Sustainability Concerns in Open Source Software: A Multi-Domain Mining Study of GitHub Issues

Eduardo Almentero
UFRRJ
Seropédica, Brazil
almentero@ufrrj.br

Henrique Sousa
UNIRIO
Rio de Janeiro, Brazil
hsousa@uniriotec.br

Roxana Portugal
UNSAAC
Cuzco, Peru
roxana.portugal@ifkw.lmu.de

Marcos Kalinowski
PUC-Rio
Rio de Janeiro, Brazil
kalinowski@inf.puc-rio.br

Abstract

Despite growing interest in Green Software Engineering, little is known about how practitioners engage with sustainability concerns during day-to-day development. This paper presents a large-scale empirical study mining 94,915 closed GitHub issues from 50 open source repositories across five domains: Systems, Backend, ML/Data Science, Frontend, and Mobile. We apply a two-step filtering protocol combining a synonym-aware keyword taxonomy grounded in Green SE principles with a domain-safe noise filter, followed by LLM-assisted classification (Cohen's $\kappa = 0.746$), reducing 6,849 candidates to 4,386 confirmed sustainability-related issues. Our analysis addresses four research questions on category distribution, domain variation, report quality, and abandonment patterns. Developers rarely use energy-specific vocabulary (only 0.3% of sustainability issues explicitly mention energy or power), preferring proxy terms such as memory leaks, slowdowns, and CPU spikes. Systems repositories show the highest sustainability issue rate (5.9%) and Mobile the lowest (3.0%), with sustainability concerns concentrated in domains where resource management is closest to the hardware. Sustainability issues carry more numeric measurements and reproduction steps but fewer code blocks and version references than non-sustainability issues. Most critically, they go stale at 2.28× the rate of non-sustainability issues (5.4% vs. 2.4%, $p < 0.001$), and stale issues persist for a median of 182 days before closure versus 7 days for resolved issues (Cliff's $\delta = 0.748$, large effect). These findings suggest sustainability concerns are deprioritized in OSS issue management, and that the gap between research and practitioner vocabulary may impede effective sustainability triage.

Keywords

Green Software Engineering, Mining Software Repositories, Software Sustainability, GitHub Issues, Empirical Software Engineering, Issue Abandonment, Energy Efficiency, Open Source Software

1 Introduction

The environmental impact of software has become an increasingly urgent concern. The information and communication technology (ICT) sector is estimated to account for 2–4% of global greenhouse gas emissions [10], a figure projected to grow as software-intensive systems proliferate across every domain of the economy. Within this broader picture, the energy consumed by software during execution through inefficient algorithms, memory leaks, redundant network calls, and unresponsive processing loops represents a substantial and largely addressable share of the ICT footprint [29].

Green Software Engineering (Green SE) has emerged as the research discipline addressing these concerns, defining principles and practices for developing software that minimizes energy consumption and resource waste across its entire lifecycle [5, 9]. A growing body of work has proposed frameworks, metrics, measurement tools, and coding guidelines for green software [4, 5, 29]. Despite this growing recognition, however, a notable lack of consensus exists regarding specific definitions and comprehensive strategies for sustainability within software engineering [11, 19]. This disagreement extends both to core terminology and to which sustainability dimensions are addressed: the environmental dimension, and energy efficiency in particular, dominates the literature, while the social, economic, and individual dimensions receive comparatively little attention [11]. This gap highlights the necessity for empirical investigations into how these nascent green software engineering principles translate into practical development workflows, particularly within the transparent and collaborative environment of open-source projects [5].

Yet a fundamental question remains underexplored: *to what extent do software developers in practice actually engage with sustainability concerns during the day-to-day work of building and maintaining open source software?*

Understanding practitioner engagement is essential for two reasons. First, Green SE tools and guidelines can only be effective if they align with how developers already perceive and describe sustainability problems. If the vocabulary, categories, and concerns of research differ from those of practitioners, even well-designed tools will fail to reach their intended users. Second, understanding how sustainability issues are reported, prioritized, and resolved, or abandoned, provides a baseline against which the effectiveness of any intervention can be measured.

GitHub issue trackers offer a unique window into this question. Issues are the primary mechanism through which OSS developers report observed problems, request improvements, and coordinate fixes. They are rich with natural language descriptions of developer concerns, labeled with project-defined categories, annotated with response times, and linked to the code changes that address them. Mining this artefact at scale allows us to characterize practitioner

sustainability engagement as it actually occurs, rather than as it is reported in surveys or prescribed in guidelines.

In this paper, we present a large-scale empirical study mining **94,915 closed GitHub issues** from **50 OSS repositories** spanning five software domains: Systems, Backend, ML/Data Science, Frontend, and Mobile. We apply a two-step filtering protocol that combines a synonym-aware keyword taxonomy grounded in the Green SE framework of Freed et al. [9] with a domain-safe noise filter, complemented by LLM-assisted classification (Cohen's $\kappa = 0.746$), to identify and categorize **4,386 sustainability-related issues**. We then investigate four research questions addressing the distribution, domain variation, report quality, and abandonment patterns of these issues.

Our results form a consistent picture. Developers rarely use energy-specific vocabulary, with only 0.3% of sustainability issues explicitly mentioning energy or power, instead relying on proxy terms such as memory leaks, slowdowns, and CPU spikes. Sustainability concerns are most prevalent in Systems repositories (5.9%) and least prevalent in Mobile repositories (3.0%), with ML/Data Science in an intermediate position (5.2%). Sustainability issue reports tend to be more quantitative than non-sustainability issue reports but less anchored to a specific build and code path, as developers more often record what they measured than the environment in which they measured it. Most critically, sustainability issues are abandoned at **2.28× the rate of non-sustainability issues** (5.4% vs. 2.4%), with stale issues persisting for a median of **182 days** before closure compared to just 7 days for resolved issues (Cliff's $\delta = 0.748$, large effect), suggesting that sustainability concerns may be less consistently addressed in OSS communities.

We operationalize sustainability concerns as performance and resource-related issues, such as memory consumption, CPU usage, and latency, which serve as observable proxies for energy efficiency. Not every such issue translates directly into energy impact, but prior work links these inefficiencies to increased energy consumption [29], and Green Software Engineering recognizes them as primary drivers of software energy use. Section 3 gives the precise operational definition.

The contributions of this paper are:

- A validated keyword taxonomy of ten sustainability-related issue categories grounded in Green SE principles [9], covering over 100 synonym-aware patterns (Section 3.3);
- A reproducible LLM-assisted classification protocol combining keyword filtering with Claude Haiku and manual validation, achieving substantial inter-rater agreement ($\kappa = 0.746$) (Section 3.4);
- The first large-scale, multi-domain empirical characterization of sustainability issue engagement in OSS, covering 50 repositories across 5 domains and 94,915 issues (Section 4);
- Empirical evidence consistent with sustainability concerns being relatively deprioritized in OSS communities, as indicated by a 2.28× higher abandonment rate and longer stale durations (Section 4.4).

All scripts, datasets, classification guidelines, and analysis code are available in a public replication package.[1]

[1] https://doi.org/10.5281/zenodo.21400528

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the study design. Section 4 presents the results. Section 5 discusses the findings. Section 6 addresses threats to validity. Section 7 presents conclusions and future work.

