GREEN CODE

AI/ML Driven Software Optimisation to Reduce Cost and Climate Impact

State of the Art Review

Deliverable 2.1, 2025

GreenCode: State of the Art Review

Acknowledgements

GreenCode is a large project, and this document is the result of inputs and feedback from over 70 contributors representing or connected to the various partner organisations involved. Specific thanks go out to the following individuals for their direct support in drafting key areas of this review, though all inputs and contributions received have been deeply valued throughout.

Bent Thomsen, Aalborg Universitet, Denmark - Esther Kim, Aalborg Universitet, Denmark - Chris Dean, Digital Tactics, UK – Philipp Zähl, Fachhochschule Aachen, Germany - Andreas Jedlitschka, Fraunhofer IESE, Germany - Julien Siebert, Fraunhofer IESE, Germany - Patricia Kelbert, Fraunhofer IESE, Germany - Andreas Biesdorf, Hochschule Trier, Germany - Tobias Böhm, Hochschule Trier, Germany - Rui Henriques, IrRADIRE, Portugal - Pedro Faria, ISEP, Portugal – Alireza Mehri, KAN Engineering, UK - Mohammad Mousavi, Kings College, UK - Ezgi Sarikayak Siemens, Germany – Fridtjof Siebert, Tokiwa Software, Germany - Michael Herrnberger, TWT, Germany - Tina Vartziotis, TWT, Germany - Daniel Esteban Villamil, UC3M, Spain - Benjamin Weigell, Uni Augsburg, Germany - Bernhard Bauer, Uni Augsburg, Germany - Yücel Şentürk, VBT, Türkiye - Johannes Passand, ZAL Aero, Germany

With Thanks,

Chris Dean, Work Package 2 Lead, December 2025

Revision History

Version	Authors	Content	Date
0.1	Oge Bozyigit	Initial plan for SotA review	05/02/2025
0.2	Chris Dean	Transition of WP Lead. Document plan re-drafted with comprehensive contributor guidance and made ready for consortium contributions	15/04/2025
0.3	All Authors	Contributions from all consortium partners gathered and intermediary structural edits made	30/09/2025
0.4	Chris Dean	Whole document Initial pass of structural edits and referencing checks	15/10/2025
0.5	Chris Dean	Final edits, checks and submission	23/12/2025

GreenCode: State of the Art Review

Contents

GreenCode: State of the Art Review

Introduction

GreenCode is an innovation project focused on AI/ML-driven optimisation of software. Its aim is to reduce the cost and climate impact of software systems (software and the infrastructure it runs upon), and by implication the carbon impact of the IT sector at scale, impacting everything from edge devices to datacentres.

Green and sustainable software is a large and growing subject area with various actors also now taking steps to improve elements of software sustainability and the efficiency of the software development lifecycle (SDLC) through education and AI interventions, some notable names being: the Green Software Foundation, GSF, that focusses on international standards, best practice policy and community building; academic organisations such as the Software Sustainability Institute, SSI, at The University of Edinburgh and the Digital Sustainability Centre at Vrije University, Amsterdam who focus on education and outreach; numerous startups and other RD&I initiatives like the GENIUS project focussing on AI tools for the SDLC.

Rather than duplicating the effort already expended by such organisations, we look to extend their work alongside our own through a detailed understanding of energy use, wastage and its relation to software quality, sustainability and performance enhancement. Through this we look to provide a demonstrable, measured route to the reduction of energy waste and inefficiency in software systems and to consider how this might be certified against new and recently published international standards such as that for Software Carbon Intensity (SCI). Specifically, we are creating an agentic AI application pipeline and associated tooling to automate the optimisation of software systems for quality and energy efficiency, that can readily track with and be continually upgraded vs. emergent state-of-the-art technologies.

As important as the tooling itself is, where we apply it is also key. Research estimates that 60-80% of all software is regarded as legacy software and that many market players are focussed on the generation of new software applications from scratch through AI, we first intend to address the low-hanging fruit of legacy system maintenance, rationalisation, optimisation and upgrade/porting. While observing the quality of AI generated code in a similar manner and considering how the tooling may serve to improve this also.

Through a variety of routes our work will provide value to the owners of existing systems through measurable reductions in TCO and technical debt, while also providing verified green credentials for their systems and numerous other benefits that improve market competitiveness. From a climate point of view software optimisation as an intervention in the climate crisis enables a fast, step-change reduction in emissions/energy consumption when implemented within applications already deployed at scale.

In this document we discuss the state-of-the-art in the key thematic areas the GreenCode project and its partners look to address, based on a combination of academic and commercial research across scientific literature and, given the pace of advancement outside of academia, various grey literature sources also. Furthermore, given the partners diverse sectorial experience we defer to their expertise when discussing the SotA of their specific area of application of this work.

GreenCode: State of the Art Review

Document Structure

This document consists of both an academic leaning thematic review of the state-of-the-art as applies to GreenCode and a similar industry-leaning sectorial review. In both cases we describe the background to a particular sub-topic, the current state-of-the-art, identified limitations and gaps, areas that have been considered but are not immediate priorities within the project, and the opportunities for the project's application to that area going forward. We close with a summary conclusion and consolidated summary of gaps, opportunities, high level requirements and new business models that may emerge as a result of the work.

Terminology

Throughout this report, "sustainability" primarily refers to energy and carbon impacts; "measurement boundary" denotes the scope of included components and allocation rules; and "closed-loop optimisation" denotes iterative measure–change–re-measure cycles with regression validation. We also use the following abbreviations and definitions regularly:

Term	Definition
SotA	State of the Art
Software System	A software application and the infrastructure required to run it
GenAI	Generative AI
Hotspot	An area of interest in a software system. Typically, a relatively high energy consuming piece of code or infrastructure; a potential cause of energy waste; or a software or hardware performance or throughput pinch-point.
Sustainability	Unless otherwise clear from the text sustainability in this document means environmental sustainability. Typically, with respect to operational energy use.
CO2e	CO2 equivalent: The equivalent emissions of CO2
GHG	Greenhouse Gas
LLM	Large Language Model (7B parameters or bigger)
SLM	Small Language Model (under 7B parameters)
xB parameters	x (where x is a number) Billion of parameters that exist within a language model

GreenCode: State of the Art Review

Executive Summary

GreenCode focuses on AI/ML-driven software optimisation to reduce the cost and climate impact of software (and the infrastructure it runs upon). Our aim is to translate the rapidly expanding body of sustainable software research and tooling into a demonstrable, measured pathway for reducing operational energy use and enabling standards-aligned certification, while avoiding duplication of efforts led by other organisations.

A core practical premise underpins GreenCode: most emissions and costs are locked into existing systems, and a large proportion of software estates are “legacy”. Although current market attention is strongly drawn to AI-generated “new build” applications, substantial near-term impact is often achievable through legacy maintenance, rationalisation, optimisation, and modernisation. Embedding measurement and optimisation into routine engineering workflows enables rapid, evidence-based improvement, with step-change reductions realised when updated software is deployed at scale. The same principles may later be extended to AI-generated code as practices and evidence mature, and this is being explored in parallel.

This review finds that software sustainability has moved from a niche concern to an emerging mainstream requirement. However, the landscape remains fragmented: measurement tools, reporting methods, and optimisation techniques exist, but are inconsistently integrated and unevenly operationalised across different software types (cloud, embedded, mainframe, media/gaming, etc.). A consistent cross-domain theme is the gap between “knowing” and “doing”: organisations may measure or estimate energy/carbon footprints, but often lack repeatable, production-ready routes to reduce them without trading off performance, reliability, or delivery speed.

Key findings across the thematic areas

- Energy measurement, metrics, and attribution are improving, but results are often hard to compare due to inconsistent boundaries, access constraints (especially in managed/cloud environments), and weak attribution from system-level energy signals to actionable software artefacts.
- AI tooling for software engineering is advancing rapidly (generation, translation, testing, refactoring), but current tools are typically optimised for correctness and productivity, not for energy/performance efficiency, and rarely close the loop from runtime evidence back into code change recommendations.
- Green AI methods and tools are growing, yet comparability and industrial validation remain weak; sustainability tooling often operates in silos and is under-integrated into CI/CD and decision workflows. A standout opportunity is a closed-loop approach: measure > optimise > validate > regress-test, enabling “green regression” detection as a normal engineering capability.
- Developer practices and organisational adoption are limiting factors: even where tactics exist, uptake is constrained by perceived trade-offs, cost, and lack of workflow-integrated tools; education and voluntary guidance alone cannot scale to the size of the industry.

