in independent field settings, so its output should be treated as candidate determinants to inspect first rather than guaranteed diagnoses. The mapping below was reviewed for plausibility by software engineers at TechQuest.ai. This informal review does not constitute an empirical validation, which is left to future work.

## 5.1 Problem to Determinant Mapping

Through careful analysis of the structured summaries of all the included studies, we identified ten recurring sustainability problem signatures. For each signature, we recorded which determinant the relevant studies identified as the primary contributor and which they recommended as the focus of remediation. Table 5 presents the resulting mapping. The Primary Determinant column lists the determinant most frequently named as the dominant contributor for that problem, the Secondary Determinants column lists supporting determinants worth investigating next, and the Representative Studies column lists studies that exemplify the link.

The mapping in Table 5 is the core of the framework. The flowchart in Figure 11 operationalises it as a decision aid, presenting the same content as a sequence of questions that a developer or researcher can walk through when faced with a concrete sustainability problem. The flowchart starts from the observed symptom, narrows down to the development phase at which the problem manifests, and then directs the reader to the primary determinant identified by the literature for that symptom.

## 5.2 How to Apply the Framework

The framework is designed for two audiences. For practitioners, it provides a fast triage path. Given a concrete symptom (a high energy bill on a service, a battery complaint on a mobile app, a CI pipeline that is taking longer to run with each release), the practitioner can walk through the flowchart in Figure 11 and obtain a short list of determinants to inspect first. This avoids the common failure mode in which sustainability work starts with the most accessible optimisations (often micro level loop tuning) when the dominant cause is something higher in the stack (often the choice of programming language or algorithm). For researchers, the same mapping serves as a prompt for future empirical studies. If a problem signature in Table 5 is supported by only a handful of studies, that link is a candidate for the next replication, extension, or contradiction.

Every row in Table 5 is supported by at least three primary studies, and the primary determinant for each row was selected only when a clear majority of those studies named it as the dominant contributor.

## 6 Per Metric Synthesis

The frequency counts of RQ2 tell us how often each determinant and impact metric is studied, but not what the studies actually found. This section closes that gap by synthesising, on a per metric basis, the exact studies that report each metric together with the analytical substance of their findings, drawn from the structured summaries in our data extraction forms. However, since the studies we selected report effects on incompatible scales (joules, watts, percentage differences, normalised speedups) and under heterogeneous workloads, a pooled quantitative effect size would imply a precision the literature does not support. Instead we provide a structured, traceable synthesis which can be tracked back to specific primary studies.

**Table 5** Literature grounded mapping from common sustainability problems to the code level determinants that should be the primary focus of remediation. Study identifiers refer to the complete study list in the Supplementary Material.

| Reported Problem                                                      | Primary Determinant                   | Secondary Determinants                    | Studies                                            |
|-----------------------------------------------------------------------|---------------------------------------|-------------------------------------------|----------------------------------------------------|
| Energy use far higher than competing implementations of the same task | Choice of programming language        | Code optimisation, choice of algorithm    | ID20, ID27, SB-005, SB-068                         |
| Performance acceptable but energy per transaction unexpectedly high   | Data access patterns, data structures | Memory usage, iterations                  | ID47, SB-001, SB-010, SB-020                       |
| Sluggish behaviour appears only under realistic load                  | Performance bugs                      | Coding malpractices, iterations           | ID57, SB-038, SB-048                               |
| Frequent garbage collection pauses and high DRAM activity             | Data representations, functions       | Memory usage, code smells                 | SB-001, SB-017, SB-061                             |
| Mobile app drains battery faster than feature use justifies           | Code smells (Android)                 | Coding malpractices, refactoring          | ID48, SB-031, SB-036, SB-066                       |
| Test suite or CI pipeline consumes disproportionate energy            | Test smells, performance bugs         | Code smells, iterations                   | ID42, SB-030, SB-041, SB-049, SB-057, SB-069       |
| Algorithm scales badly with input size                                | Choice of algorithm                   | Iterations, types of loops                | ID20, ID46, SB-023, SB-055                         |
| ML inference too costly at deployment scale                           | Model size, inference latency         | Choice of algorithm, data representations | ID13, ID58, SB-016                                 |
| ML training run produces excessive carbon footprint                   | Training time, choice of algorithm    | Choice of language, data access patterns  | ID11, ID58, SB-079                                 |
| Refactoring or maintenance suspected of regressing energy             | Refactoring, coding malpractices      | Code smells, functions                    | ID50, ID54, SB-015, SB-063, SB-064, SB-066, SB-078 |

## 6.1 Metrics Against Their Reporting Studies

Table 6 is the core of this section. For each metric in the taxonomy of Section 2.2, it records the metric’s category, its frequency across the final set of studies, a representative set of the studies that report it, and a summary of what those studies collectively

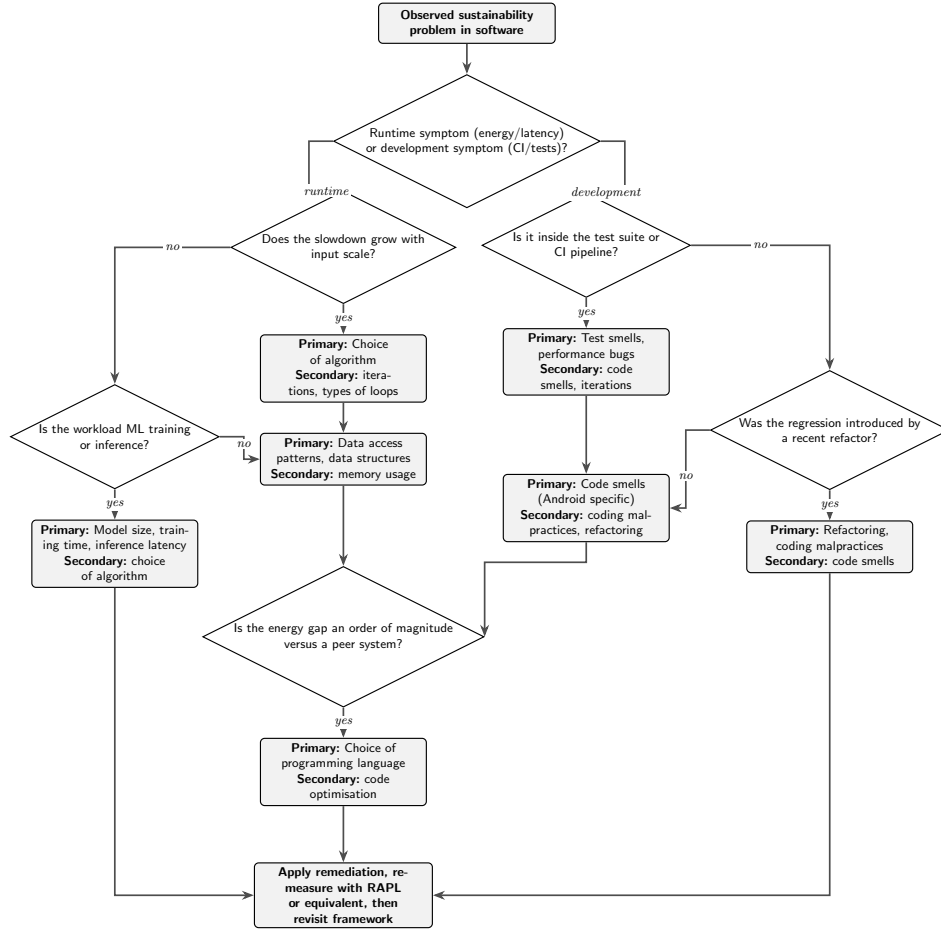


**Fig. 11** Decision framework flowchart that operationalises Table 5. Starting from the observed sustainability problem, the developer or researcher answers a short sequence of yes or no questions and is directed to the determinant or determinants the literature identifies as the primary focus for remediation.