2 Related Work

We organize related work around three themes that directly inform our study: Green Software Engineering frameworks and principles; empirical studies of energy and performance concerns in software repositories; and the use of LLMs for software engineering annotation tasks.

2.1 Green Software Engineering

The field of Green Software Engineering (Green SE) has grown substantially over the past decade. Calero and Piattini [5] established a foundational taxonomy positioning resource efficiency as a first-class software quality attribute, while Verdecchia et al. [29] situated software-level interventions within the broader Green IT landscape. More recently, Freed et al. [9] synthesized Green SE into four core principles (energy efficiency, sustainable design, life cycle assessment, and renewable energy sources), the first three of which provide the conceptual foundation for our issue taxonomy. Despite this progress, König et al. [19] argue that Green SE remains fragmented, lacking systematic approaches and consistent definitions, which complicates empirical research and practical adoption.

Empirical work has investigated how design choices affect energy consumption. Connolly et al. [8] review studies on design patterns, code smells, and refactorings across 2010–2023, finding results to be contradictory and context-dependent. Palomba et al. [24] show that Android-specific code smells increase energy consumption and that targeted refactorings mitigate these effects. Ahmadisakha and Andrikopoulos [1] complement this by showing that practitioners on Stack Exchange frame sustainability in terms of cost and operational efficiency rather than explicit energy terminology, consistent with König et al.'s [19] observation of conceptual ambiguity and with our own findings on proxy vocabulary in issue trackers.

2.2 Mining Energy and Performance Concerns in Repositories

The MSR community has produced a growing body of work on energy-related concerns in software artefacts. Pinto et al. [25] analyze 325 Stack Overflow questions about software energy consumption, identifying five themes: measurements, code design, context-specific concerns, general knowledge, and hardware. Extending this line of work, Jin et al. [17] conduct a multi-dimensional empirical analysis of energy-efficient software development on Stack Overflow, reporting an increasing focus on cloud and IoT energy concerns. While valuable, both studies are limited to explicit energy queries on a question-and-answer platform, potentially overlooking the broader and more implicit ways in which sustainability concerns manifest in practice, a limitation consistent with the fragmented approaches noted by König et al. [19].

At the code and repository level, Linares-Vásquez et al. [21] mine energy-greedy API patterns in Android apps, establishing green

mining as a viable research direction. Albonico et al. [2] analyze robotics repositories for sustainability practices, and Cannizza and Albonico [4] catalogue open-source energy measurement tools on GitHub. These studies mine code, commits, or tool artefacts within a single domain, whereas we mine issue-tracker discussions across five domains, capturing how developers describe sustainability concerns rather than how tools measure them. Complementing these, Mazuera-Rozo et al. [23] define a taxonomy of performance bug types in Android and iOS apps and study their survivability (days between bug introduction and fix), a design closely related to our RQ4 abandonment analysis. The concept of Green Mining, introduced by Hindle et al. [15], motivated indirect, proxy-based approaches such as ours, which treats issue tracker discussions as sustainability indicators complementing direct measurement methods.

Our study differs from these Stack Overflow analyses [17, 25] along five dimensions: (i) *platform*, mining GitHub issue trackers rather than Stack Overflow; (ii) *scale*, with 94,915 issues across 50 repositories; (iii) *vocabulary*, using a broader synonym-aware taxonomy that captures proxy terms such as memory, CPU, and latency, revealing a practitioner vocabulary gap; (iv) *domain scope*, spanning five domains to enable cross-domain comparisons; and (v) *lifecycle perspective*, examining resolution dynamics and abandonment patterns. Rather than directly extending prior work, this study complements it by examining a different development artefact (GitHub issue trackers) and capturing a broader practitioner vocabulary around sustainability concerns, aligned with recent calls for more systematically grounded approaches [19].

Issue lifecycle and triage provide theoretical grounding for our RQ4. Shihab et al. [28] show that issues requiring deeper system knowledge are more likely to be reopened, and Giger et al. [12] demonstrate that issue type and complexity influence resolution time, two findings that support our investigation of sustainability issues as harder-to-resolve artefacts. Mazuera-Rozo et al. [23] further show that performance bugs in mobile apps survive significantly longer than functional bugs, a finding that anticipates and corroborates our own stale duration results (182 days vs. 7 days).

2.3 LLM-Based Annotation in Software Engineering

The use of large language models for text classification and annotation in software engineering has expanded rapidly since 2022. Gilardi et al. [13] show that ChatGPT can outperform crowd workers on several text annotation tasks, suggesting that LLMs are viable annotators for empirical SE studies. Imran and Zaman [16] introduce OLAF, a methodological framework for LLM-based annotation in empirical software engineering, highlighting the importance of confidence calibration, human oversight of low-confidence cases, and inter-rater agreement reporting, all of which inform our classification protocol.

In the context of issue classification specifically, recent work has applied LLMs to distinguish bug reports, feature requests, and questions in OSS issue trackers [3]. Our use of Claude Haiku for sustainability classification differs from prior work in two ways: we use a multi-class taxonomy derived from domain-specific theory (Freed et al. [9]) rather than generic issue type labels, and we integrate confidence-based routing that directs uncertain cases to human review rather than accepting all LLM outputs unconditionally. This hybrid approach yields $\kappa = 0.746$, comparing favorably with agreement rates reported in related annotation studies [13].

Table 1: Positioning of this study relative to related work. ✓ = addressed; ○ = partially; – = not.

Study	Issue track.	Multi-domain	Theory ground.	LLM class.	Aban-don.
Pinto [25]; Jin [17]	–	–	○	–	–
Albonico [2]	○	–	–	–	–
Ahmadisakha [1]	–	–	○	–	–
Cannizza [4]	–	–	–	–	–
Mazuera-Rozo [23]	○	–	–	–	○
This study	✓	✓	✓	✓	✓

2.4 Positioning of This Work

Table 1 contrasts our study with the most closely related prior work across five dimensions: issue-tracker mining, multi-domain scope, theory-grounded taxonomy, LLM-assisted classification, and abandonment analysis. No prior study addresses all five simultaneously; ours is the first to combine them.

3 Study Design

This section describes the empirical methodology adopted to collect, filter, classify, and analyze GitHub issues related to energy efficiency, resource consumption, and performance in open source software (OSS) projects. The study follows a quantitative Mining Software Repositories (MSR) approach [14].

Operational definition. In this study, a **sustainability-related issue** is a GitHub issue whose primary concern is the software's runtime energy or resource efficiency, specifically memory consumption, CPU or compute utilization, execution time and latency, or redundant I/O and network activity, where the reported inefficiency plausibly increases the energy footprint of executing the software. We adopt these performance and resource concerns as observable proxies for energy efficiency, following the established link between computational inefficiency and energy consumption [29]. We do not include social or economic dimensions of sustainability, nor maintainability concerns that lack a runtime-efficiency component.

Guided by this definition, the study is structured around four research questions:

- **RQ1:** How are sustainability-related concerns distributed across issue categories in OSS projects?
- **RQ2:** How do sustainability concerns vary across software domains?
- **RQ3:** How does the technical detail of sustainability issue reports differ from that of non-sustainability issue reports?
- **RQ4:** Are sustainability issues more likely to be abandoned than non-sustainability issues?