- Standards and certification (SCI and related schemes) provide an increasingly important foundation for reporting, system-specific iteration and procurement incentives, but they currently emphasise calculation and disclosure more than automated, evidence-backed optimisation, leaving a gap between compliance and real impact.

Consolidated gaps in the State-of-the-Art and research needs

A credible sustainability optimisation workflow forms a closed loop: (1) system and workload characterisation, (2) measurement and instrumentation with explicit boundaries, (3) hotspot attribution across code and infrastructure layers, (4) optimisation/refactoring with guardrails for correctness, performance, and security, and (5) regression validation with reporting suitable for governance and audit. The major SotA gap is that most current approaches cover only fragments of this loop and do not reliably connect runtime evidence to repeatable optimisation decisions.

The gaps therefore cluster into three themes. First, evidence quality: measurements are often non-repeatable, boundaries and allocation rules vary across shared infrastructure, and traceability is insufficient for governance and audit. Second, technical linkage: attribution to code/architecture is limited and closed-loop optimisation is weakly integrated into routine engineering workflows, with lifecycle and regression handling underdeveloped. Third, practical adoption: methods frequently fail to generalise across domains and real deployment constraints, and organisational barriers (tool friction, skills, incentives) prevent sustained use; additionally, GenAI-enabled development commonly lacks energy-aware defaults and clear evidence of net benefit.

This review surveys the evidence behind these gaps, compares existing approaches, and highlights where current tooling and practice fall short of end-to-end, repeatable improvement.

Why GreenCode's contribution is valuable

GreenCode is positioned to unify these developments into a modular platform and evidence pipeline that links: (i) software quality/performance engineering, (ii) measurement and attribution, (iii) AI-assisted optimisation, and (iv) standards-aligned reporting and assurance, delivering continuous improvement rather than one-off reporting. The project's high-level requirements emphasise evidence-first automation, multi-source measurement adapters, traceability to artefacts, closed-loop optimisation, standards-aligned export, confidence grading for shared infrastructure, and lifecycle-aware regression detection, implemented with a modular architecture that supports sector-specific adapters (e.g., domain-specific adapters across legacy, cloud, embedded, and high-performance contexts) while maintaining a common evidence model.

GreenCode: State of the Art Review

Thematic Review

The thematic section of our review reflects the state of the art of GreenCode's technical themes and concerns. This is a wide and rapidly evolving area with hundreds of references reviewed across the contributing partners. For this reason, relevant references have been surfaced as a priority with many opportunities identified to refer to extended source material for detail and connections to extended research.

It has been our intention to describe the most recent research and to develop GreenCode relevant views from that, while not repeating the source material in detail. In this sense the document extends beyond a literature review alone and into richer valuable commentary on the SotA in each case, providing a specific understanding of opportunities for GreenCode to fill gaps.

Topics within this document are also finely divided (there are many discrete sub sections) so that it may also act as a centralised quick-reference for all partners seeking SotA information on a specific topic.

GreenCode: State of the Art Review

Energy Measurement, Metrics, and Attribution Techniques

Contributing Partner(s): TWT, Digital Tactics, Aalborg University, University of Augsburg, Trier University of Applied Sciences

Background

In recent years the world has seen rising global attention on the energy and environmental (water, materials, carbon) impact of digital technologies. Policy frameworks are increasing the pressure on organizations to quantify and reduce the environmental footprint of IT. Many enterprises are also embedding sustainability targets within their ESG reporting frameworks, with software efficiency emerging as a measurable lever for reducing Scope 2 emissions as well as reducing operational costs especially in software only businesses (SaaS providers, TV and video games, digital media agencies, etc.).

Optimizing software for energy efficiency requires balancing multiple, and sometimes conflicting, objectives such as performance, energy and accuracy. In AI/ML systems for example, higher accuracy often requires larger models and longer training runs which increase energy use[1]. Similarly, time/latency-sensitive workloads may consume disproportionate amounts of energy to ensure real-time deadlines are met.

Many computing platforms exhibit substantial idle power draw (often reported as a significant fraction of peak power, depending on hardware generation, power management features, and configuration), making consolidation and right-sizing important levers alongside code-level efficiency[2].

To meet these requirements and others we need well defined methods for measuring or closely estimating energy consumption and standardised metrics that enable comparison, however determining these in detail on a micro-level is inherently challenging because modern computing systems are highly complex and non-deterministic.

Power consumption is influenced not only by the instructions executed, but also by microarchitectural behaviour (e.g., cache hits, branch prediction, memory stalls), concurrent background processes, and dynamic hardware features such as frequency scaling, turbo boost, and power-saving states. Even when identical code is executed repeatedly, variations in workload scheduling, thermal conditions, and system interrupts can lead to measurable differences in energy use.

Furthermore, isolating the energy attributable to a single piece of software is difficult in multi-tenant environments such as cloud platforms, where resource sharing obscures per-application measurement.

These issues mean that energy profiling less straightforward than traditional performance benchmarking, and studies often require controlled conditions, repeated runs, and careful methodological design to obtain results to high level of accuracy although these rapidly depart from real execution environments.

While absolute precision is important in determining energy consumption metrics on a micro-scale e.g. in microcontroller and IoT contexts or when considering the merits of a particular discrete

algorithm, the degree of component precision becomes rapidly less important in macroscale environments where algorithms operate and interact in bulk as part of a complete application. In these contexts, while precision remans important, the net change in energy consumption of the application as whole can be impacted by wider architectural and development choices which can be determined by a level of macroscale measurement and analysis under stress conditions, which opens up opportunities for GreenCode to have bulk impact on IT energy consumption despite the inherent challenges of micro-scale measurements.

Current State of the Art

Definition of Measurement Metrics and Attribution Patterns

Sustainability hotspots in IT systems can be identified using specific metrics and patterns. Metrics support the quantification and comparison of sustainability aspects across individual components and the infrastructure as a whole, while patterns help uncover structural inefficiencies.

The following sections examine current sustainability metrics and patterns used in IT.

Sustainability Metrics

Sustainability metrics can be classified into three domains:

1. Power and energy consumption metrics,
2. Energy efficiency metrics, and
3. Environmental impact metrics.

Each domain addresses different aspects of sustainability, but they are conceptually interlinked.

Power and energy consumption metrics, commonly applied at both the hardware and software levels, form the foundation of sustainability assessments. These metrics capture the raw quantity of energy consumed by components at a specific point in time or over a given interval (e.g. watts, joules, kilowatt-hours), and they provide the necessary baseline for further analysis and comparison.

Energy efficiency metrics build on consumption data by relating energy usage to system performance, output, or provisioning, such as transactions, operations, computational tasks, or infrastructure utilization. This allows for the comparison of different elements for sustainability.

Some metrics assess energy use per unit of work, while others capture structural or provisioning inefficiencies that influence overall energy effectiveness. Finally, environmental impact metrics contextualize energy consumption within broader sustainability frameworks, translating usage into indicators such as CO2e or overall GHG emissions.

Power and Energy Consumption Metrics and Measurement

The fundamental distinction between power and energy metrics lies in the nature of the metrics themselves. Power is measured in watts (W) and indicates the instantaneous rate of work performed at a specific point in time. Energy however is quantified in joules (J) or kilowatt-hours (kWh) and describes the total work performed over a period of time[3].

At the hardware level, measurements focus on the physical components that consume energy. Measurement approaches typically begin at a coarse-grained level, capturing power and energy consumption at the scale of entire machines, including servers, racks, routers, and switches[4,5,6,7,8,9].

Typically used tools include Power Distribution Units (PDU), which measure power per outlet and support remote monitoring, standalone power and watt meters such as Watts Up Pro, which are typically plugged in between the device and the power source[10,11,12] and the Intelligent Platform Management Interface (IPMI) 2.0 or the newer Datacentre Manageability Interface (DCMI) Specification, which provide server-level power data. However, these interfaces suffer from low temporal resolution, reporting power consumption data only at intervals of several seconds[13]. Additionally, some vendors offer their proprietary APIs for measurement, such as HP iLO[14] or DELL iDRAC[15].

From this system-wide view, the focus shifts to finer-grained components, including CPU, memory, motherboard, storage, GPUs and cooling components.

The CPU is frequently examined[16], often in conjunction with associated elements such as the motherboard and memory.

To obtain energy readings for the CPU and memory, Intel's Running Average Power Limit (RAPL) interface is often used [8,11,17,18]. It was introduced around 2011 and has since emerged as a de-facto standard as it is now also supported by AMD with AMD Family 17h Processors: Model 30h [19].