find. To keep the summary honest about the strength of evidence, the final column also notes, where relevant, the direction and consistency of the reported effect, including the cases where the literature is contradictory.

**Table 6:** Per metric evidence synthesis. Frequency is the number of studies (of 137) that report the metric. Representative studies are a traceable sample of the given metric, identified by the study identifiers listed in the Supplementary Material. The synthesis column mentions the analytical findings recorded in the data extraction forms.

| Metric (category)                       | Freq. | Representative studies             | Synthesis of reported findings                                                                                                                                                                                                                 |
|-----------------------------------------|-------|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Consumption metrics</i>              |       |                                    |                                                                                                                                                                                                                                                |
| Energy usage                            | 131   | ID2, ID20, ID43, ID18, SB-005      | The near universal metric. Most studies measure it directly with Intel RAPL and report it in joules, kWh, or watts. It is both the dominant outcome variable and the metric against which other determinants are judged.                       |
| Memory usage                            | 46    | ID20, ID19, ID24, SB-001           | Strongly correlated with energy in DRAM heavy workloads but a weaker signal elsewhere, reported as a correlate of energy more often than as an independent driver.                                                                             |
| <i>Temporal and performance metrics</i> |       |                                    |                                                                                                                                                                                                                                                |
| Execution cycles / time                 | 88    | ID7, ID18, ID20, SB-050, SB-073    | A strong proxy for energy, but several studies show time and energy can diverge (for example, under power caps), so it is reliable for ranking but not a substitute for direct energy measurement.                                             |
| Inference latency                       | 5     | ID13, ID22, ID58, SB-079           | Rare but high impact, driving per request energy at deployment scale in machine learning systems and growing with model complexity.                                                                                                            |
| Training time                           | 8     | ID11, ID13, ID22, SB-016           | Each appearance reports large absolute energy (and, where measured, carbon) costs, dominated by model and data scale.                                                                                                                          |
| <i>Code structural determinants</i>     |       |                                    |                                                                                                                                                                                                                                                |
| Choice of programming language          | 41    | ID20, ID18, SB-005, SB-050, SB-068 | The highest impact structural determinant. Controlled benchmarks consistently report compiled versus interpreted gaps of roughly one order of magnitude (around 120 J versus 2365 J in ID20), with direction highly consistent across studies. |

*continued on next page*

Table 6 – *continued from previous page*

| Metric (category)                                 | Freq. | Representative studies               | Synthesis of reported findings                                                                                                                                                                                |
|---------------------------------------------------|-------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Choice of algorithm                               | 45    | ID20, ID46, SB-023, SB-055           | Routinely produces order of magnitude energy differences at scale, and the consistent message is that asymptotic improvement dominates local tuning.                                                          |
| Data representations / structures                 | 38    | ID47, SB-001, SB-017, SB-020, SB-034 | Cache locality differences yield single digit factor energy gaps for equivalent operations (for example, array backed versus pointer based collections), with direction consistent when access is sequential. |
| Data access patterns                              | 21    | SB-010, SB-061, ID24                 | High impact via cache miss behaviour, but comparatively understudied. Cache friendly access can change energy significantly.                                                                                  |
| Types of loops                                    | 27    | ID44, ID40, SB-028                   | Tiling, fusion, and unrolling deliver modest but reliable gains.                                                                                                                                              |
| Iterations                                        | 16    | ID20, ID44, SB-062                   | Mostly a multiplier on whatever lives inside the loop body, so reducing redundant iterations is a direct lever.                                                                                               |
| Conditions                                        | 8     | ID24, ID50, ID51                     | Important inside hot loops, otherwise dominated by neighbouring determinants.                                                                                                                                 |
| Control statements                                | 12    | ID24, ID50, SB-062                   | Branch misprediction matters in hot paths, marginal elsewhere.                                                                                                                                                |
| Sequence statements                               | 3     | ID29, ID49, ID50                     | Cumulative effect only, rarely a leading cause.                                                                                                                                                               |
| Functions                                         | 32    | ID3, ID19, ID20, SB-061              | Call overhead and inlining behaviour matter chiefly inside hot loops, and pure functions enable compiler optimisations.                                                                                       |
| Reusable components                               | 37    | ID35, SB-035, ID26                   | Mature, well tuned libraries usually beat bespoke code, but dependency bloat can negate the gain.                                                                                                             |
| <i>Code quality and anti pattern determinants</i> |       |                                      |                                                                                                                                                                                                               |
| Code smells                                       | 20    | ID48, SB-031, SB-036, SB-037         | High impact in Android specific contexts (smells such as Slow Loop, Leaking Thread), inconsistent for general purpose code.                                                                                   |
| Coding malpractices                               | 39    | ID5, ID43, ID23, SB-014              | High impact in hot paths, low in cold paths, and a frequent root cause that static and dynamic analyses target.                                                                                               |

*continued on next page*

Table 6 – continued from previous page

| Metric (category)                        | Freq. | Representative studies             | Synthesis of reported findings                                                                                                                                                 |
|------------------------------------------|-------|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Performance bugs                         | 13    | SB-038, SB-060, ID57               | Low frequency but very high impact when present, and a gateway to testing phase work.                                                                                          |
| Dead code                                | 7     | ID50, ID57, SB-015                 | Mostly an indirect effect through binary size and cache pressure.                                                                                                              |
| Refactoring                              | 30    | ID50, ID54, SB-015, SB-063, SB-078 | The most contradictory determinant. Both energy savings and regressions are reported, depending on the refactoring and the surrounding code.                                   |
| Code optimisation                        | 70    | ID8, SB-024, SB-062, SB-013        | Frequent and moderate to high impact, but context dependent, since the same family of optimisations can save or regress energy.                                                |
| <i>Machine learning specific metrics</i> |       |                                    |                                                                                                                                                                                |
| Model size                               | 5     | ID13, ID58, ID22                   | A linear or worse driver of training and inference energy, with consistent direction across the few studies that report it.                                                    |
| <i>Other or emerging metrics</i>         |       |                                    |                                                                                                                                                                                |
| Other (named)                            | 60    | ID19, ID24, SB-029, SB-042         | A heterogeneous bucket (concurrency, network I/O, logging, GPU kernels, OS interaction). Impact must be judged per study, but its prevalence signals the breadth of the field. |