Figure 1 presents an overview of the five-step pipeline, whose components are described in the following subsections.

3.1 Repository Selection

We selected 50 GitHub repositories using a stratified sampling strategy designed to maximize diversity across programming language ecosystems and application domains. Five strata were defined: Systems/Low-level, Backend/Server, ML/Data Science, Frontend/JavaScript, and Mobile. Ten repositories were selected per stratum, weighting each domain equally in the cross-domain comparisons of RQ2 and RQ3 at the cost of not reflecting relative ecosystem size.

Repositories were required to satisfy all of the following inclusion criteria: (i) at least 500 closed issues, ensuring a sufficient data volume for statistical analysis; (ii) at least three years of development history; (iii) GitHub Issues enabled and actively used, as some mature projects manage contributions through external trackers such as Jira or mailing lists; and (iv) at least 100 GitHub stars, indicating community relevance and active maintenance.

During data collection, several initially selected repositories proved to have GitHub Issues disabled. The most notable case is `torvalds/linux`, which manages all contributions through the Linux kernel mailing list despite being one of the most active OSS projects in the world. These repositories were replaced with functionally equivalent alternatives meeting all inclusion criteria. Table 2 summarizes the final sample of 50 repositories, organized by domain.

3.2 Data Collection

Issue data were collected programmatically using the GitHub REST API (v2022-11-28). For each repository, we retrieved up to 2,000 closed issues ordered by most recent update. Only closed issues were collected, as they provide a definitive time-to-resolution (TTR) measurement. Pull requests, which the GitHub Issues endpoint also returns, were excluded by detecting the presence of the `pull_request` field in the API response.

For each issue we recorded: title, body text, labels, comment count, reporter account and type (User vs. Bot), creation and closing timestamps, number of assignees, and whether a linked pull request existed. Repository-level metadata (primary language, stars, forks, age) were also collected and joined to each issue record. TTR was computed in decimal days as the difference between the `closed_at` and `created_at` timestamps.

The GitHub REST API enforces a hard ceiling of 10,000 items per paginated query (100 pages of 100 items). Since the issues endpoint interleaves issues and pull requests in a shared pagination sequence, repositories with large pull request volumes yield fewer than 2,000 issues before the ceiling is reached. This affected three repositories in our sample (`rust-lang/rust`: 1,862; `curl/curl`: 1,634; `facebook/react-native`: 1,341), all of which have substantially more closed issues. Because the API returns results ordered by most recent update, this constraint introduces a potential recency bias for those three repositories, which we acknowledge as a threat to construct validity. All other repositories were collected without truncation.

Data were saved incrementally to disk after each repository was processed, enabling automatic resumption in case of network interruption or API rate-limit exhaustion. The full corpus comprised **94,915 closed issues** across 50 repositories.

3.3 Issue Identification Protocol

We applied a two-step filtering protocol to identify sustainability-related issues from the full corpus.

3.3.1 Step 1 — Keyword Filter. The first filter applies a broad, synonym-aware keyword taxonomy to the concatenated title and body of each issue. Following the Green Software Engineering framework proposed by Freed et al. [9], we defined ten issue categories organized around three of its four principles, excluding renewable energy sources because energy sourcing is an operator decision that leaves no trace in issue trackers: *Energy Efficiency* (Memory Consumption, CPU Consumption, Energy & Power Usage, Computational Overhead), *Sustainable Design* (Responsiveness Degradation, I/O & Network Overhead, General Resource Waste), and *Life Cycle Assessment* (Performance Regression, Latency & Response Time, Profiling & Benchmarking). Each category was mapped to a set of synonyms and linguistic variants expressed as regular expressions with word-boundary anchors to prevent false substring matches. As a representative example, the Memory Consumption category includes terms such as *memory leak*, *heap growth*, *OOM*, *RAM usage*, and *retained objects*. The remaining categories follow the same structure. The full taxonomy comprises over 100 patterns across all ten categories and is available in the replication package. An issue is flagged as a candidate if its text matches at least one pattern from at least one category.

3.3.2 Step 2 — Noise Filter. The keyword filter maximizes recall at the cost of precision, capturing support questions, environment-specific complaints, and bot-generated issues that contain sustainability terms incidentally. A second filter removes these using five domain-safe exclusion rules: issues reported by bots are excluded; issues with bodies shorter than 80 characters are excluded; issues labeled as questions, help requests, or duplicate are excluded; issues whose titles follow question-style patterns (e.g., *How to*, *What is*) are excluded; and issues containing only environment-specific complaints (e.g., *works on my machine*) are excluded. Critically, the filter does not require stack traces or structured reproduction steps, which would systematically disadvantage Mobile and Frontend repositories where such artefacts rarely appear in issue reports. After both steps, **6,849 candidate issues** remained for classification.

Each candidate also receives a quality score on a 0–4 scale based on the presence of four signals: numeric measurements with units (e.g., 500 ms, 100% CPU), reproduction steps, code blocks, and version references. This score is not used as an exclusion criterion; rather, it is stored as a variable for subsequent analysis (RQ3).

3.4 LLM-Assisted Classification

Each candidate issue was submitted to a large language model (LLM) for fine-grained classification. We used Claude Haiku (claude-haiku-4-5, Anthropic) via the Messages API, which offers a favorable balance between classification accuracy and inference cost for large-scale text classification tasks [13].

Each API call received a structured prompt containing the issue title and body (truncated to 600 characters), the keyword cluster that triggered the Step 2 match, and the definitions of all ten categories plus a *not_relevant* category to capture false positives. The category definitions supplied to the model operationalize the notion

Table 2: Repository sample stratified by domain (10 repos per domain). The full list is available in the replication package.

Domain	Languages	Representative repos	Issues	Sustainability Issues
Systems	C, Go, Rust	redis, rust-lang/rust, duckdb	19,254	1,141
Backend	Java, Go	netty, golang/go, quarkus	19,310	953
ML/Data	Python, Scala	pandas, transformers, xgboost	19,138	997
Frontend	JavaScript, TypeScript	nodejs/node, react, vscode	19,510	765
Mobile	Kotlin, Java, Dart	flutter, okhttp, glide	17,703	530
Total		**50 repositories**	**94,915**	**4,386**

of a sustainability-related issue introduced in Section 3, restricting positive labels to runtime energy and resource-efficiency concerns rather than performance issues in general. The model was instructed to respond exclusively in JSON with four fields: `category`, `confidence` (high, medium, or low), `justification` (one sentence), and `is_relevant` (boolean). High-confidence responses were accepted automatically. Medium- and low-confidence responses were routed to a manual review queue, where the first author examined each issue and assigned the final label. **208 issues** required manual review. After classification, **4,386 issues** were identified as sustainability-related.

3.5 Validation

Two offline validation procedures were conducted to assess the quality of the filtering protocol.