Additionally, storage devices such as HDDs and SSDs[Error! Bookmark not defined.Error! Bookmark not defined.Error! Bookmark not defined.Error! Bookmark not defined.Error! Bookmark not defined.] must be considered. For modern storage options like NVMe drives, the NVMe Management Interface Specification can be used to query power consumption readings.

With the growing adoption of machine learning and data-intensive workloads, hardware accelerators, especially NVIDIA GPUs, have been increasingly utilised. To measure the power consumption of a GPU, NVIDIA integrated shunt resistors and current monitor ICs into its GPUs starting with the Kepler architecture[20]. Power consumption data can be accessed programmatically via the NVIDIA Management Library (NVML)[21] or, more conveniently, through the Nvidia Smi command-line utility.

A network's power and energy consumption can be considered from two viewpoints. A host-only view considers solely the power/energy consumption by the network interface card. This can be approximated by multiplying utilization measurements with the idle and active power values specified in the hardware vendor's documentation. This approach has the drawback of being limited solely to the host and not considering intermediate transmission steps.

In contrast, the whole-path view extends the boundary to the external network. For this estimation, two methods can be applied. First, an average electricity-intensity factor (kWh/GB) converts transmitted bytes into energy consumption. Aslan et al.[22] report 0.06 kWh/GB for UK fixed networks in 2015 and observe that the figure halves roughly every two years, which results in 0.001875 kWh/GB for 2025 if the historical trend continues. Second, Mytton et al.[23] proposed a two-component model that separates transmission (core-network) energy from customer-premises equipment energy. The model combines fixed baseline constants with terms that scale with link speed and utilization. Finally,

cooling systems such as fans and air conditioning are examined due to their considerable contribution to total energy usage in IT infrastructures.

A summary of the presented measurement approaches is given in the table below.

Component	Metric type	Interface / API	Scope covered
Server / rack / switch	Power, Energy	PDU, standalone watt-meter, IPMI / DCMI, vendor APIs (iLO, iDRAC)	Whole machine
CPU	Power, Energy	RAPL	On-chip CPU domain
Memory	Power, Energy	RAPL	System DRAM
GPU / accelerator	Power, Energy	NVML / nvidia-smi	GPU card
Storage (HDD / SSD / NVMe)	Power, Energy	NVMe Management Interface	Individual drive
NIC (host)	Power, Energy	Vendor idle / active datasheet	Network-interface card only
Network, whole path (average)	Energy intensity(kWh / GB)	Average factor model Error! Bookmark not defined.	Core + access network (aggregate)
Network, whole path (segmented)	Power, Energy	Two-component power model Error! Bookmark not defined.	Core + access network (aggregate)
Cooling	Power, Energy	PDU channels, building-management sensors	machine or facility cooling

Energy Efficiency Metrics

This section focuses on energy efficiency metrics applicable to IT infrastructure components and the IT infrastructure itself. Facility-level or datacentre metrics such as Power Usage Effectiveness (PUE), Compute Power Efficiency (CPE), and the Datacentre Energy Efficiency and Productivity Index (DC-EEP Index) are acknowledged but likely require adaptation to be calculable for IT infrastructures.

A list of currently available energy efficiency metrics is as follows:

Component	Metric Name	Formula
CPU	FLOPS / Watt Error! Bookmark not defined.,Error! Bookmark not defined.	Energy consumed / Number of executed instructions
CPU	MHz per Watt Error! Bookmark not defined.,Error! Bookmark not defined.	MHz / Watt
Storage	Capacity per Watt Error! Bookmark not defined.,Error! Bookmark not defined.	Capacity / Watt
Storage	IOPS per Watt Error! Bookmark not defined.,Error! Bookmark not defined.	IOPS / Watt
Server / Rack	Space, Watts and Performance (SWaP) Error!	SWaP = Performance/ (Space × Power Consumption)

	Bookmark not defined.,Error! Bookmark not defined.	
System of Systems	Deployed HW Utilisation Ratio (DH-UR) Error! Bookmark not defined.,Error! Bookmark not defined.	Active servers / Total servers deployed
System of Systems	Deployed HW Utilisation Efficiency (DH-UE) Error! Bookmark not defined.,Error! Bookmark not defined.	minimum quantity of servers needed to handle peak load / total number of deployed servers
System of Systems	Power Usage Effectiveness (PUE) Error! Bookmark not defined.,Error! Bookmark not defined.	Total Facility Power / IT Equipment Power
System of Systems	Datacentre Infrastructure Efficiency (DCIE) Error! Bookmark not defined.,Error! Bookmark not defined.	IT Equipment Power / Total Facility Power
System of Systems	Compute Power Efficiency (CPE) Error! Bookmark not defined.,Error! Bookmark not defined.	DCIE / IT Equipment Utilisation
System of Systems	Datacentre Energy Efficiency and Productivity Index (DC-EEP Index) Error! Bookmark not defined.,Error! Bookmark not defined.	IT-PEW * SI-EER
System of Systems	Site Infrastructure Energy Efficiency Ratio (SI-EER) Error! Bookmark not defined.,Error! Bookmark not defined.	Conditioned Power Out / Power In
System of Systems	Site Infrastructure Power Overhead Multiplier (SI-POM) Error! Bookmark not defined.,Error! Bookmark not defined.	datacentre power consumption at the utility meter / total hardware AC power
System of Systems	IT hardware power overhead multiplier (H-POM) Error! Bookmark not defined.,Error! Bookmark not defined.	total hardware load at the plug / total Hardware Compute Load
System of Systems	Datacentre energy productivity (DCeP) Error! Bookmark not defined.,Error! Bookmark not defined.	work output (bytes) / total energy
System of Systems	Datacentre performance efficiency (DCPE) Error! Bookmark not defined.,Error! Bookmark not defined.	effective IT workload / total facility power
System of Systems	Coefficient of performance of the ensemble (COP Ensemble) Error! Bookmark not defined.,Error! Bookmark not defined.	COP Ensemble = Total Heat Dissipation/(Flow Work+ Thermodynamic Work) of cooling system
System of Systems	Greenup [24]	Energy consumed baseline / Energy consumed optimized
System of Systems	Green energy coefficient [25]	Green energy consumed / total energy consumed

Network	Bandwidth per Watt Error! Bookmark not defined.,Error! Bookmark not defined.	Bandwidth / Watt

Environmental Impact Metrics

Energy and power consumption metrics and energy efficiency metrics focus exclusively on quantifying energy or power consumption. To place these in a broader sustainability context, environmental impact metrics translate energy use into environmental consequences and capture more general sustainability in an IT infrastructure.

This translation is typically based on carbon-related indicators, such as CO2e, greenhouse gas emissions, or carbon footprint[26]. In addition, broader sustainability concerns in IT infrastructures include electronic waste, lifecycle costs, and supply chain impacts related to hardware and software.

Energy Impact of Data Storage

The storage, transfer and manipulation of data is an important sub-topic for energy-efficient optimisation given the pace of data growth globally.

Banijamali et al. (2024)[27] conducted an industrial case study in collaboration with Schuberg Philis (SBP), a leading IT provider, to define Key Performance Indicators (KPIs) for monitoring and improving the energy efficiency of cloud data storage. Building on the KPI framework proposed by Fatima et al. (2024)[28], they developed three KPIs tailored to SBP's sustainability roadmap: energy consumption, storage capacity utilisation, and capacity per watt. These KPIs resulted in specific recommendations, including the replacement of inefficient hardware, architectural redesigns, and compression strategies.

Measurement-Relevant Sustainability Patterns

The Green Software Foundation offers a catalogue of various patterns that can contribute to structured solutions for reducing the carbon footprint in software development and operation. These patterns are categorised into AI, Cloud and Web, defined in a platform-neutral manner and reviewed by various experts.

In the Web category, for example, the pattern Defer Offscreen Images describes how images that are not visible on the first display page are only reloaded when needed. This saves bandwidth and computing power on both the server and client sides.

In the AI category, there is the pattern Optimize the Size of AI/ML Models, which reduces the complexity of AI and ML models. This can significantly reduce computing power and memory requirements in particular.

For cloud applications, the pattern Encrypt What is Necessary addresses the high resource requirements of modern encryption algorithms and recommends only encrypting data that really needs to be protected.

All these patterns combine technical practices with measurable effects on software carbon intensity (SCI), making them not only ecologically effective but also practical to implement. Their application

demonstrates how systematic ecological design can be anchored at various levels of software architecture.