## 6.2 What the Synthesis Tells Us

Table 6 suggests three patterns worth dwelling on. The first is a misalignment between how often a determinant is studied and how much it actually matters. The choice of programming language and the choice of algorithm produce the largest and most consistent effects, yet neither sits near the top of the frequency ranking, so the literature collectively underweights what matters most for practice. A second pattern runs in the opposite direction. The rarest determinants, model size, training time, and inference latency, are often the highest in impact when they apply, and these same metrics are the ones whose publication rate is climbing fastest in the 2024 to 2025 cohort, evidence that the field is already correcting the gap in real time. The third pattern concerns the determinants the literature most often contradicts itself about, refactoring, code smells, and code optimisation, whose effect direction depends on surrounding context, which makes them the obvious targets for replication studies that could pin down when they help and when they hurt. These observations carry directly into the

developer guidelines of Section 7, which prioritise the high impact determinants and insist on measurement wherever the evidence pulls in both directions.

## 7 Guidelines for Developers

The synthesis in Section 6 helps identify which determinants combine high prevalence in the literature with high reported impact on sustainability outcomes. This section translates that synthesis into a number of actionable guidelines that practising developers can apply directly. Each guideline names the action, explains why it is supported by the literature, and points to the studies that underpin it. The guidelines are deliberately narrow and concrete since a handful of high impact rules that are anchored in evidence are more likely to influence everyday practice.

**G1. Choose the programming language and the algorithm first, as these are the highest leverage decisions.** The two determinants with the largest and most consistent effects in the literature are the choice of programming language and the choice of algorithm, and both are settled early. Cross language benchmarks repeatedly report gaps of roughly one order of magnitude between the most and least efficient mainstream languages on equivalent workloads (Pereira et al. 2017a; Mahadevappa and Figueira 2021; Oliveira et al. 2017; Pereira et al. 2021; Georgiou et al. 2018; Kholmatova 2020), and algorithmic improvement from a quadratic to a near linear approach yields order of magnitude energy savings at scale that no amount of local tuning can match (Pereira et al. 2017a; Felix and Mohankumar 2024; Uddin et al. 2015; Aljabri et al. 2019). The practical rule has two parts. When a system is energy critical and the team controls the language, prefer a compiled language (such as C, C++, Rust, Go) for the energy hot path. The choice need not be all or nothing, since a common and effective pattern is to keep a productivity language at the boundary and push the inner loops into a compiled extension (Oliveira et al. 2017). Then, before optimising, profile to find the dominant cost and ask whether a better asymptotic algorithm exists for it. Applying micro optimisations to an inefficient algorithm, or tuning a language hot path before fixing the algorithm, is the most common way the literature reports energy budget being wasted.

**G2. Pick data structures for cache locality and keep dependencies lean.** Equivalent asymptotic complexity does not imply equivalent energy. Contiguous structures routinely outperform pointer based ones by a single digit factor on the same logical operations because contiguous memory is cache friendly and pointer chasing is not (Pereira et al. 2023; Hasan et al. 2016; Pereira et al. 2018; Oliveira et al. 2021; Michanan et al. 2015), so when access is predominantly sequential, prefer contiguous structures and reserve linked structures for workloads dominated by arbitrary position insertion and deletion, where an empirical comparison is warranted. The same locality logic extends to dependencies. Mature, well exercised libraries are usually more energy efficient than bespoke code because they have been profiled and tuned across many use cases (Balanza-Martinez et al. 2024; Moises et al. 2018; Kim et al. 2017), but a library used for one of the ten things it does well still ships the other nine, inflating binary size, cache pressure, and transitive dependencies. Track which library is used for what, and audit the dependency footprint at release time.

**G3. Measure energy directly, and let measurement, not intuition, govern refactoring and optimisation.** The determinants where the literature most often contradicts itself are refactoring and code optimisation, where the same transformation can reduce or increase energy depending on the surrounding code (Gottschalk et al. 2016; Manikyala 2024; Abdulsalam et al. 2015; Verdecchia et al. 2018; Anwar et al. 2019; Cruz et al. 2019; İbrahim Şanlıalp and Öztürk 2020). The discipline that resolves this is direct measurement. Wall clock time is a useful proxy but not a substitute, and the studies contain cases where a faster implementation consumed more energy (Abdulsalam et al. 2015). Measure energy itself with RAPL on x86, JoularMX or similar on managed runtimes, and the platform battery API on mobile (Pereira et al. 2017a, 2016; Babakol et al. 2020). Concretely, measure energy before and after any refactor or optimisation on a realistic workload, accept the change only if the measurement supports it, and keep the comparison in version control as a regression baseline. This rule also subsumes a robust mobile finding, namely that Android code smells are repeatedly linked to elevated battery drain and can be detected and fixed with measurable savings (Fatima et al. 2020; Palomba et al. 2019; Carette et al. 2017; Dhaka and Singh 2016; Couto et al. 2020), so on mobile, treat code smells as energy bugs rather than mere style.

**G4. Extend energy discipline beyond production code, to tests, machine learning models, and an explicit budget.** The highest impact opportunities outside the obvious hot path are consistent across the literature. Test code that runs thousands of times a day can consume more total energy than the production code it exercises, so apply the same hygiene to it. Reduce duplicate work, prefer selective test selection over running everything, and treat test smells as energy bugs (Misu et al. 2025; Jabbarvand et al. 2016; Jabbarvand and Malek 2017; Chowdhury et al. 2019; Jabbarvand et al. 2019, 2020; Mancebo et al. 2021). For machine learning systems, the dominant deployment scale contributors are model size and inference latency, both scaling linearly or worse with parameter count, so start from the smallest model that meets the accuracy target and apply quantisation or pruning before scaling up, rather than starting large and trimming (Duran et al. 2024; Ben Chaaben et al. 2025; McIntosh et al. 2019). Finally, make the energy budget explicit, exactly as teams make performance budgets explicit. Studies report that teams tracking an explicit budget per feature, request, or transaction catch and reverse regressions faster than teams that leave energy implicit (David 2025; Hindle 2015). A single number per critical path, tracked over time and treated as a defect when it regresses, is enough.

## 8 Implications for Researchers and Industry Professionals

### 8.1 Implications for Researchers

The synthesis in this review points to several research directions that are both under explored and high value, in the sense that they combine high reported impact in the few existing studies with low current publication density.

**Replication of high impact, low frequency findings.** Determinants such as data access patterns, code smells outside the Android context, and refactoring effects on energy are characterised by a few influential studies with large effect sizes and a long tail of single study claims. The field would benefit from systematic replication studies that pin down the conditions under which these effects hold. The framework in Section 5 can be used as a prompt for the next replication, any row in Table 5 that is supported by three or four studies is a defensible candidate for a stronger replication.