Keyword filter recall. To estimate the false negative rate of the keyword filter, a stratified random sample of 200 issues rejected in Step 2 was manually inspected by the first author. Eleven issues (5.5%) were found to be genuinely relevant but missed by the keyword patterns, indicating a false negative rate of 5.5% and an estimated recall of 94.5%. This rate is consistent with thresholds reported in related MSR studies [18] and is acknowledged as a threat to construct validity in Section 6. Whereas the category structure and its mapping to Green SE principles are derived directly from the framework of Freed et al. [9], providing theoretical grounding for the taxonomy rather than ad-hoc keyword selection, this recall estimate assesses its empirical coverage.

LLM classification agreement. To validate the LLM classification, all 208 medium- and low-confidence issues were manually reviewed by the first author. The LLM and the researcher agreed on 164 out of 208 cases (78.8%). Inter-rater agreement was assessed using Cohen's Kappa [7], yielding $\kappa = 0.746$, indicating substantial agreement [20]. The primary source of disagreement was the LLM over-applying the Profiling & Benchmarking category (LLM: 70, researcher: 45). In all cases of disagreement, the researcher's label was used as the final classification. The complete prompt template, including the definitions of all ten categories and the *not_relevant* category supplied to the model, is available in the replication package.

3.6 Statistical Analysis

To answer RQ3 and RQ4, we compare sustainability issues against a control group comprising all 2,463 candidate issues that passed the two-step filtering pipeline but were classified as *not_relevant* by the LLM. Because these issues matched the sustainability keyword taxonomy yet were judged non-sustainability-related, they form a topically adjacent within-corpus baseline drawn from the same 50 repositories, rather than an arbitrary sample of unrelated issues. Given the expected non-normal distributions of continuous variables, we employ the two-tailed Mann-Whitney U test for all group comparisons, reporting effect sizes via Cliff's δ [6]. We interpret effect magnitudes following Romano et al. [26]: negligible ($|\delta| < 0.147$), small (< 0.33), medium (< 0.474), and large (≥ 0.474). A significance threshold of $\alpha = 0.05$ is used throughout. For binary outcomes (RQ4 stale rate), we apply the chi-square test of independence. All analyses were conducted using Python 3.14 with the `pandas`, `scipy`, and `matplotlib` libraries. Analysis scripts are included in the replication package.

Table 3: Distribution of sustainability-related issues by category (RQ1).

Green SE Principle	Category	n	%
Energy Efficiency	Memory Consumption	1,393	31.8
	CPU Consumption	200	4.6
	Computational Overhead	415	9.5
	Energy & Power Usage	11	0.3
Sustainable Design	Responsiveness Degradation	1,009	23.0
	I/O & Network Overhead	160	3.6
	General Resource Waste	108	2.5
Life Cycle Assessment	Performance Regression	367	8.4
	Latency & Response Time	290	6.6
	Profiling & Benchmarking	433	9.9
Total		**4,386**	**100.0**

4 Results

This section reports the results of applying the pipeline described in Section 3 to the corpus of 94,915 closed issues collected from 50 OSS repositories. After keyword filtering, noise removal, and LLM-assisted classification, **4,386 issues** (4.6% of the full corpus) were identified as sustainability-related. We report results for each of the four research questions in turn.

4.1 RQ1 — Category Distribution

Table 3 presents the distribution of sustainability-related issues across the ten categories of our taxonomy, grouped by Green SE

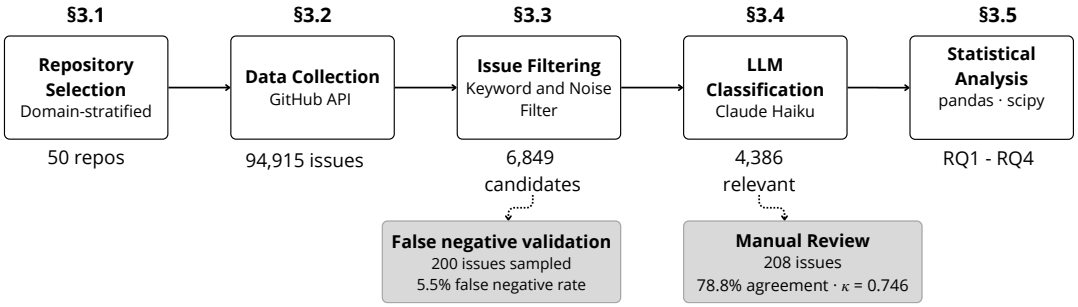

Figure 1: Overview of the study pipeline aligned with paper sections.

principle [9]. Two categories account for the majority of all identified issues: *Memory Consumption* (1,393 issues, 31.8%) and *Responsiveness Degradation* (1,009 issues, 23.0%), together representing over half the dataset. Issues related to *Profiling & Benchmarking* (9.9%) and *Computational Overhead* (9.5%) form a secondary tier, followed by *Performance Regression* (8.4%) and *Latency & Response Time* (6.6%).

The most striking finding is the near-absence of issues explicitly framed in energy terms: the *Energy & Power Usage* category contains only 11 issues (0.3% of the sustainability-related corpus). Despite growing concern about software energy consumption in the research community [9], OSS developers overwhelmingly describe the same underlying concerns using proxy vocabulary, such as memory leaks, slowdowns, and CPU spikes, rather than energy-specific language. This proxy-term pattern is consistent with survey evidence from practitioners, who report expressing energy requirements in terms of more easily captured metrics such as running time or CPU utilization because direct energy measurement lacks tool support [22]. This vocabulary gap between research discourse and practitioner language has direct implications for automated sustainability monitoring tools that rely on energy-specific keywords.

RQ1 summary: Memory and responsiveness concerns dominate sustainability discussions in OSS (54.8% combined). Explicit energy vocabulary appears in 0.3% of sustainability issues, revealing a practitioner vocabulary gap relative to Green SE research.

4.2 RQ2 — Domain Variation

Table 5 shows the sustainability issue rate and stale rate per domain. Systems repositories produce the highest proportion of sustainability related issues (5.9%), followed by ML/Data Science (5.2%) and Backend (4.9%). Frontend (3.9%) and Mobile (3.0%) repositories show the lowest rates. The spread across domains is modest, ranging from 3.0% to 5.9%, and the ordering is consistent with the degree to which each domain manages hardware resources directly.

Figure 2 presents a heatmap of category distribution across domains, revealing distinct domain-specific sustainability profiles. Table 4 summarises the two dominant categories per domain.

Memory Consumption is the dominant category in Systems, Backend, and ML/Data repositories. This is consistent with the nature of these domains: infrastructure software and ML pipelines are long-running processes where memory accumulation over time has severe operational consequences, from out-of-memory crashes in production servers to training job failures in GPU-constrained environments. In Systems repositories, memory concerns are particularly acute because low-level software typically lacks garbage collection and must manage memory manually, and any leak propagates directly to the OS and hardware.

Table 4: Top two sustainability categories per domain. Resp. Degradation = Responsiveness Degradation.

Domain	1st category	2nd category
Systems	Memory Consumption	Resp. Degradation
Backend	Memory Consumption	Resp. Degradation
ML/Data	Memory Consumption	Performance Regression
Frontend	Resp. Degradation	Memory Consumption
Mobile	Resp. Degradation	Memory Consumption

In contrast, Responsiveness Degradation leads in Frontend and Mobile repositories. This reflects the user-facing nature of these domains: a 500 ms UI freeze is immediately visible and reported by users, whereas a 10% increase in server memory consumption may go unnoticed for weeks.