Physical Energy Measurement Techniques

There two ways of measuring energy consumption from running software, either by an external device, often referred to as a wall-plug measurement, or through on chip sensors in this section we will discuss both opportunities.

External "Wall-plug" Measurements

Power Meters

Wall-plug measurements can be performed on a headline level using a simple smart plug such as the CloudFree EU smart Plug, a minimal intrusion device such as the MN60 AC Current Clamp, or a full-fledged power analyzer as required by detailed testing frameworks such The Standard Performance Evaluation Corporation (SPEC) for the SPECpower_ssj® 2008 benchmarks[29] and the EET/FEETING framework[30] [31].

PSU Measurements

A potentially more refined view of wall-plug measurements might be achieved through power monitoring at the PSU level as the energy is received and distributed within the machine (workstation, server, etc.). PSU manufacturers such as Corsair[32] have certain options for power monitoring on this level though these facilities are typically geared towards delivering gaming experiences and are proprietary.

Another option is to introduce a manufacturer agnostic power monitor between the PSU and the components of the machine; both are under active exploitation[33].

Challenges of External Measurements

A challenge with external power measurements is to synchronize the power data with the software under test. This can be done by time stamping or by the software under test signalling start and stop of measurements, however, the latter usually requires some form of network access which is often turned off to reduce jitter in energy measurements. Furthermore, external measurements are typically sampled once every second whereas software operates on a much shorter timescale (milliseconds or less) meaning that tests need to crafted such that measurement timescales correspond between the running software and the resolution of the monitoring device.

Internal Sensor Measurements (RAPL)

A prominent example of on chip sensors is the Running Average Power Limit (RAPL) interface, supported on modern Intel processors and some AMD processors. These have a much finer time resolution that wall-plug measurements (samples taken per millisecond vs. second).

Studies show that wall-plug measurements and RAPL measurements are well aligned[34] and therefore longer running processes might be adequately measured by wall-plug methods.

How RAPL Works

RAPL leverages model-specific registers (MSRs) and hardware performance counters to calculate the energy consumption of the CPU. The MSRs used by RAPL monitor the power domains PKG (All components of the CPU package), DRAM (main memory referenced by the CPU), PP0 (the CPU Cores, and caches as a subset of PKG), and PP1 (typically the GPU in chips with integrated graphics handling as a subset of PKG).

Working with RAPL

RAPL can be used directly but managing the conversion from MSR to Joules and the fact that the MSRs used by RAPL wraps around roughly every 60 seconds can be cumbersome, thus several tools and libraries exist for runtime energy measurement built on top of RAPL, as described later in Summary of software energy measurement frameworks and hardware/system-level profilers.

Several programming languages such as Haskell and Python, also have APIs or frameworks for accessing energy information from RAPL such as pyRAPL

Security Distorting Power Measurement

Unfortunately, RAPL has come under some criticism from a security standpoint as its fine-grained power monitoring could lead to the inference and exfiltration of sensitive information through power data (PLATYPUS Attack).

For this reason, access to full RAPL data is only available to highly privileged operating system accounts which can provide a barrier to monitoring in various environments. Not only this but RAPL also distorts data under some circumstances to mask secure operations[35] reducing its reliability in specific security scenarios.

Predicted/Simulated Measurement and Modelling Approaches

While measured energy profiling and post-use analysis is the foundation for optimizing software energy performance, there is growing interest in predictive methods. AI and ML techniques are being developed to infer likely energy consumption directly from code patterns, compiler outputs, or hardware counters[36] .

A significant body of work has explored simulation and modelling tools to estimate software energy consumption. These approaches are particularly valuable in early design phases, where hardware may not yet be available, or when fine-grained attribution is required.

One of the most widely used frameworks in computer architecture research is McPAT (Multicore Power, Area, and Timing), which models the energy and power consumption of processors at the architectural level [37]. Combined with simulators such as gem5[38], McPAT enables researchers to estimate the energy implications of different software workloads running on simulated hardware configurations. Similarly, tools like Wattch and SimplePower provide cycle-level simulation of energy costs for specific processor components [39].

For software developers, higher-level estimation frameworks have also emerged. For example, the Green-Marl graph processing language incorporated energy-aware optimizations based on analytical models[40]. Other research has investigated building energy cost models for programming languages or libraries, mapping code constructs to estimated joules consumed, which allows "back-of-the-envelope" calculations without requiring specialized hardware[41].

While these modelling approaches provide valuable insights, they are often limited by their assumptions and calibration. Their accuracy depends on detailed hardware characterizations, which may not always be accessible for proprietary systems. Furthermore, most simulation frameworks are computationally expensive, making them impractical for use in continuous integration or production environments. Nevertheless, they remain an important complement to direct measurements, especially for what-if analyses, design exploration, and early-stage energy-aware software engineering.

Language Specific Energy Awareness Features and Measurements

Programming language and compiler research is advancing towards energy-aware software development. Extensions to LLVM and GCC have been proposed that integrate energy models into the compilation process, enabling energy annotations at the function level[42] though these approaches remain mostly in research prototypes.

Execution-time control features exist in languages like Ada and Java RTSJ, providing precise per-task CPU time measurements and budgets, but they are lacking in most mainstream languages. Since energy consumption is essentially power × time, having reliable task-level execution-time data makes it possible to attribute energy costs fairly across tasks. By combining CPU time measurements from language APIs with hardware counters such as RAPL or analytical models like McPAT, developers can estimate the energy consumption of individual tasks rather than just the processor package as a whole.

This capability enables task-level energy accounting, budgeting, and enforcement. For example, a CPU-time budget can be mapped directly to an energy budget, with overrun handlers acting as watchdogs for tasks that consume more energy than expected. This bridges the gap between hardware energy monitoring and software-level task awareness, providing the foundation for energy-aware programming models, green software practices, and sustainability reporting.

Language	Execution-Time API Support	Notes
Ada	Strong (Annex D, execution time clocks, timers) [43]	Hard real-time support
Java	With RTSJ / JSR-282[44]	Requires real-time JVM. An ongoing JDK Enhancement Proposal[45] adds CPU time monitoring for runtime profiling tools, but not as part of a standard API.
C/C++	Via POSIX/RTOS APIs	Not in language standard
Erlang (RT-Erlang)	Soft real-time process reductions	Not hard enforcement

.NET (specialized runtimes)	Limited research/embedded	Not in standard C#/CLR

Cloud Energy Measurements

In addition to established techniques such as RAPL-based profiling and wall-plug measurements, recent developments have extended energy measurement into cloud-native and multi-tenant environments. For instance, Kepler (Kubernetes Efficient Power Level Exporter)[46] and Cloud Carbon Footprint[47] provide open-source methods to attribute energy use and carbon emissions to individual pods, containers, or cloud accounts. These tools address the limitations of traditional profiling, which often cannot separate energy consumption in virtualized or containerized infrastructures.

Major industry platforms are also embedding sustainability features: Microsoft's Sustainability Calculator (Azure), Google's carbon-intelligent load shifting[48], and AWS/GCP's publication of region-level carbon intensity data[49] exemplify how providers are enabling greener workload placement.

Barriers to Measurement

As this section shows it is possible to give a developer (or development team) a good idea of the energy consumption of running code, but there are barriers to truly accurate measurements which must be understood.

First, since modern computer systems are nondeterministic it is necessary to run any tests several times, using statistical methods to find averages, means, etc. as such there will always be some variance in outcome.

Next, as tools monitor the energy consumption of the system, either the entire computer or the CPU, then (outside of highly contrived test scenarios) it is not possible to be completely sure that the energy measured is due to the execution of a particular piece of code. Certainty may be improved through stress testing methods but again there needs to be an acceptable level of certainty defined.

Then, to reduce jitter in the measurements it is recommended to stop as many background processes as possible, turn off Wi-Fi and other radios, turn of network connections, turn off power saving modes and turbo boost features, keep the computing environment at a stable temperature and thus create a lab environment where as many variables as possible are controlled.

This is perfect for comparative studies where two systems or even two versions of the same system, e.g. the old version vs. the energy optimized version, are compared but might not reflect real-world scenarios. Therefore sensible test configurations need to be developed.

And finally, if relying on Software Carbon Intensity (SCI) as a proxy for energy we need to understand that despite being the ISO standard it is a relatively simple formula. The Software Carbon Intensity (SCI) Specification is rather weak on how energy consumption from server-based software systems should be measured and it is especially weak on how energy consumption of software running in multi-tenant environments should be accounted for.