**Energy aware testing and DevOps.** The testing phase is materially underrepresented in the literature, with only about 12% of studies addressing testing phase metrics. Given the ubiquity of CI/CD in modern software, this is an unusually large gap relative to its practical importance. Concrete research questions include the energy implications of test selection strategies, the energy footprint of major CI/CD platforms, and how to design test smells that are also energy regressors (Misu et al. 2025; David 2025).

**Address carbon, not just energy.** Only three of the 137 studies adopt carbon as a primary metric (Gupta and Gupta 2025; Cheung et al. 2025; Vemuri 2021). As CSRD style disclosure obligations mature, the gap between the energy first literature and the carbon first regulatory frame will widen. Researchers can close it by adopting the Software Carbon Intensity formula, by reporting grid intensity assumptions, and by treating carbon and energy as twin metrics rather than substitutes.

**Trade off models.** The literature repeatedly reports tensions between energy efficiency on the one hand and maintainability, portability, correctness, and developer productivity on the other, but it rarely models the trade off explicitly (Balanza-Martinez et al. 2024). Formal multi objective optimisation models, possibly framed around the energy debt analogy (Maia et al. 2020; Manikyala 2024), would let teams reason about these tensions rather than resolve them ad hoc.

**Methodological standardisation.** A persistent obstacle to formal meta analysis in this field is that energy measurements are reported on incompatible scales across studies. A community level standard for reporting energy alongside hardware, ambient, and workload context, ideally aligned with SCI, would unlock genuinely quantitative synthesis and would help future systematic reviews move past frequency counts into pooled effect size estimates.

**Geographic and platform diversity.** Africa is absent in the studies, and South-east Asia is barely represented. Hardware platforms outside x86 and consumer ARM are similarly absent. Researchers in underrepresented regions and on underrepresented platforms can fill a defensible gap by replicating canonical benchmarks on local hardware and reporting the energy results.

## 8.2 Implications for Industry Professionals

The same synthesis carries direct implications for engineering organisations and the practitioners inside them.

**Concentrate effort on the highest impact determinants first.** The synthesis in Section 6 is unambiguous about where the leverage is. Programming language choice, algorithm choice, and data structure choice carry the highest reported impact.

An organisation that wants to make a measurable dent in its software carbon footprint should target these determinants before spending time on lower impact micro optimisations.

**Use the decision framework as a triage aid.** Most production incidents that are eventually traced to sustainability problems start as vague symptoms (a service that costs more to run than expected, a phone app that drains battery, a CI pipeline that is taking longer each quarter). The flowchart in Figure 11 is intended for exactly this stage. It is a short, literature backed yes or no walkthrough that surfaces a primary determinant to inspect, which is usually faster and more accurate than relying on intuition alone.

**Adopt an explicit energy budget.** The most reliable predictor in the literature for whether a team will catch energy regressions early is whether the team tracks an explicit energy budget at all (David 2025; Hindle 2015). The exact form of the budget matters less than its existence. Joules per request, watt hours per build, or kilowatt hours per training run all work, provided the number is tracked over time and a regression is treated as a defect.

**Measure with hardware counters and standardise the measurement.** There are many cases where wall clock improvements masked energy regressions and vice versa (Abdulsalam et al. 2015). The fix is to measure energy directly, with RAPL on x86, JoularMX on the JVM, or platform battery APIs on mobile, and to standardise the workload, the hardware, and the environment so that the measurement is reproducible across releases.

**Make sustainability a code review concern.** Several studies report that introducing sustainability checks at code review time, even as informal checklists, catches energy regressions before they reach production (Jagroep et al. 2017; Delfim et al. 2017; Ournani et al. 2020). The check does not have to be sophisticated. A short list aligned with Section 7 guidelines, applied during the same review that already covers correctness, security, and style, is enough to shift the average behaviour.

**Plan ahead of CSRD and similar regulation.** The regulatory frame is moving from energy to carbon, and from voluntary disclosure to mandatory reporting. Industry teams that wait until the obligation lands will discover that their telemetry is shaped for energy and that the carbon view requires assumptions about grid intensity that were never recorded. The fix is to start recording grid intensity context alongside energy now, in line with the SCI formula, so that the carbon view can be reconstructed when it is required.

## 9 Threats to Validity

Like all systematic literature reviews, this work is subject to threats to validity that should be understood and acknowledged when interpreting and applying our findings. We organise the threats into four standard categories.

### 9.1 Internal Validity

Internal validity threats concern the risk of bias in our study selection, quality assessment, and data extraction processes. The interpretation of inclusion and exclusion

criteria, of quality assessment criteria, and of which determinants and impact metrics a paper actually covers all involve a degree of subjectivity. We mitigated this by using two independent reviewers at each screening stage, by establishing detailed, written decision rubrics in our protocol, by piloting the data extraction form on five papers before the full extraction, and by having the corresponding author serve as the third reviewer and arbiter in cases of disagreement. Inter rater agreement was high (Cohen’s kappa exceeding 0.85 for categorical fields and intraclass correlation exceeding 0.90 for continuous fields), which gives us confidence that the residual bias is small. A different review team might still make slightly different inclusion decisions or quality ratings, and we have made the inclusion criteria, the quality criteria, and the data extraction form public to support replication.

## 9.2 External Validity

External validity threats concern the generalisability and applicability of our findings beyond the stated studies. The restriction to English language publications means that non English research traditions, particularly research published in Chinese, Spanish, Portuguese, German, or Japanese, are underrepresented. Access constraints during the review, including paywall limitations, resulted in the exclusion of two papers whose full text could not be obtained, which may introduce a small bias toward open access and highly cited work. The December 2015 to January 2026 publication window excludes earlier foundational work, although a backward snowballing step partly compensates for this. The geographic skew across the studies (heavily European and North American, with minimal African and Southeast Asian representation) limits the universality of the findings, and our guidelines and framework should be applied with care in contexts that the literature does not cover, particularly low resource hardware and electricity grids with substantially different carbon intensity profiles.

## 9.3 Search Validity

A further threat concerns search term selection. The sustainability pillar of our string omits broader framings such as power, battery, profiling, and performance energy trade off, so relevant studies that use only those framings and are not citation connected to the green software literature could have been missed. We attempted to mitigate this through systematic backward and forward snowballing, which contributed to studies framed in exactly those terms, and by verifying that several of the field’s most cited studies, including the programming language energy rankings, appear in the final set of studies. The residual risk is limited to isolated studies outside the field’s citation network, which the retrieval profile of the final set of studies reached through citation links (Section 4.4), suggests are uncommon.