Performance Regression is the second most common category in ML/Data repositories (16.6%), the highest share of that category in any domain, consistent with development practices where model and pipeline changes are routinely benchmarked against previous versions.

RQ2 summary: Systems repositories show the highest sustainability issue rate (5.9%), while Mobile repositories show the lowest (3.0%), with ML/Data Science in an intermediate position (5.2%). The spread across domains is modest, ranging from 3.0% to 5.9%. Memory Consumption dominates infrastructure domains; Responsiveness Degradation dominates user-facing domains.

4.3 RQ3 — Report Quality

Table 6 compares the presence of four quality signals between sustainability issues and the control group of non-sustainability issues. Two signals appear more frequently in sustainability reports:

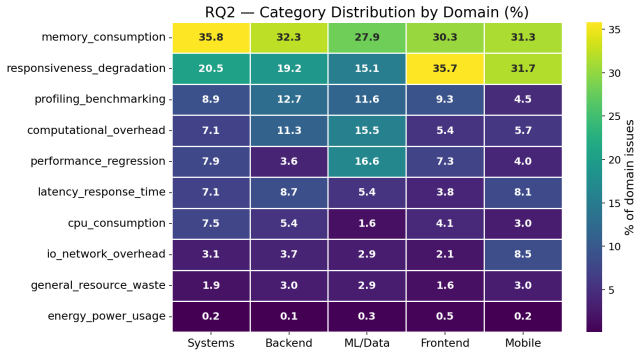

Figure 2: Category distribution per domain (%). Each cell shows the percentage of sustainability issues in that domain belonging to the given category.

Table 5: Sustainability issue rate and stale rate per domain (RQ2, RQ4).

Domain	Total	Sustain.	Rate (%)	Stale (%)
Systems	19,254	1,141	**5.9**	5.1
ML/Data	19,138	997	5.2	**3.5**
Backend	19,310	953	4.9	5.8
Frontend	19,510	765	3.9	6.3
Mobile	17,703	530	**3.0**	**7.5**

Table 6: Presence of quality signals in sustainability issues vs. control group (RQ3).

Quality signal	Sustain. (%)	Control (%)
Numeric unit (e.g. ms, %, MB)	**35.8**	20.4
Reproduction steps	**31.1**	22.4
Code block	60.5	**75.4**
Version information	46.0	**61.3**
Mean quality score (0–4)	1.73	1.79

numeric measurements with units (35.8% vs. 20.4%) and reproduction steps (31.1% vs. 22.4%). Developers reporting sustainability concerns are more likely to include concrete measurements (for example, "CPU usage reaches 100% after 30 seconds" or "response time increased from 200 ms to 2 s"), suggesting that performance degradations are more readily quantifiable than many functional issues.

Conversely, sustainability issues are less likely to include code blocks (60.5% vs. 75.4%) and version references (46.0% vs. 61.3%). This asymmetry is not a general absence of context. Reporters describe what they measured and how they triggered the behaviour, since both numeric units and reproduction steps are more frequent than in the control group, but they less often identify where it occurred, that is the specific code path, library version, or configuration involved. What is missing is the provenance of the observation rather than the observation itself. Mann-Whitney U analysis confirms a statistically significant difference in overall quality scores ($p = 0.006$), though the effect size is negligible (Cliff's $\delta = -0.039$), indicating the practical difference is small.

Table 7: Report quality by domain for sustainability issues (RQ3).

Domain	Mean score	Num. unit %	Repro. steps %	Code block %	Version ref. %
Frontend	2.13	38.7	**48.1**	**67.1**	**59.1**
Systems	1.86	34.5	36.6	63.5	51.1
Mobile	1.73	**39.6**	38.9	53.4	41.3
Backend	1.61	33.8	28.3	54.4	44.5
ML/Data	1.41	34.9	**10.3**	61.5	34.2

By category, *Latency & Response Time* issues have the highest mean quality score (2.05 of 4) and *Energy & Power Usage* the lowest (1.18), which may reflect the greater difficulty of quantifying and reproducing explicit energy concerns. By domain, Frontend scores highest (2.13) and ML/Data lowest (1.41).

Table 7 details quality scores and signal presence by domain, revealing patterns that complement the domain variation finding of RQ2.

Frontend repositories produce the best documented sustainability reports, leading on reproduction steps (48.1%), code blocks (67.1%), and version references (59.1%), likely a consequence of a browser centric culture where issues are reproducible by any user and detailed reproduction steps are established community norms. Numeric units are the one signal on which Frontend does not lead, and this signal varies little across domains (33.8%–39.6%), suggesting that the presence of a measurement is a property of sustainability issues generally rather than of any particular domain culture.

ML/Data Science repositories show a distinctive profile. Reproduction steps appear in only 10.3% of their sustainability issues, against 28.3%–48.1% elsewhere, a gap far larger than any other signal exhibits, and version references are also lowest (34.2%). Numeric units sit near the cross-domain norm (34.9%) and code blocks are second highest (61.5%). The issue is therefore not uniformly poor reporting but a loss of both procedure and provenance: ML developers supply code and measurements while rarely describing how to reproduce the condition or which version produced it, consistent with workloads where reproducing a memory or throughput problem may require a specific dataset, model checkpoint, and hardware configuration that cannot be conveyed in an issue description [27]. The resulting reports are quantitative but not independently actionable, a plausible mechanism for the low mean quality score (1.41) in this domain.

The comparison also has a tail. 18.1% of sustainability issues exhibit none of the four signals, against 12.2% of non-sustainability issues, even though the two groups have almost identical mean quality scores (1.73 vs. 1.79). Sustainability reporting is therefore more polarized than the means suggest: a larger share of reports carry a measurement and a procedure, and a larger share carry nothing at all.

RQ3 summary: Sustainability issues more often include numeric measurements and reproduction steps than non-sustainability issues, but less often code blocks and version references, so reports describe the behaviour and how to trigger it while omitting the code path and build in which it occurred. The sharpest domain contrast is reproduction steps: 48.1% in Frontend against 10.3% in ML/Data.

4.4 RQ4 — Abandonment

Sustainability issues are labeled *stale* at **5.4%**, compared to **2.4%** for non-sustainability issues ($\chi^2 = 34.4$, $p < 0.001$), representing a **2.28× higher abandonment rate**. The difference is not limited to abandoned issues: across the full classified set, sustainability issues show a longer median TTR (8.14 vs. 4.98 days, $\delta = 0.096$, negligible).

The consequences of abandonment are severe. Stale sustainability issues persist for a median of **182.1 days** before being closed, compared to just **7.0 days** for resolved issues (Mann-Whitney $p < 0.001$, Cliff's $\delta = 0.748$, large effect). This large effect size, the strongest in our study, indicates that stale sustainability issues are categorically different from resolved ones: they are not delayed fixes but effectively permanent deferrals. Most OSS projects in our sample release software every one to three months; the 182-day median stale duration means that sustainability issues survive multiple release cycles unresolved before automated staleness bots close them without any fix. The two medians, rather than a long tail between them, are what produce the large effect size: outcomes cluster at prompt resolution or at eventual automated closure, with few in between.