In summary, existing methodologies lack the rigor and consistency necessary for fair comparisons across various software applications and device configurations. Where possible these should be improved. Where impossible then the limitations should be clearly stated.

Measurement Tools, Benchmarks and Techniques

A broad ecosystem of tools supports software energy measurement. Open-source solutions such as CodeCarbon[121], Scaphandre[131], and Kepler[46] provide system-level monitoring, while Intel Power Profiler[50], ARM Streamline[51], and RAPL-based approaches[17] deliver hardware-level granularity. Hybrid methods that combine hardware counters with machine learning prediction models represent a leading edge, enabling estimation of energy use for specific code paths with higher accuracy. Sector-specific initiatives like DIMPACT[52] extend measurement to end-user services, particularly in high-bandwidth media applications.

The following tables represent state of the art tools and benchmarks which will be useful in the GreenCode project.

Hardware and System-Level Profilers

Hardware and system-level profilers provide low-level telemetry directly from the operating system and underlying platform (e.g., CPU power domains, performance counters, frequency and idle states). They are often the most accurate and lowest-overhead way to observe energy-relevant behaviour at runtime, making them valuable for calibration, benchmarking, and identifying coarse-grained hotspots, but they typically offer limited attribution to individual applications or code paths and may require privileged access and specialist expertise. The following table summarises representative tools.

Tool	Year / Origin	Platform / Scope	Key Features	Relevance	Challenges / Gaps
perf	Longstanding Linux kernel tool	Linux, system-wide	Low-level performance counters incl. energy (if RAPL enabled)	Ubiquitous in performance profiling; foundation for research tools	Very low-level; requires expertise; not developer-friendly; limited portability
powercap (Linux power capping framework)	~2010s	Linux (kernel interface)	Exposes CPU RAPL counters; allows capping power usage	Provides raw energy data; used by frameworks like Scaphandre	Limited scope (CPU/DRAM); not end-user facing
turbostat	Intel tool	Linux (Intel CPUs)	Reports CPU frequencies, C-states, package power	Useful for fine-grained CPU energy studies	Intel-only; not integrated with application-level metrics
Intel Power Gadget (IPG)	Intel official tool	Windows & macOS (Intel CPUs)	GUI + API for real-time CPU power/temperature	Widely used in Windows/macOS benchmarking	Intel-only; no container/process attribution
Libre Hardware Monitor (LHM)	Community fork of Open Hardware Monitor	Windows	System telemetry (CPU, GPU, disks, sensors); logs power metrics	Developer-friendly alternative for Windows; integrates with dashboards	Accuracy varies by hardware; not purpose-built for software attribution

Software Energy Measurement Frameworks

Recent work has produced a new generation of software-centric energy measurement frameworks. Unlike earlier hardware-oriented benchmarks these tools focus on application and workload-level measurement, enabling developers and operators to quantify and optimise energy use in real systems. The following table summarises these:

Framework / Tool	Year	Platform / Scope	Key Features	Relevance / Notes	Challenges / Gaps
CodeCarbon	2020	ML workloads (PyTorch, TensorFlow, Hugging Face)	Estimates runtime energy use; maps to grid carbon intensity; reproducible CO_2 metrics	Widely adopted in ML research and industry; training/inference footprints	Limited to ML; accuracy depends on assumptions about hardware and grid mix
Scaphandre	2020	Linux, processes & containers	Uses Intel RAPL (CPU/DRAM); exports to Prometheus/StatsD	Designed for observability pipelines; validated in cloud & virtualised envs	Linux-only; RAPL support may be restricted on some CPUs; limited GPU coverage
Kepler (Kubernetes Power Level Exporter)	2022	Kubernetes / CNCF	Pod- & container-level estimates; CPU/GPU mapping to power; cluster-wide metrics	Cloud-native monitoring; CNCF integration	Still maturing; calibration issues in heterogeneous clusters; limited outside Kubernetes
PowerProf (Windows Energy Estimation Engine (E3) / ETW-based analysis)	2023	Windows (process/thread)	Energy attribution accessed via OS-level energy estimation (e.g., Energy Estimation Engine (E3) surfaced through Windows performance tracing/analysis workflows) rather than a single dominant open-source tool comparable to RAPL-based Linux stacks.	Fills Windows gap; fine-grained feedback for enterprise/desktop apps	Windows-only; still emerging; fewer community integrations vs Linux profilers

PowerAPI	Updated 2024	Research-driven, heterogeneous	Software-defined power meters; calibrated energy models at process/microservice level	Flexible, fine-grained attribution; strong for experiments	Research-focused; requires calibration; limited mainstream adoption
Carbon Aware SDK (GSF)	2022	Cross-platform, grid-aware	Links workloads to carbon intensity signals; enables carbon-aware scheduling	Complements measurement with carbon-shifting strategies	Does not measure energy directly; adoption in production pipelines is limited
HF Carbon Tracker & ML CO_2 Impact Calculator	2020s	ML ecosystem	Integrated into training pipelines; emissions reporting; reproducibility support	Widely cited in research; helps benchmark ML impacts	ML-specific; not a general-purpose profiler; depends on model reporting
Cloud Carbon Footprint	2020s	Cloud workloads	Estimates footprint from cloud provider usage metrics	Useful for sustainability reporting; complements ML/Linux profilers	Relies on provider APIs; accuracy limited by cloud reporting granularity
EnergiBridge	2023	Cross-platform (Linux, Windows, macOS)	Supports Intel, AMD, Apple ARM CPUs; broad hardware coverage	First cross-platform tool covering diverse CPU architectures	Still new; adoption low; GPU/accelerator support limited
CPPJoules	2024	C++ workloads on Intel systems	Built on Intel RAPL; energy attribution for compiled C++ code	Extends profiling to C++ beyond ML ecosystems	Intel-only; requires RAPL access; limited to C++ scope
PowerSensor 3	2025	SoC boards, GPUs, FPGAs, accelerators	Open-hardware + software; real-time power measurement up to 20 kHz	High-resolution profiling of accelerators; suitable for AI workloads	Hardware-dependent; requires instrumentation; less accessible for average devs
GreenFrame	~2023	Web applications	Scenario-based testing; CI/CD integration; energy/carbon reporting	Developer-friendly; brings sustainability into web/frontend workflows	Focused on web stack; limited to scenario testing; accuracy depends on test design

Measurement Methods: Accuracy & Practicality

Different measurement methods trade off accuracy, granularity, and operational practicality. Some approaches (e.g., external power meters) provide high-fidelity whole-system readings but offer little attribution, while others (e.g., RAPL, GPU telemetry, and software-defined meters) improve temporal resolution and can support finer-grained allocation, but introduce platform constraints, calibration requirements, or uncertainties, especially in virtualised and cloud environments.

The table below summarises the main measurement options, the level at which they operate, their typical caveats, and the tools most commonly used in practice, helping GreenCode select "fit-for-purpose" measurement modes with clear confidence bounds.

Method	Granularity	Accuracy & caveats	Overhead/practicality	Representative tools
External AC power meters	Whole machine	High accuracy; but cannot attribute per process/app	Requires physical access; not scalable	Used for calibration in studies
RAPL / CPU energy counters	CPU package, DRAM	Good for relative trends; Intel/AMD only; limited GPU coverage	Very low; kernel/sysfs APIs	Basis for Scaphandre, PowerAPI
GPU vendor telemetry	GPU board and/or subsystem	Reasonable accuracy; polling resolution varies	Low; CLI/API access	NVML[53], DCGM[54] (NVIDIA); AMD SMI[55]
Software-defined meters (model-based)	Process and/or service and/or container	Requires calibration; attribution possible	Moderate setup; scalable	PowerAPI, Scaphandre, Kepler
Windows energy estimation (E3 + PowerProf)	Process / thread (Windows desktop/server)	OS model–based estimation; depends on calibration / hardware; less transparent than counter-based methods.	low; ETW tracing can be enabled on demand; suitable for endpoint and server profiling.	PowerProf[56]; Windows Performance Toolkit (WPR/WPA); ETW (xperf).
Carbon-intensity aware meters	Job and/or run level	Energy × regional carbon factor; accuracy depends on energy input quality	Low overhead; library/SDK integration	CodeCarbon, Carbon Aware SDK
Java RTSJ profiling	JVM-level	Potential for deterministic real-time metrics; but mainstream JVMs do not support RTSJ	Limited to niche VMs (e.g., JamaicaVM); profiling tools immature and not production-ready	JamaicaVM, RTSJ profilers (research only)
Cloud environment limitation	VM / managed service	Hardware-counter–based tools (RAPL, NVML, PowerProf) generally	Restricts measurement to on-premise or bare metal unless estimation models are used	Kepler (K8s), Cloud Carbon Footprint[57],

	inaccessible in cloud; accuracy depends on provider telemetry if available	provider APIs (limited)

Limitations and Gaps

Despite rapid progress in tooling and methods, energy measurement and attribution still face practical barriers that limit repeatability and actionability in real engineering workflows. The limitations are not only about sensor accuracy, but also about comparability across toolchains, measurement overhead, restricted access in cloud and multi-tenant environments, and ambiguity in allocating shared energy costs to specific components or code paths.