## 9.4 Construct Validity

Construct validity threats concern whether our operationalisation of key concepts (the code level determinants, the impact metrics, the development phases, and the impact rating in the synthesis) accurately reflects the intended constructs. The taxonomy of determinants and impact metrics was derived inductively from the literature, and a

different inductive coding could plausibly produce a different set of categories or different boundaries. We document each metric in Section 2.2 so that the reader can judge the boundaries. The development phase categorisation (Implementation, Testing, Runtime) is necessarily a simplification. Several metrics, such as data structure choice and choice of language, influence all three phases, and we have classified each by its dominant phase rather than its only phase. The impact rating in the synthesis is deliberately ordinal because the underlying effect sizes are reported on incompatible scales, and a pseudo numerical aggregation would imply a precision the literature does not support.

## 9.5 Conclusion Validity

Conclusion validity threats concern the risk of drawing incorrect conclusions from the synthesised data. The field’s rapid growth means that our conclusions may not apply to rising subfields such as AI. Furthermore, the concentration of research effort among a small number of authors and institutions, particularly the University of Minho group, introduces the risk that our synthesis partially reflects the preferences and methodologies of that community. The dominance of conference publications (67.2%) over journal articles may introduce a small bias toward positive results, since negative results are reported more often in journals. The methodological diversity in energy measurement across studies, with different hardware, ambient conditions, workloads, and instruments, limits our ability to synthesise quantitative findings into pooled effect sizes, which is why our synthesis is ordinal rather than numerical. We have attempted to mitigate these risks through transparent reporting and by grounding our conclusions in direct evidence from the included studies at each step.

## 10 Conclusion

This systematic literature review has synthesised 137 peer reviewed studies published between December 2015 and January 2026, conducted in accordance with established systematic review guidelines, to characterise the landscape of code level sustainable software engineering. The review constructs a taxonomy of the code level determinants and impact metrics of software sustainability, maps them to the implementation, testing, and runtime phases of development, characterises the impact metrics used, identifies the demographic and venue distribution of research effort, uses literature based evidence to suggest a decision framework and flowchart that practitioners can use as triage aids, presents a per metric evidence synthesis that contrasts frequency with reported impact, and offers actionable guidelines for developers along with explicit implications for researchers and industry professionals.

Our findings indicate that the field has matured substantially, with clear research directions, established methodologies, and growing international participation. The renewed growth in publication activity suggests that sustainability is moving from a niche concern toward a mainstream priority in software engineering. The taxonomy of determinants and impact metrics provides a structured vocabulary for discussing code level sustainability. The mapping of these metrics to development phases reveals that the implementation and runtime phases are well studied while the testing phase

is critically underexplored. The frequency versus impact synthesis shows that the highest impact determinants, namely the choice of programming language and the choice of algorithm, are not the most frequent in the literature, which is itself a finding worth acting on. Most notably, energy is measured almost universally while carbon is measured in only three studies, a gap that the Software Carbon Intensity specification makes readily closable and that the maturing regulatory landscape makes urgent.

The most impactful code level decisions identified in the literature, including choice of programming language, choice of algorithm, and data structure selection, offer immediate, quantifiable opportunities to reduce software energy consumption. A developer who consciously chooses a compiled language over an interpreted language can reduce energy consumption by an order of magnitude. A developer who selects an  $O(n \log n)$  algorithm in place of an  $O(n^2)$  algorithm can reduce energy consumption by orders of magnitude at scale. These findings provide an empirical foundation for green software development practices that can be taught in software engineering courses and applied in professional development contexts.

## 10.1 Limitations

This review has several limitations beyond the threats to validity discussed in Section 9. The restriction to English language publications, the rapid evolution of the field, and the absence of standardised energy measurement protocols across studies limit the degree to which quantitative findings can be directly synthesised through formal meta analysis. The decision framework and the guidelines, while grounded in the studies, will need to be revisited as the AI subset of the literature continues to grow and as standardisation efforts mature.

## 10.2 Future Work

Future work should focus on the six thematic areas identified in RQ5, with particular priority for definitional and conceptual foundations grounded in ISO/IEC 25010, low overhead assessment tools and IDE plugins, sustainable DevOps and energy aware testing, domain specific studies in game development and cloud native systems, explicit trade off models including energy debt, and machine learning and AI sustainability. Replication studies aimed at the high impact, low frequency rows of the framework in Section 5 would be especially valuable, as would studies that report carbon alongside energy and that record grid intensity context to support SCI based reanalysis.

## Statements and Declarations

**Funding.** This work was carried out at the NCRA UAV Dependability Lab, National University of Computer and Emerging Sciences, with the support of the National Centre of Robotics and Automation (NCRA), Pakistan. The funding source had no role in the study design, in the collection, analysis, and interpretation of data, in the writing of the report, or in the decision to submit the article for publication.

**Competing Interests.** The authors declare that they have no competing interests, financial or non financial, that could have appeared to influence the work reported in this paper.

**Ethics Approval.** Not applicable. This article is a systematic review of previously published literature and does not involve studies with human participants or animals performed by any of the authors.

**Data Availability.** The data supporting the findings of this review are available in the project repository at [https://github.com/ZoyaMahboob27/Sustainability\\_SLR](https://github.com/ZoyaMahboob27/Sustainability_SLR). The complete list of the 137 primary studies, with their quality tiers and full references, is provided in the Supplementary Material.

**Author Contributions.** Z.M. led the conceptualisation, methodology, data curation, and writing of the original draft. A.A.A.J. provided supervision and contributed to conceptualisation, methodology, and reviewing and editing. Q.U.A. contributed to data curation, validation, and reviewing and editing. All authors read and approved the final manuscript.

**Supplementary information.** The online version of this article contains supplementary material. Supplementary Material (SLR on Sustainability of Software), provides the complete list of the 137 primary studies included in this review, together with their quality assessment tiers and full bibliographic references.

## References

- Abdulsalam, S., Zong, Z., Gu, Q., Qiu, M.: Using the Greenup, Powerup, and Speedup Metrics to Evaluate Software Energy Efficiency. In: IEEE International Conference on Green Computing and Communications (2015 IEEE, ISBN 978-1-5090-0172-9) (2015)
- Alessi, F., Thoman, P., Georgakoudis, G., Fahringer, T., Nikolopoulos, D.S.: Application-Level Energy Awareness for OpenMP. In: IWOMP 2015: International Workshop on OpenMP (LNCS 9342, Springer) (2015)
- Aljabri, N., Al-Hashimi, M., Saleh, M., Abulnaja, O.: Investigating Power Efficiency of Mergesort. *The Journal of Supercomputing* (Springer), Volume 75, pages 6277–6302 (2019)
- Amalfitano, D., Metzger, A., Autili, M., Fulcini, T., Hey, T., Keim, J., Pelliccione, P., Scotti, V., Koziolk, A., Mirandola, R., Vogelsang, A.: A research roadmap for augmenting software engineering processes and software products with generative AI. *ACM Transactions on Software Engineering and Methodology* (2026). DOI: 10.1145/3788879
- Anistyasari, Y., Ekohariadi, Kurniawan, A.: Exploring computational thinking to improve energy-efficient programming skills. In: *Proceedings of the International Conference on Electrical Engineering, Computer Science and Informatics*. IEEE, (2018)