Figure 3 shows the stale rate comparison and the breakdown by category. Among categories, *Profiling & Benchmarking* (7.9%) and *Latency & Response Time* (6.9%) exhibit the highest abandonment rates, consistent with the diagnostic complexity of these concerns. Among domains, Mobile repositories show the highest stale rate (7.5%), likely because mobile performance and energy issues frequently require OS-level or hardware-level changes outside the control of application developers. ML/Data repositories show the lowest (3.5%), possibly because performance regressions there directly affect model quality metrics, creating stronger resolution incentive.

RQ4 summary: Sustainability issues go stale at 2.28× the rate of non-sustainability issues (5.4% vs. 2.4%, $p < 0.001$), and stale issues persist for a median of 182 days before closure (Cliff's $\delta = 0.748$, large effect), suggesting that sustainability concerns may be less consistently addressed in OSS workflows. Outcomes cluster at prompt resolution or at eventual automated closure, with few in between.

5 Discussion

Our four research questions were designed independently, yet their results converge on a coherent picture of how sustainability concerns are surfaced and handled in OSS issue tracking workflows.

Vocabulary gap and abandonment patterns. RQ1 shows that developers rarely use energy-specific vocabulary (0.3%), instead relying on proxy terms such as memory leaks, slowdowns, and CPU spikes. RQ4 shows that sustainability issues are more likely to become stale than non-sustainability issues (2.28× higher rate). While our study does not establish causality, these findings suggest a potential relationship: the absence of explicit sustainability-related terminology may make such issues harder to identify, aggregate, and prioritize within project management workflows. For example, an issue labeled *"memory leak in connection pool"* is treated as an isolated defect, whereas a set of issues explicitly labeled as *"sustainability"* or *"energy"* could be collectively triaged and addressed. This fragmentation may contribute to an implicit accumulation of sustainability-related technical debt that is less visible than functional debt.

What sustainability reports omit. RQ3 indicates that sustainability issues are less likely to include code blocks and version references than non-sustainability issues, even though they more frequently include numeric measurements and reproduction steps. Developers quantify the behaviour and often describe how to trigger it, but supply less information about the code path and version in which it appeared, which is the part a maintainer needs in order to localize the cause. Prior work in bug triage has shown that incomplete reports are harder to resolve [12], consistent with the higher abandonment rates observed in RQ4. We do not directly measure this relationship, and future work should investigate the extent to which report quality predicts resolution outcomes.

The ML/Data reproducibility gap. ML/Data is the one domain that departs from the pattern above. Elsewhere reporters supply the procedure and omit the environment, but here reproduction steps collapse to 10.3% while measurements and code remain at or above the cross-domain norm, so the reports carry neither procedure nor provenance. This is the profile least likely to be actionable by anyone other than the reporter, and it is a plausible reason why the domain combines an intermediate issue rate with the lowest report quality.

6 Threats to Validity

We discuss threats to validity following the framework of Wohlin et al. [30], organized into four categories.

6.1 Construct Validity

Performance and resource issues as sustainability proxies. Our operational definition treats runtime energy and resource-efficiency concerns (memory, CPU, latency, I/O) as observable proxies for software sustainability. This scope captures the environmental dimension that leaves traces in issue trackers, but it does not encompass the full sustainability construct, which also spans social, economic, and individual dimensions [11]. Moreover, not every performance or resource issue carries a meaningful energy footprint, so some classified issues may have limited sustainability impact. We mitigate this by grounding the category taxonomy in an established Green SE framework [9] and by instructing the classifier to assign positive labels only where the primary concern is runtime efficiency (Section 3.4), but the mapping from efficiency symptom to environmental impact remains indirect.

Keyword filter recall. The keyword filter produced a false negative rate of 5.5%, estimated from a stratified random sample of

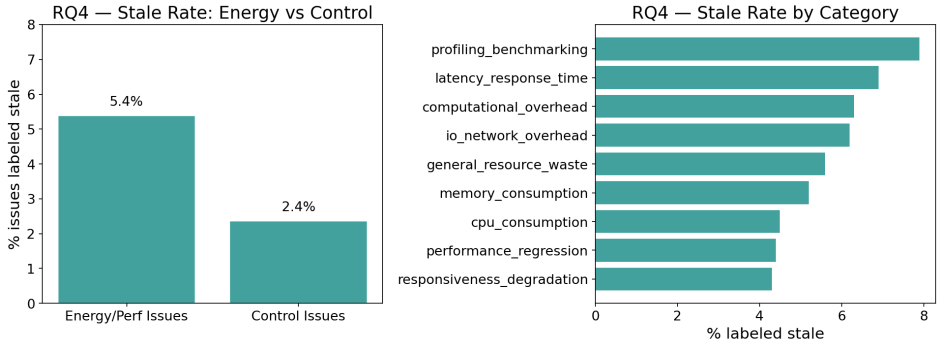

Figure 3: Stale rate for sustainability issues vs. control group (left) and stale rate by category (right).

200 rejected issues. Using the Wilson score interval at 95% confidence, this estimate yields a plausible recall range of 90.4%–96.9%. While this range is acceptable for an exploratory MSR study, it implies that some sustainability-related issues were not captured, particularly those using highly technical or domain-specific terminology not covered by our 100+ patterns. Issues describing GPU memory pressure using CUDA-specific terms, for example, may have been missed. Future work should extend the taxonomy with domain-specific vocabulary derived from the communities themselves. Our findings regarding the vocabulary gap are therefore conditioned on this operationalization and may underrepresent alternative expressions of sustainability concerns.

Quality score as proxy. The quality score (0–4) measures report richness (the presence of numeric units, reproduction steps, code blocks, and version references) rather than issue importance or resolution difficulty. A low-quality score does not mean the issue is unimportant, and a high-quality score does not guarantee resolution. This score is used exclusively as a comparative variable between groups and is not interpreted as an absolute quality measure. The quality score is a heuristic proxy and does not capture all dimensions of report quality or diagnosability.

Stale label as proxy for abandonment. The stale label is applied automatically by GitHub's stale bot after a configurable period of inactivity (typically 60–90 days). It reflects inactivity rather than deliberate intent. We acknowledge that inactivity may arise from various factors, including contributor capacity constraints or issue complexity, rather than a conscious decision to overlook the concern. Accordingly, stale status should be interpreted as an indicator of inactivity rather than definitive evidence of prioritization decisions. Nevertheless, an issue that remains unaddressed for 60+ days in an active repository can be reasonably interpreted as having been overlooked, regardless of the underlying reason. We therefore interpret stale issues as signals of reduced prioritization rather than intentional neglect. The 182-day median stale duration, more than twice the typical staleness threshold, reinforces this interpretation.

LLM classification validity. While manual validation was performed for low- and medium-confidence cases, high-confidence LLM classifications were not independently validated and may contain systematic errors not captured in the agreement analysis.

6.2 Internal Validity

LLM classification bias. The LLM systematically over-applied the Profiling & Benchmarking category (LLM: 70, researcher: 45 in the manual review sample), suggesting it is sensitive to mentions of benchmarking tools and profiling terminology even when the issue is not primarily about performance measurement. This bias was corrected through manual review of all 208 medium- and low-confidence issues, with the researcher's label used as the final classification in all 44 disagreements. The corrected dataset is what we report and analyze. However, high-confidence LLM classifications (not reviewed manually) may contain residual errors of the same type, which we acknowledge as a limitation.