Cross-layer telemetry often fails to align cleanly (hardware < > OS < > container < > application), and carbon-intensity integration introduces additional timing and boundary assumptions. The table below summarises the key gaps, their real-world implications, and how GreenCode could address them through explicit boundaries, confidence grading, and workflow-integrated evidence generation.

Limitation/Gap	Description	Example/Implication	Application To GreenCode
Measurement accuracy & comparability	RAPL (Linux), NVML (GPU), PowerProf (Windows), Kepler (K8s) each vary in scope/accuracy.	Teams can't agree on results or compare across toolchains.	Define validation protocols and error bars within GreenCode.
Overhead limits continuous use	Many profilers add non-trivial runtime overhead, making them unsuitable for everyday development or production monitoring.	PowerAPI and Scaphandre can distort performance benchmarks if run continuously.	GreenCode should aim for lightweight profiling modes or integrate predictive estimation for development workflows.
Limited applicability in cloud environments	Tools relying on hardware counters (RAPL, NVML, PowerProf) cannot be used in most public cloud or virtualised settings.	Developers running workloads on AWS, Azure, or GCP lack low-level access, making energy measurement difficult.	GreenCode should combine cloud-native profilers (e.g., Kepler) with estimation models and integrate available provider telemetry.
Attribution boundary ambiguity	Even with good meters, attributing energy to a specific component/function is uncertain under concurrency and shared resources.	"Optimisation" may just shift energy to another process/thread or hide in idle allocation.	GreenCode should define attribution rules (baselines, shared-cost allocation, confidence levels).
Cloud/managed-service measurement limits	Hardware counters are often unavailable; provider telemetry can be coarse, delayed, or opaque.	Constrains validation in cloud-native pipelines; reliance on estimation increases uncertainty.	Add "cloud-grade" measurement modes with documented confidence + calibration options.
Cross-layer telemetry inconsistency	Energy signals differ across hardware/OS/container/app layers and don't align cleanly.	Counter scopes can overlap or omit key components (GPU/network/storage).	Build reconciliation logic + consistent

			scopes for end-to-end tracing.
Measurement overhead and perturbation	Instrumentation can change runtime behaviour (and therefore energy) or add non-trivial overhead.	"Observed" savings disappear in production or at scale.	Require lightweight collection + A/B protocols that quantify measurement overhead.
Carbon-intensity alignment and timing	Carbon factors are time/location dependent and can be mismatched with energy sampling windows.	Errors when multiplying energy by a coarse carbon value.	Align time-series energy with regional carbon data; document assumptions.
Weakness in Multi-tenant Measurement	SCI standard does not adequately handle shared environments like cloud/multi-tenant apps	Difficult to assign fair energy/emissions to one software service	GreenCode could extend SCI with methods to apportion emissions in multi-tenant cloud contexts

Non-Priorities

Given GreenCode's aim to deliver a practical, workflow-integrated optimisation capability, it is equally important to be explicit about what the project will not attempt to solve in the first instance. Some challenges, such as defining universal metrics that hold across all workloads and platforms, creating perfect per-function attribution in highly virtualised environments, or replacing established infrastructure monitoring stacks, either sit outside the project's scope or would dilute effort away from building an evidence-driven optimisation loop.

The table below therefore captures non-priorities to keep the work focused on "fit-for-purpose" measurement and attribution with transparent boundaries and confidence, rather than pursuing theoretical completeness.

Area	Reason to De-prioritise	Implication
Hardware-only benchmarks (SPECpower, Green500)	Server-level or HPC-centric, not actionable at developer/app level.	Use for context, but not a GreenCode measurement focus.
Building new low-level power profilers from scratch	The SotA already provides mature profilers/counters; GreenCode value is integration + attribution + workflow usability.	Focus on consuming existing signals (RAPL/NVML/E3/Kepler/etc.) and standardising "measurement modes" + confidence.
Consumer/end-user device power profiling at scale	Device heterogeneity makes results non-comparable and hard to standardise at project level.	Prioritise reproducible server/edge/cloud measurement contexts where GreenCode can validate methods.

Summary and Future Opportunities for GreenCode

Energy measurement for standalone systems and highly specific lab testbeds is reasonably mature. The SotA is moving towards cloud-native, AI-intensive, and multi-tenant environments, where attribution, comparability, and actionable feedback remain open challenges.

GreenCode: State of the Art Review

The new software energy measurement frameworks demonstrate clear potential but remain fragmented, vary in accuracy, and lack comparability across contexts.

Industrial validation is sparse: while case studies exist in domains such as IoT, cloud storage, and AI/ML, there are no shared benchmarks equivalent to MLPerf (for example) for software sustainability.

The most significant opportunities lie in consolidating fragmented tools into a unified sustainability toolkit, embedding green testing within CI/CD workflows, and providing IDE-level developer feedback.

GreenCode can also lead in establishing reproducible benchmarks and KPIs, extending maintenance practices to account for "green debt," and integrating sustainability heuristics into AI-assisted coding.

In doing so, GreenCode can bridge the gap between measurement and actionable optimisation, positioning itself as both a consolidator of existing tools and a catalyst for industry-wide standards in sustainable software development.

Opportunities

The gaps above point directly to high-impact opportunities for GreenCode to advance the SotA. In particular, there is clear leverage in making measurement *repeatable and comparable*, improving *actionable attribution* from system telemetry to software artefacts, and embedding energy/carbon evidence into normal engineering workflows (CI/CD, observability, and governance reporting). Rather than pursuing "perfect" measurement, GreenCode can focus on pragmatic advances such as confidence scoring, boundary-aware baselining, hybrid metering strategies (hardware + software), and automated hotspot-to-change recommendations with regression validation.

The table below summarises the most relevant opportunity areas and how they translate into actionable project directions.

Opportunity	Description	Example / Implication	Application to GreenCode
Fine-grained software energy attribution	Develop methods to attribute energy consumption to functions, modules, services, or architectural elements rather than whole processes or machines.	Enables identification of "energy hotspots" within large codebases instead of coarse task or application-level averages.	GreenCode can generate function- and component-level energy maps to guide targeted optimisation and refactoring.
Cross-layer energy correlation models	Link measurements across hardware, OS, runtime, container, and application layers into unified models.	Helps explain why identical code behaves differently across runtimes, clouds, or hardware.	GreenCode can fuse RAPL, OS metrics, runtime telemetry, and code structure into interpretable energy models.

Measurement under abstraction (cloud, serverless, containers)	Address loss of visibility caused by cloud abstraction and managed services.	Serverless and managed platforms obscure infrastructure energy costs from developers.	GreenCode can infer energy usage via controlled benchmarking, workload profiling, and provider-specific estimation models.
Predictive energy estimation at design time	Use static analysis and learned models to estimate energy impact before execution.	Allows early rejection of inefficient designs without costly runtime testing.	GreenCode can provide pre-deployment energy estimates during code review or architecture design phases.
Standardised energy metrics for software comparison	Define comparable, reproducible metrics for evaluating software energy efficiency.	Enables “apples-to-apples” comparison between languages, frameworks, and implementations.	GreenCode can help define and validate benchmark suites aligned with SCI and future certification schemes.
Benchmark suites and KPIs for software sustainability	Establish reproducible measures (benchmarks + KPIs) usable across tools/teams.	Equivalent to MLPerf-style comparability for software sustainability.	Publish benchmark protocols/results; enable regression baselines per system type.
Domain-specific KPI library	KPI templates tailored to domains (IoT, web, ML/LLMs, etc.).	Makes metrics actionable and comparable within a domain.	Provide KPI packs + measurement guidance per deployment context.
Tiered measurement modes with confidence reporting	Define “best available” measurement modes (direct / estimated / proxy) with explicit confidence.	Avoids false precision; improves cross-team trust and repeatability.	Make confidence/uncertainty first-class in reporting + optimisation decisions.