- Anwar, H., Pfahl, D., Srirama, S.N.: Evaluating the Impact of Code Smell Refactoring on the Energy Consumption of Android Applications. In: SEAA 2019: 45th Euromicro Conference on Software Engineering and Advanced Applications (2019)
- Anwar, H., Pfahl, D., Srirama, S.N.: An Investigation into the Energy Consumption of HTTP POST Request Methods for Android App Development. In: ICSOFT 2018: 13th International Conference on Software Technologies (2018)
- Babakol, T., Canino, A., Mahmoud, K.: Calm energy accounting for multithreaded Java applications. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. ACM, (2020)
- Balanza-Martinez, J., Lago, P., Verdecchia, R.: Tactics for software energy efficiency: A review. In: Advances and New Trends in Environmental Informatics 2023. Springer, (2024)
- Banerjee, A., Chong, L.K., Ballabriga, C., Roychoudhury, A.: EnergyPatch: Repairing Resource Leaks to Improve Energy-efficiency of Android Apps. IEEE Transactions on Software Engineering, Volume 44, Issue 5 (2018)
- Ben Chaaben, E., Koch, J., Mackay, W.E., Apel, S.: “Should I Choose a Smaller Model?”: Understanding ML model selection and its impact on sustainability. In: Proceedings of the ACM/IEEE International Conference on Software Engineering: Software Engineering in Practice. ACM, (2025)
- Brundtland, G., et al.: Our common future (Brundtland report). Technical report, United Nations World Commission on Environment and Development (1987). <http://www.un-documents.net/our-common-future.pdf>
- Carette, A., Younes, M.A., Hecht, G., Moha, N., Rouvoy, R.: Investigating the Energy Impact of Android Smells. In: SANER 2017: 24th IEEE International Conference on Software Analysis, Evolution and Reengineering (2017)
- Cheung, K.S., Kaul, M., Jahangirzadeh, G., Wachsmuth, W.: Comparative analysis of carbon footprint in manual vs. LLM-assisted code development. In: Proceedings of the International Conference on AI and Sustainability. ACM, (2025)
- Chowdhury, S.A., Sapra, V., Hindle, A.: Client-Side Energy Efficiency of HTTP/2 for Web and Mobile App Developers. In: SANER 2016: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (2016)
- Chowdhury, S.A., Nardi, M., Hindle, A., Jiang, Z.M.: An exploratory study on assessing the energy impact of logging on Android applications. Empirical Software Engineering (Springer) (2018)
- Chowdhury, S., Borle, S., Romansky, S., Hindle, A.: GreenScaler: Training Software Energy Models With Automatic Test Generation. Empirical Software Engineering

(Springer) (2019)

- Connolly Bree, D., Ó Cinnéide, M.: How software design affects energy performance: A systematic literature review. *Journal of Software: Evolution and Process* **37**, 70014 (2025). DOI: 10.1002/smr.70014
- Corral-Garcia, J., Lemus-Prieto, F., Gonzalez-Sanchez, J.-L., Perez-Toledano, M.-A.: Analysis of Energy Consumption and Optimization Techniques for Writing Energy-Efficient Code. *Electronics (MDPI)*, Volume 8, Issue 10, Article 1192, 2019 (2019)
- Couto, M., Saraiva, J., Fernandes, J.P.: Energy Refactorings for Android in the Large and in the Wild. In: *SANER 2020: 27th IEEE International Conference on Software Analysis, Evolution and Reengineering* (2020)
- Cruz, L., Abreu, R., Grundy, J., Li, L., Xia, X.: Do Energy-oriented Changes Hinder Maintainability? In: *ICSME 2019: IEEE International Conference on Software Maintenance and Evolution* (2019)
- Cunha, S., Silva, L., Saraiva, J., Fernandes, J.P.: Trading runtime for energy efficiency: Leveraging power caps to save energy across programming languages. In: *Proceedings of the ACM International Workshop on Green and Sustainable Software*. ACM, (2024)
- Danushi, O., Forti, S., Soldani, J.: Carbon-efficient software design and development: A systematic literature review. *ACM Computing Surveys* **57**(10), 267 (2025). DOI: 10.1145/3728638
- David, I.: SusDevOps: Promoting sustainability to a first principle in software delivery. In: *Proceedings of the ACM/IEEE International Conference on Software Engineering: New Ideas and Emerging Results*. ACM, (2025)
- Delfim, F.M., Pinto, G., Castor, F.: Assisting Non-Specialist Developers to Build Energy-Efficient Software. In: *ICSE-C 2017: 39th International Conference on Software Engineering Companion* (2017)
- Dhaka, G., Singh, P.V.: An Empirical Investigation into Code Smell Elimination Sequences for Energy Efficient Software. In: *APSEC 2016: 23rd Asia-Pacific Software Engineering Conference* (2016)
- Duran, P., Castaño, J., Gómez, C., Franch, X., Martínez-Fernández, S.: GAISSALabel: A tool for energy labeling of ML models. In: *Proceedings of the International Conference on Software Architecture*. IEEE, (2024)
- Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. *Information and Software Technology* **50**(9-10), 833–859 (2008)

- Fatima, I., Anwar, H., Pfahl, D., Minelli, R.: Detection and correction of Android-specific code smells and energy bugs: An Android Lint extension. In: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, (2020)
- Felix, P.S., Mohankumar, M.: Energy optimization in software development: A comparative study of sorting techniques. *International Journal of Computer Applications* **186**(14) (2024)
- Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G.S., Friday, A.: The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. *Patterns* **2**(9), 100340 (2021) <https://doi.org/10.1016/j.patter.2021.100340>
- Georgiou, S., Rizou, S., Spinellis, D.: Software Development Lifecycle for Energy Efficiency: Techniques and Tools. *ACM Computing Surveys (CSUR)*, Volume 52, Issue 4 (2019)
- Georgiou, S., Kechagia, M., Louridas, P., Spinellis, D.: What Are Your Programming Language’s Energy-Delay Implications? In: MSR 2018: 15th International Conference on Mining Software Repositories (2018)
- Gottschalk, M., Jelschen, J., Winter, A.: Refactorings for energy-efficiency. In: Proceedings of the 5th International Workshop on Requirements and Design in Education. Springer, (2016)
- Green Software Foundation: Software Carbon Intensity (SCI) Specification. <https://sci.greensoftware.foundation/>. ISO/IEC 21031:2024 (2024)
- Gupta, S., Gupta, A.: Advancing carbon-efficient software development: A sustainable path forward. In: Proceedings of the International Conference on Green and Sustainable Technology. IEEE, (2025)
- Hamizi, I., Bakare, A., Fraj, K., Bilami, A.: A meta-analytical comparison of energy consumed by two different programming languages. In: Proceedings of the International Conference on Systems and Informatics. IEEE, (2021)
- Hasan, S., King, Z., Hafiz, M., Sayagh, M., Adams, B., Hindle, A.: Energy Profiles of Java Collections Classes. In: ICSE ’16: Proceedings of the 38th International Conference on Software Engineering (2016)
- Hayduk, Y., Sobe, A., Felber, P.: Enhanced energy efficiency with the actor model on heterogeneous architectures. In: Proceedings of the 2016 ACM International Workshop on Programming Models and Applications for Multicores and Manycores. ACM, (2016)