Control group construction. For RQ3 and RQ4, the control group comprises all 2,463 issues that passed the filtering pipeline but were classified as non-sustainability-related, rather than a matched sample. This within-corpus complement avoids arbitrary sampling, but the two groups are not matched on issue age, reporter experience, or project maturity, and differ in size. Observed differences in report quality or abandonment may therefore be influenced by confounders such as issue complexity, reporter experience, or project-specific practices rather than by the sustainability character of the issues alone. Additionally, differences in diagnostic context (e.g., presence of code blocks and version information) may partially explain the observed differences in abandonment rates. A matched-pair design would provide stronger causal evidence but was outside the scope of this exploratory study.

6.3 External Validity

Repository selection. The 50 selected repositories are all popular, actively maintained projects with at least 500 closed issues and 100 stars. They may not represent smaller, less active, or domain-specific OSS projects. Findings about abandonment rates and vocabulary patterns may differ in younger projects, projects with smaller contributor communities, or projects explicitly committed to sustainability (e.g., those using the `green-software` GitHub topic).

GitHub pagination constraint. Three repositories were truncated by GitHub's 10,000-item pagination ceiling, biasing their samples toward recent issues:

- `rust-lang/rust`: 1,862 issues collected

- `curl/curl`: 1,634 issues collected
- `facebook/react-native`: 1,341 issues collected

These three repositories represent 6% of our sample. They are not the only source of truncation, however, as discussed next.

Per-repository sampling cap. Due to API constraints, our dataset includes up to 2,000 of the most recent closed issues per repository, a cap reached by 39 of the 50 repositories. This introduces a recency bias that may affect the observed distribution of issue lifetimes, particularly for long-lived or unresolved issues. It also means per-domain totals are determined largely by the number of repositories per domain (10 each) rather than by true issue volume, so the cross-domain rates reported in RQ2 should be read as comparisons over a fixed per-repository sample rather than over complete issue histories.

Closed-source software. All repositories in our study are open source. Sustainability engagement patterns may differ substantially in closed-source software, where development workflows, issue tracking practices, and sustainability governance policies are not publicly observable.

6.4 Conclusion Validity

Statistical tests. We apply the Mann-Whitney U test for all continuous variable comparisons and the chi-square test for categorical outcomes, consistent with prior MSR studies [6, 26]. Given the exploratory nature of this study and the conceptual independence of our research questions, we do not apply Bonferroni correction for multiple comparisons; however, this choice may increase the likelihood of Type I errors.

We report effect sizes (Cliff's δ) alongside p-values for all comparisons, enabling readers to assess practical significance independently of statistical significance. The two findings with negligible effect sizes (RQ3 quality score: $\delta = -0.039$; RQ3/RQ4 TTR: $\delta = 0.096$) are explicitly framed as statistically significant but practically small, and are not used to anchor our main conclusions.

Additionally, our analysis relies on univariate statistical tests and does not account for potential interactions between variables (e.g., domain, issue category, or report quality). As a result, the observed associations should not be interpreted as causal relationships. Future work should employ multivariate models to better isolate the contribution of individual factors to issue outcomes.

7 Conclusion

This paper presented an empirical study of how open source developers report, discuss, and resolve sustainability-related concerns. Mining 94,915 closed GitHub issues from 50 repositories across five domains, we combined keyword filtering grounded in the framework of Freed et al. [9] with LLM-assisted classification ($\kappa = 0.746$) to identify 4,386 sustainability-related issues.

Our analysis surfaced four interconnected findings: developers almost never use energy-specific vocabulary, relying instead on proxy terms such as memory leaks and CPU spikes (RQ1); sustainability engagement varies by domain, highest in Systems (5.9%) and lowest in Mobile (3.0%) (RQ2); sustainability reports carry more numeric measurements and reproduction steps but fewer code blocks and version references than non-sustainability issues (RQ3); and sustainability issues go stale at 2.28× the rate of non-sustainability issues (5.4% vs. 2.4%, $p < 0.001$), persisting for a median of 182 days before closure (Cliff's $\delta = 0.748$, large effect) (RQ4).

Taken together, these findings suggest that sustainability concerns are present in OSS communities and observable in development workflows, but may be relatively deprioritized compared to functional correctness. The practical difficulty is therefore less one of raising awareness than of making concerns that are already being reported visible enough to be triaged before automated staleness mechanisms close them.

7.1 Implications

For **practitioners**, the RQ4 pattern, where sustainability concerns are either addressed promptly or left unresolved for extended periods, indicates that explicit labeling and governance policies for resource efficiency could reduce abandonment rates. The 182-day median stale duration means these issues survive multiple release cycles before a staleness bot closes them, so the deferral becomes permanent without any explicit decision to reject. Issue templates prompting for code examples and version context, the two signals most lacking per RQ3, could also improve resolution rates.

For **tool builders**, the near-absence of energy-specific vocabulary argues for detection tools that operate on proxy signals (memory, CPU, latency, responsiveness) rather than energy keywords. Our keyword taxonomy and classification protocol, available in the replication package, provide a reusable foundation.

For **researchers**, the RQ2 domain variation suggests that ML/Data Science sustainability deserves dedicated study. Reproduction steps appear in only 10.3% of its sustainability reports against 28.3%–48.1% elsewhere, so concerns are quantified but rarely actionable, arguing for channels beyond issue trackers such as experiment logs, model cards, and benchmark repositories. The vocabulary gap also highlights the value of bridging Green SE terminology with practitioner language in empirical studies and developer-facing guidelines.

7.2 Future Work

Future work should extend this analysis to pull request discussions, commit messages, and code review comments, which may surface concerns not captured in issue trackers. Longitudinal studies tracking whether sustainability issues are resolved after closure, or resurface as new issues, would deepen understanding of the abandonment pattern in RQ4. Replicating the study across closed-source repositories and non-GitHub platforms (e.g., GitLab, Bitbucket, Apache Jira) would test generalizability.

Artifact Availability

In compliance with the SBES 2026 Open Science policy, all data collection scripts, classification prompts, datasets, manual review guidelines, and statistical analysis code are available at: https://doi.org/10.5281/zenodo.21400528

Acknowledgements

We gratefully acknowledge the support of CAPES (Finance Code 001 and PROEX), CNPq (Grants 312275/2023-4 and 420191/2025-9), FAPERJ (Grant E-26/204.256/2024), and Instituto Kunumi for their generous support.

References

[1] Sahar Ahmadisakha and Vasilios Andrikopoulos. 2024. Mining for Sustainability in Cloud Architecture Among the Discussions of Software Practitioners: Building a Dataset. In *Software Architecture. ECSA 2024 Tracks and Workshops*, Apostolos Ampatzoglou, Jennifer Pérez, Barbora Buhnova, Valentina Lenarduzzi, Colin C. Venters, Uwe Zdun, Khalil Drira, Luciana Rebelo, Daniele Di Pompeo, Michele Tucci, Elisa Yumi Nakagawa, and Elena Navarro (Eds.). Springer Nature Switzerland, Cham, 150–166.