GreenCode: State of the Art Review

AI Tooling for Code and Energy Optimisation

Contributing Partner(s): Fraunhofer IESE, Digital Tactics, Kings College London, Siemens, TWT, UC3M, Panel Sistemas

Background

AI-assisted development has evolved from autocomplete, rule-based linters, and static analysis into generative code assistants that can propose, transform, and explain code, and increasingly support quality assurance and security remediation inside mainstream developer workflows. Since the release of GitHub Copilot (2021) and the broad adoption of conversational LLM interfaces (late 2022 onwards), tooling has expanded rapidly from IDE copilots toward multi-step pipelines that can operate across repositories and attempt larger-scale modifications.

However, most of this ecosystem remains optimised primarily for correctness, speed, and developer productivity. Energy, carbon, and runtime efficiency impacts are often treated as secondary effects, assessed via coarse proxies, or omitted entirely because they require workload-specific measurement and careful attribution. For GreenCode, the central challenge is therefore to move from “helpful suggestions” to closed-loop optimisation: define system and workload boundaries, establish a reproducible baseline, generate AI-assisted changes, and validate them with regression guardrails for correctness, performance, security, and maintainability before adoption in production. This framing is particularly important for legacy estates, where heterogeneity, operational constraints, and accumulated technical debt mean that efficiency improvements must be evidence-led and safe to deploy at scale.

For GreenCode, the key question is not whether AI can generate code, but whether it can reduce operational energy and cost without degrading correctness, performance, security, or maintainability. This shifts emphasis from IDE-level convenience to measured, repeatable optimisation workflows that operate at repository and system scale. In practice, this means coupling AI-generated suggestions to profiling and attribution, applying automated regression checks, and only then promoting changes into production. This also implies treating measurement boundaries, workload representativeness, and evidence quality as first-class concerns, so that efficiency gains are credible, comparable, and auditable.

GreenCode: State of the Art Review

Current State of the Art

Model Foundations

Small Language Models (SLMs) for Code Generation

Techniques for obtaining Small Language Models were surveyed in papers: Wang et al.[58], Subramanian et al.[59], Nguyen et al.[60], or Zhang et al.[61] providing comprehensive overviews.

While many benchmarks exist for assessing the functional performances of LLM's for code generation only few papers target SLMs. Benchmarks primarily evaluate software engineering tasks (e.g. change summarisation, classification, refinement) and may optionally be extended with energy measurement overlays, which are not yet applied consistently across studies.

Hasan et al.[62] assessed 20 SLMs (up to 9B parameters) on five benchmarks (HumanEval, MBPP, Mercury, CodeXGLUE (2CT) and HumanEvalPack) that represents code generation tasks in seven languages Python, Java, JavaScript, PHP, Ruby, Go, and C++.

The Qwen2.5-Coder models (1.5B, 3.0B, and 7.0B) were the best on in their categories (≤1.5B, >1.5B to ≤3B, and >3B to ≤10B). In terms of efficiency, they measured inference time and VRAM consumption but not energy consumption. Souza et al.[63] evaluated five open-source models (from 3B to 14B) on Python generation tasks (neither code efficiency nor energy consumption were measured).

Cho et al.[64] used fine tuning (self-correction) to improve the performances of very small models (<3B). Showing the potential of such approach.

Cong et al.[65] assessed the performance of three open-source SLMs for code change tasks: change summarization (i.e., commit message generation), change classification, and code refinement (e.g., during for code reviews). They also proposed a benchmark datasets for these tasks covering five programming languages: C++, Java, JavaScript, Python, and Go. Other papers study relevant maintenance tasks such as vulnerability detections[66] or bug detections and refactoring[67].

Energy consumption of SLMs was studied in Durán et al.[68] However the analysed models are now obsolete.

Large Language Models (LLMs) for Software Engineering Tasks

Software maintenance and testing are categories where research is relatively important[69]. More recently Braberman et al.[70] analysed and categorized prompting techniques for software engineering. Both works provide insights into existing gaps within the scientific literature, facilitate the rapid identification of pertinent articles, and offer an understanding of prompt engineering techniques along with empirical results across various tasks.

The figure below summarises recent research on LLMs for software engineering categorized by SWEBOK categories and demonstrates clear gaps in software requirements and design, configuration management, software engineering process, models, practice and software quality generally.

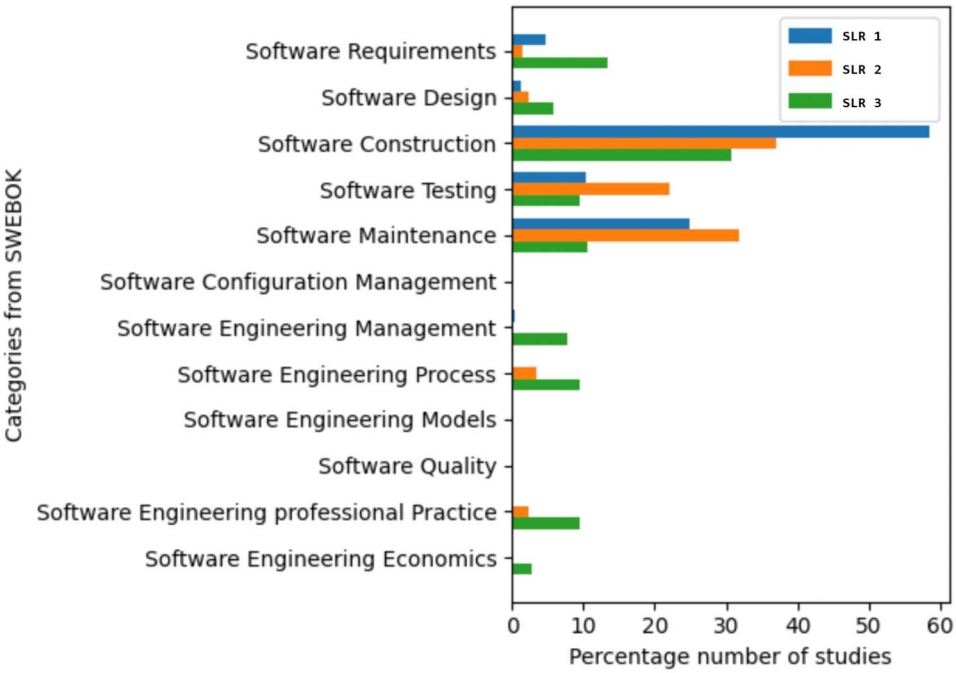

Figure . Number of LLMs for software engineering identified from research papers categorized by SWEBOK categories. Source Master Thesis Shana S. Goudar (2024) A Systematic Evaluation of Prompts for LLM-assisted Software Engineering Tasks. Rheinland-Pfälzische Technische Universität Kaiserslautern (RPTU).
SLR 1: Hou et al. *(2024). Large Language Models for Software Engineering: A Systematic Literature Review. ACM Trans. Softw. Eng. Methodol. 33, 8, Article 220. https://doi.org/10.1145/3695988.*
SLR 2: *Fan et al. (2023). Large language models for software engineering: Survey and open problems. arXiv preprint arXiv:2310.03533.*
SLR 3: *Nguyen-Duc et al. (2023). Generative AI for software engineering-a research agenda. arXiv preprint arXiv:2310.18648.*

In a recent paper, Cheung et al.[71], compared the carbon footprint of manual coding with LLM-assisted coding. Their initial findings indicate that LLM-assisted coding can have substantially (20- to 40-fold) larger carbon footprints than manual coding for the cases tested, and the absolute gap between the carbon footprint of the two approaches increases with the complexity of the task.

To mitigate this they recommend human-in-the-loop LLM-assisted processes where the tasks are broken into simpler tasks before moving on to LLM-assisted coding.

Mixture of Experts (MOE) for Code Generation and Optimization

Mixture of Experts (MOE) architectures represent a paradigm shift in machine learning model design, offering significant potential for sustainable AI-driven code optimization. At its core, MOE employs multiple specialized sub-models (experts) coordinated by a gating network that routes inputs to the most appropriate experts[72]. This approach fundamentally aligns with green software principles by activating only necessary computational resources, thereby reducing energy consumption during inference while maintaining or improving performance.

The relevance of MOE to green software development stems from its sparse activation property, only a subset of experts processes each input, dramatically reducing computational overhead compared to dense models of equivalent capacity[73]. For code generation and optimization tasks, this efficiency

gain becomes particularly valuable as it enables deployment of sophisticated AI assistance tools with reduced environmental impact, supporting the GreenCode project's mission of sustainable software development.