- Hindle, A.: Green Mining: A Methodology of Relating Software Change and Configuration to Power Consumption. *Empirical Software Engineering* (Springer), Volume 20, pages 374–409, 2015 (2015)
- International Energy Agency: World energy outlook 2025. Technical report, International Energy Agency (IEA), Paris, France (2025). <https://www.iea.org/reports/world-energy-outlook-2025>
- Jabbarvand, R., Malek, S.:  $\mu$ Droid: An Energy-Aware Mutation Testing Framework for Android. In: *ESEC/FSE 2017: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (2017)
- Jabbarvand, R., Lin, J.-W., Malek, S.: Search-Based Energy Testing of Android. In: *ICSE 2019: 41st International Conference on Software Engineering* (2019)
- Jabbarvand, R., Mehralian, F., Malek, S.: Automated Construction of Energy Test Oracles for Android. In: *ESEC/FSE 2020: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2020)
- Jabbarvand, R., Sadeghi, A., Bagheri, H., Malek, S.: Energy-Aware Test-Suite Minimization for Android Apps. In: *ISSTA 2016: Proceedings of the 25th International Symposium on Software Testing and Analysis* (2016)
- Jagroep, E., Broekman, J., Werf, J.M.E.M., Brinkkemper, S., Lago, P., Blom, L., Vliet, R.: Awakening Awareness on Energy Consumption in Software Engineering. In: *ACM 39th International Conference on Software Engineering: Software Engineering in Society Track (ICSE-SEIS)* (2017)
- Jalili, M., Faghih, F.: Static/dynamic analysis of Android applications to improve energy-efficiency. In: *Proceedings of the 5th International Conference on Web Research*. IEEE, (2022)
- Kholmatova, Z.: Impact of Programming Languages on Energy Consumption for Mobile Devices. In: *ESEC/FSE 2020: 28th ACM Joint European Software Engineering Conference (Short Paper)* (2020)
- Kim, C., Choi, M., Hong, J.: Evaluating energy efficiency of IoT software architecture based on reusable software components. *International Journal of Distributed Sensor Networks*, Volume 13, Issue 1 (2017)
- Kitchenham, B.A., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-01, EBSE (2007)
- König, C., Lang, D.J., Schaefer, I.: Sustainable software engineering: Concepts, challenges, and vision. *ACM Transactions on Software Engineering and Methodology* **34**(5) (2025)

- Lago, P., Koçak, S.A., Crnkovic, I., Penzenstadler, B.: Framing sustainability as a property of software quality. *Communications of the ACM* **58**(10), 70–78 (2015)
- Lima, L.G., Soares-Neto, F., Lieuthier, P., Castor, F., Melfe, G., Fernandes, J.P.: Haskell in Green Land: Analyzing the Energy Behavior of a Purely Functional Language. In: *SANER 2016: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering* (2016)
- Lima, L.G., Soares-Neto, F., Lieuthier, P., Castor, F., Melfe, G., Fernandes, J.P.: On Haskell and Energy Efficiency. *Journal of Systems and Software* (Elsevier) (2019)
- Liu, K., Pinto, G., Liu, Y.D.: Data-Oriented Characterization of Application-Level Energy Optimization. In: *FASE 2015: Fundamental Approaches to Software Engineering* (LNCS 9033, Springer-Verlag) (2015)
- Mahadevappa, S., Figueira, S.: Energy-efficient programming languages for mobile applications. In: *Proceedings of the 12th International Conference on Green, Pervasive, and Cloud Computing*. Springer, (2021)
- Maia, D., Couto, M., Saraiva, J., Pereira, R.: E-Debitum: Managing Software Energy Debt. In: *ASE '20: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (2020)
- Mancebo, J., Calero, C., Garcia, F., Moraga, M.A., Guzman, I.G.-R.: FEETINGS: Framework for Energy Efficiency Testing to Improve Environmental Goal of the Software. *Sustainable Computing: Informatics and Systems* (Elsevier), Volume 30, 2021 (2021)
- Manikyala, A.: Code refactoring for energy-saving distributed systems: A data analytics approach. *International Journal of Innovative Technology and Exploring Engineering* **13**(7) (2024)
- Maquoi, J., Cauz, M., Vanderhoofstadt, B., Massonnet, P., Mens, T.: Energy codesumption: Leveraging test execution for source code energy consumption analysis. In: *Proceedings of the ACM International Conference on Mining Software Repositories*. ACM, (2025)
- McIntosh, A., Hassan, S., Hindle, A.: What Can Android Mobile App Developers Do About the Energy Consumption of Machine Learning? *Empirical Software Engineering*, Volume 24, Issue 1 (2019)
- Michanan, N., Dewri, R., Rutherford, M.J.: Predicting Data Structures for Energy Efficient Computing. In: *IGSC 2015: 6th International Green and Sustainable Computing Conference* (2015)
- Misu, M.R.H., Li, J., Bhattiprolu, A., Liu, Y., Almeida, E.S., Ahmed, I.: Test smell: A parasitic energy consumer in software testing. *Information and Software Technology*

- Moises, A.C., Malucelli, A., Reinehr, S.: Practices of energy consumption for sustainable software engineering. In: Proceedings of the 14th International Conference on ICT for Sustainability. Springer, (2018)
- Mourão, E., Pimentel, J.F., Murta, L., Kalinowski, M., Mendes, E., Wohlin, C.: On the performance of hybrid search strategies for systematic literature reviews in software engineering. *Information and Software Technology* **123**, 106294 (2020)
- Munoz, D.-J., Pinto, M., Fuentes, L.: Green software development and research with the HADAS toolkit. In: Proceedings of the 12th International Conference on ICT for Sustainability. Springer, (2017)
- Oliveira, W., Oliveira, R., Castor, F.: A Study on the Energy Consumption of Android App Development Approaches. In: MSR '17: Proceedings of the 14th IEEE/ACM International Conference on Mining Software Repositories (2017)
- Oliveira, W., Oliveira, R., Castor, F., Pinto, G., Fernandes, J.P.: Improving Energy-Efficiency by Recommending Java Collections. *Empirical Software Engineering*, Volume 26, Issue 55 (2021)
- Ournani, Z., Rouvoy, R., Rust, P., Penhoat, J.: On Reducing the Energy Consumption of Software: From Hurdles to Requirements. In: ESEM 2020: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (2020)
- Ournani, Z., Belgaid, M.C., Rouvoy, R., Rust, P., Penhoat, J.: Evaluating the Impact of Java Virtual Machines on Energy Consumption. In: ESEM 2021: 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (2021)
- Page, M.J., McKenzie, J.E., Bossuyt, P.M., Boutron, I., Hoffmann, T.C., Mulrow, C.D., Shamseer, L., Tetzlaff, J.M., Akl, E.A., Brennan, S.E., Chou, R., Glanville, J., Grimshaw, J.M., Hróbjartsson, A., Lalu, M.M., Li, T., Loder, E.W., Mayo-Wilson, E., McDonald, S., McGuinness, L.A., Stewart, L.A., Thomas, J., Tricco, A.C., Welch, V.A., Whiting, P., Moher, D.: The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ* **372**, 71 (2021) <https://doi.org/10.1136/bmj.n71>
- Palomba, F., Nucci, D.D., Panichella, A., Zaidman, A., Lucia, A.D.: On the Impact of Code Smells on the Energy Consumption of Mobile Applications. *Information and Software Technology* (Elsevier), Volume 105 (2019)
- Pathania, P., Mehra, R., Sharma, V.S., Kaulgud, V.: Towards a knowledge base of common sustainability weaknesses in green software development. In: Proceedings of the ACM International Conference on Software Engineering: New Ideas and Emerging