[2] Michel Albonico, Ivano Malavolta, Gustavo Pinto, Emitza Guzman, Katerina Chinnappan, and Patricia Lago. 2021. Mining Energy-Related Practices in Robotics Software. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, Piscataway, NJ, USA, 483–494. doi:10.1109/MSR52588.2021.00060

[3] Gabriel Aracena, Kyle Luster, Fabio Santos, Igor Steinmacher, and Marco Aurelio Gerosa. 2024. Applying Large Language Models to Issue Classification. In *Proceedings of the Third ACM/IEEE International Workshop on NL-Based Software Engineering* (Lisbon, Portugal) *(NLBSE '24)*. Association for Computing Machinery, New York, NY, USA, 57–60. doi:10.1145/3643787.3648043

[4] Manuela Bechara Cannizza and Michel Albonico. 2026. A Curated List of Open-source Software-only Energy Efficiency Measurement Tools: A GitHub Mining Study. In *Proceedings of the 2026 IEEE International Conference on Software Architecture Companion (ICSA-C) (GreenArch '26: 1st International Workshop on Software Architecture for Green Sustainable Carbon-aware Software Systems)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–6. Accepted as paper at GreenArch Workshop, part of ICSA 2026..

[5] Coral Calero and Mario Piattini (Eds.). 2015. *Green in Software Engineering* (1 ed.). Springer Cham, Cham, Switzerland. doi:10.1007/978-3-319-08581-4 XII, 327 pp..

[6] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114 (1993), 494–509. Issue 3. doi:10.1037/0033-2909.114.3.494

[7] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46. doi:10.1177/001316446002000104

[8] Déaglán Connolly Bree and Mel Ó Cinnéide. 2025. How Software Design Affects Energy Performance: A Systematic Literature Review. *Journal of Software: Evolution and Process* 37, 4 (2025), e70014. doi:10.1002/smr.70014 e70014 smr.70014.

[9] Martina Freed, Sylwia Bielinska, Carla Buckley, Andreea Coptu, Murat Yilmaz, Richard Messnarz, and Paul M. Clarke. 2023. An Investigation of Green Software Engineering. In *Systems, Software and Services Process Improvement*, Murat Yilmaz, Paul Clarke, Andreas Riel, and Richard Messnarz (Eds.). Springer Nature Switzerland, Cham, 124–137.

[10] Charlotte Freitag, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon S Blair, and Adrian Friday. 2021. The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. *Patterns* 2 (9 2021), 100340. Issue 9. doi:10.1016/j.patter.2021.100340 doi: 10.1016/j.patter.2021.100340.

[11] Gabriel Alberto García-Mireles, Ma Ángeles Moraga, Félix García, and Coral Calero. 2026. Sustainability in the Field of Software Engineering: A Tertiary Study. *ACM Trans. Softw. Eng. Methodol.* 35, 5, Article 141 (April 2026), 86 pages. doi:10.1145/3747178

[12] Emanuel Giger, Martin Pinzger, and Harald Gall. 2010. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering* (Cape Town, South Africa) *(RSSE '10)*. Association for Computing Machinery, New York, NY, USA, 52–56. doi:10.1145/1808920.1808933

[13] Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. 2023. ChatGPT outperforms crowd workers for text-annotation tasks. *Proceedings of the National Academy of Sciences* 120, 30 (2023), e2305016120. doi:10.1073/pnas.2305016120

[14] Ahmed E. Hassan. 2008. The road ahead for Mining Software Repositories. In *2008 Frontiers of Software Maintenance*. IEEE, Piscataway, NJ, USA, 48–57. doi:10.1109/FOSM.2008.4659248

[15] Abram Hindle, Alex Wilson, Kent Rasmussen, E. Jed Barlow, Joshua Charles Campbell, and Stephen Romansky. 2014. GreenMiner: a hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 12–21. doi:10.1145/2597073.2597097

[16] Mia Mohammad Imran and Tarannum Shaila Zaman. 2026. OLAF: Towards Robust LLM-Based Annotation Framework in Empirical Software Engineering. In *Proceedings of the 3rd International Workshop on Methodological Issues with Empirical Studies in Software Engineering (WSESE 2026)*. ACM, New York, NY, USA, 1–6.

[17] Bihui Jin, Heng Li, Pengyu Nie, and Ying Zou. 2026. Energy-Efficient Software Development: A Multi-dimensional Empirical Analysis of Stack Overflow. In *Proceedings of the 48th International Conference on Software Engineering*. ACM, New York, NY, USA, 1–13. doi:10.1145/3744916.3773177

[18] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 92–101. doi:10.1145/2597073.2597074

[19] Christoph König, Daniel J. Lang, and Ina Schaefer. 2025. Sustainable Software Engineering: Concepts, Challenges, and Vision. *ACM Trans. Softw. Eng. Methodol.* 34, 5, Article 135 (May 2025), 28 pages. doi:10.1145/3709352

[20] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174. http://www.jstor.org/stable/2529310

[21] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2014. Mining energy-greedy API usage patterns in Android apps: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 2–11. doi:10.1145/2597073.2597085

[22] Irene Manotas, Christian Bird, Rui Zhang, David Shepherd, Ciera Jaspan, Caitlin Sadowski, Lori Pollock, and James Clause. 2016. An Empirical Study of Practitioners' Perspectives on Green Software Engineering. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. 237–248.

[23] Alejandro Mazuera-Rozo, Catia Trubiani, Mario Linares-Vásquez, and Gabriele Bavota. 2020. Investigating types and survivability of performance bugs in mobile apps. *Empirical Software Engineering* 25 (2020), 1644–1686. Issue 3. doi:10.1007/s10664-019-09795-6

[24] Fabio Palomba, Dario Di Nucci, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. 2019. On the impact of code smells on the energy consumption of mobile applications. *Information and Software Technology* 105 (2019), 43–55. doi:10.1016/j.infsof.2018.08.004

[25] Gustavo Pinto, Fernando Castor, and Yu David Liu. 2014. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 22–31. doi:10.1145/2597073.2597110

[26] Jeanine Romano, Jeffrey D. Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys?. In *Annual Meeting of the Florida Association of Institutional Research*. Florida Association of Institutional Research (FAIR), Cocoa Beach, FL, 1–3. https://www.scribd.com/document/45715000/Appropriate-Statistics-for-Ordinal-Level-Data

[27] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2020. Green AI. *Commun. ACM* 63, 12 (Nov. 2020), 54–63. doi:10.1145/3381831

[28] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M. Ibrahim, Masao Ohira, Bram Adams, Ahmed E. Hassan, and Ken-ichi Matsumoto. 2010. Predicting Re-opened Bugs: A Case Study on the Eclipse Project. In *2010 17th Working Conference on Reverse Engineering*. IEEE, Los Alamitos, CA, USA, 249–258. doi:10.1109/WCRE.2010.36

[29] Roberto Verdecchia, Patricia Lago, Christof Ebert, and Carol de Vries. 2021. Green IT and Green Software. *IEEE Software* 38, 6 (2021), 7–15. doi:10.1109/MS.2021.3102254

[30] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2024. *Experimentation in Software Engineering* (2 ed.). Springer Berlin, Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-662-69306-3 XXV, 274 pp..