- Results. ACM, (2024)
- Penzenstadler, B.: Towards a definition of sustainability in and for software engineering. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing. SAC '13, pp. 1183–1185. ACM, (2013)
- Pereira, R., Carção, T., Couto, M., Cunha, J., Saraiva, J., Fernandes, J.P.: SPELLing out energy leaks: Aiding developers locate energy inefficient code. *Science of Computer Programming* **122**, 83–100 (2016)
- Pereira, R., Couto, M., Ribeiro, F.J., Rua, R.A., Cunha, J., Fernandes, J.P., Saraiva, J.: 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. SLE '17, pp. 256–267. ACM, (2017a)
- Pereira, R., Carção, T., Couto, M., Cunha, J., Fernandes, J.P., Saraiva, J.: Helping Programmers Improve the Energy Efficiency of Source Code. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C) (2017b)
- Pereira, R., Simão, P., Cunha, J., Saraiva, J.: jStanley: Placing a Green Thumb on Java Collections. In: ASE 2018: Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (2018)
- Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., Saraiva, J.: Ranking Programming Languages by Energy Efficiency. *Science of Computer Programming Volume 205*, 1 May 2021, 102609 (2021)
- Pereira, R., Couto, M., Cunha, J., Saraiva, J., Fernandes, J.P.: Paint your programs green: On the energy efficiency of data structures. In: Proceedings of the ACM International Workshop on Green and Sustainable Software. ACM, (2023)
- Poy, O., Moraga, M.Á., García, F., Calero, C.: Impact on energy consumption of design patterns, code smells and refactoring techniques: A systematic mapping study. *Journal of Systems and Software* **222**, 112303 (2025). DOI: 10.1016/j.jss.2024.112303
- Procaccianti, G., Fernández, H., Lago, P.: Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software* **117**, 185–198 (2016)
- Roberts, S., Wright, S., Lecomber, D., January, C., Byrd, J., Oró, X., Jarvis, S.: POSE: A Mathematical and Visual Modelling Tool to Guide Energy Aware Code Optimisation. In: IGSC 2015: 6th International Green and Sustainable Computing Conference (2015)
- Rocha, G., Castor, F., Pinto, G.: Comprehending Energy Behaviors of Java I/O APIs. In: ESEM 2019: 13th ACM/IEEE International Symposium on Empirical Software

Engineering and Measurement (2019)

Rodriguez, C., Degioanni, L., Kameni, L., Vidal, R., Neglia, G.: Evaluating the energy consumption of machine learning: Systematic literature review and experiments. arXiv preprint arXiv:2408.15128 (2024)

Roque, E.B., Cruz, L., Durieux, T.: Unveiling the energy vampires: A methodology for debugging software energy consumption. In: Proceedings of the ACM/IEEE International Conference on Software Engineering. ACM, (2025)

Shanbhag, S., Chimalakonda, S., Sharma, V.S., Kaulgud, V.: Towards a catalog of energy patterns in deep learning development. In: Proceedings of the ACM International Workshop on Green and Sustainable Software. ACM, (2022)

Song, W., Zhang, J., Huang, J.: ServDroid: Detecting Service Usage Inefficiencies in Android Applications. In: ESEC/FSE 2019: 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (2019)

Stokke, K.R., Strømmen-Bakhtiar, H.K., Kvakkestad, A.H.: A high-precision, hybrid GPU, CPU and RAM power model for generic multimedia workloads. In: Proceedings of the ACM International Conference on Multimedia. ACM, (2016)

Taherikhonakdar, Z., Fazlollahta, H.: Proposing the Method for Operating System Energy Consumption Measurement with Energy Efficiency Approach. Research Report (2023)

Toosi, F.G.: Green software engineering: A dual-perspective overview of stakeholder and societal interpretations. In: 2025 Computing, Communications and IoT Applications (ComComAp), pp. 385–394. IEEE, (2026). DOI: 10.1109/ComComAp68359.2025.11353190

Uddin, M., Darabidarabkhani, Y., Shah, A., Memon, J.: Evaluating Power Efficient Algorithms for Efficiency and Carbon Emissions in Cloud Data Centers: A Review. Renewable and Sustainable Energy Reviews, Volume 51 (Elsevier) (2015)

Gastel, P., Gastel, B., Moonen, L.: Detecting energy bugs and hotspots in control software using model checking. In: Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE '18. ACM, (2018)

Vemuri, V.P.K.: Greening the web: Front-end optimization for reducing energy consumption and carbon footprints. International Journal of Computer Science and Technology (2021)

Verdecchia, R., Saez, R.A., Procaccianti, G., Lago, P.: Empirical Evaluation of the Energy Impact of Refactoring Code Smells. In: ICT4S 2018: 5th International

Conference on ICT for Sustainability (2018)

Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE). ACM, (2014)

Wohlin, C., Kalinowski, M., Romero Felizardo, K., Mendes, E.: Successful combination of database search and snowballing for identification of primary studies in systematic literature studies. *Information and Software Technology* **147**, 106908 (2022)

Öztürk, M.M.: Deep Learning-Based Software Energy Consumption Profiling. In: ICAIAME 2019: Artificial Intelligence and Applied Mathematics in Engineering Problems, Lecture Notes on Data Engineering and Communications Technologies, Volume 43, Springer, 2020 (2020)

Şanlıalp, Öztürk, M.M.: Investigating the Impact of Code Refactoring Techniques on Energy Consumption in Different Object-Oriented Programming Languages. In: ICAIAME 2019: Artificial Intelligence and Applied Mathematics in Engineering Problems, Lecture Notes on Data Engineering and Communications Technologies, Volume 43, Springer, 2020 (2020)

## Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- [SupplementaryMaterialSLRonSustainabilityofSoftware.pdf](#)