Recent advances in MOE for code generation have demonstrated substantial improvements in both efficiency and specialization. Mixtral 8x7B represents a breakthrough in sparse MOE architectures, introducing an 8-expert system where only 2 experts are activated per token, achieving 47B parameter capacity while using only 13B active parameters during inference[74]. The model vastly outperforms Llama 2 70B on mathematics, code generation, and multilingual benchmarks, demonstrating the effectiveness of expert specialization for programming tasks.

Switch Transformer[73] established the foundation for modern MOE scaling, demonstrating that expert routing can achieve performance comparable to dense models while using significantly fewer parameters during inference. This sparse activation approach has become the standard for current MOE implementations in code generation systems.

Recent comprehensive surveys highlight the rapid evolution of MOE in LLMs, with particular emphasis on their application to code-related tasks[75]. These surveys indicate that MOE models have emerged as an effective method for substantially scaling up model capacity with minimal computation overhead, gaining significant attention from both academia and industry.

Programming Language-Specific Expert Specialization

Recent research is moving beyond “generalist” code LLMs by explicitly separating shared programming knowledge from programming-language-specific knowledge using sparse expert or adapter structures. For example, MOLE (Mix-of-Language-Experts) introduces shared and per-language LoRA adapters with automatic language routing, improving multilingual programming performance while remaining more parameter-efficient than maintaining separate per-language models[76].

MultiPL-MoE similarly proposes a hybrid MoE design (combining token- and segment-level expert selection) to improve multi-programming-lingual code generation under constrained compute budgets. Related work[77] shows MoE-based training/merging schemes can also materially improve code instruction-tuning outcomes across standard code benchmarks (e.g., HumanEval/HumanEval+, MBPP+, DS-1000), suggesting that expert routing can benefit not only “which language,” but also “which capability regime” within code tasks[78].

Outside software engineering, “contextual MoE” work demonstrates how explicit process knowledge can be injected into expert structures to improve predictive performance and interpretability; analogous context signals (e.g., repo conventions, build/runtime configuration, architectural constraints) could be explored for software engineering workflows as an open research direction[79].

Energy-Efficient Inference Frameworks

Recent surveys on inference optimization techniques for MOE models reveal significant advances in energy efficiency. Model-level optimizations include efficient expert design, compression techniques

such as pruning and quantization, and dynamic routing strategies that reduce computational overhead. These optimizations can achieve 40-60% reduction in inference energy consumption compared to dense model deployments[80].

System-level optimizations focus on distributed computing approaches and load balancing mechanisms[81]. Recent MoE inference work shows that uneven routing can create expert load imbalance and communication bottlenecks[82], and that techniques such as dynamic load balancing and expert sharding/placement improve utilization and end-to-end serving efficiency[83] [84]. Given evidence that inference energy is strongly driven by runtime and GPU-hours / time-at-power in deployment settings, these utilization gains can translate into lower energy per served token at scale[85] [86] [87].

Advanced MOE Techniques

Hierarchical Routing Mixture of Experts (HRME) represents an evolution in MOE architectures, employing tree-structured routing with classifiers as non-leaf nodes and regression models as leaf nodes[88]. This hierarchical approach enables more sophisticated expert specialization for complex programming tasks.

Expert merging and pruning techniques have emerged as key optimization strategies[89]. These methods identify and group similar experts in feature space, then merge experts within the same group to reduce computational overhead while maintaining performance quality.

Understanding and Quality Analysis

Static Analysis and Code Quality Tools

Traditional static analysis tools remain foundational in improving maintainability, security, and efficiency of code. Tools such as SonarQube, PMD, ESLint, and Pylint detect inefficient constructs (e.g., nested loops, redundant calculations) and security flaws before runtime. While most are not explicitly energy-focused, recent research has demonstrated that inefficient patterns correlate with higher energy consumption. For instance, Pinto et al.[90] showed that energy-inefficient programming practices can be detected via static metrics such as cyclomatic complexity and method call depth. Emerging AI-driven extensions (e.g., Sonar AI CodeFix, Snyk Code) now provide automated patch suggestions, reducing developer overhead while maintaining code quality. However, a gap remains in extending these static checks with explicit energy-aware rule sets.

ML in Software Quality Analysis

In recent years, advancements in machine learning (ML) and AI have come together for the effective analysis of software quality and several state-of-the-art reviews now exist. For instance, Alaswad and Poovammal[91] emphasize the role of ML techniques and used software quality metrics to predict the reliability of software. Moreover, Shafiq et al.[92] (~50% of 263 references) identified ML tools and techniques for concrete software lifecycle stages for quality assurance and analytics, i.e., applications about the prediction of faults, bugs, or defects using ML techniques, most of them being supervised.

Shafiq et al. also report that artificial neural network was the most used ML technique (30 articles), followed by support vector machines (28 articles), random forest (24 articles), and naive bayes (21 articles). Focusing on the identification of code smells i.e. design flaws in the source code, Caram et al.[93] and Azeem et al.[94] provide insights into various ML methods used for detecting code smells, with decision trees, random forest and support vector machines being among the most effective techniques. A crucial aspect of improving the prediction accuracy of the ML models is the selection of relevant features as highlighted by Alsolai and Roper[95].

Besides the individual application of concrete ML models and different techniques, various tools exist that support software quality analysis, i.e. they can help to identify errors, security vulnerabilities, and violations of coding standards to assess and improve code quality.[96] Tools focus on different programming languages and vary regarding their objectives including different features. As ML approaches can provide benefits, several tools integrate AI features nowadays, for example, to enhance analysis accuracy, such as SonarQube[97] or Snyk[98]. Moreover, generative AI can also play a role in software quality analysis and tools, particularly in code generation and automated refactoring (as indicated in the previous sections).

Change generation and safety nets

Automated Refactoring and Maintenance

Refactoring has long been supported in industrial IDEs such as IntelliJ IDEA, Eclipse, and Visual Studio, which offer transformations like method extraction, loop unrolling, and redundant code elimination. Open-source tools like Spoon[99] and OpenRewrite[100] enable large-scale automated refactoring across codebases. Recent research investigates the intersection of AI and refactoring: Gheyi et al.[101] show that small LLMs can effectively detect refactoring bugs, while Nguyen-Duc et al.[102] highlight the use of generative AI in proposing code changes. Despite this progress, few systems optimise explicitly for energy and performance efficiency, leaving a gap for sustainability-oriented refactoring.

Code Modernisation and Translation

The ongoing systematic literature review associated with the Master Thesis by Eshwar Sastry at RPTU[103] indicates that limited research has been conducted on code modernisation using LLMs, with only 30 relevant papers identified. Most of these studies focus on code translation (25 out of 30), while deployment strategies (3 out of 30) and architectural enhancements (2 out of 30) are areas that have received comparatively less attention.

The approaches taken by these first reviewed papers show that iterative approaches (workflow) or agentic ones are often more performant. These often includes steps like code translation, unit test generation, iterative error fixing (syntax of functional error), control-flow or call graph analysis.

The table below provides an overview of the papers analysed so far with their input and output languages.

Reference	Input Language	Output Language
Liu et al.[104]	C, C++	Rust
Hong et al.[105]	C	Rust
Eniser et al.[106]	C, Go	Rust
Pietrini et al.[107]	Fortran	Python
Chen et al.[108]	Fortran	C++
Fischer-Heselhaus et al.[109]	COBOL	Java
Kumar et al. 2024[110]	COBOL	Java
Kumar et al 2025[111]	C#	Java
Yuan et al.[112]	Python	Java

Code Documentation

Generating code documentation is less researched nowadays because generative AI solutions can now document snippets of code or files quite well. Thie functionality is already either integrated in IDE[113,114] or as a standalone tool[115]. One aspect of documentation generation that is still researched is the capability to generate documentation on a repository level (and not on a file or code snippet level). Luo et al.[116] proposed RepoAgent a multiagent system that can generate code for a full repository. Diggs et al.[117] and Dvivedi et al.[118] Both analyse the quality of the generated documentation in terms of readability, relevance (usefulness), and completeness and found LLM generated documentation to be on par with human generated documentation.

While various documentation tools handle specific aspects of the documentation process well, it remains necessary to use multiple tools for complete and thorough coverage. Connecting these solutions as part of its codebase understanding processes could be a useful outcome from GreenCode.

Generative AI and Testing

Software Testing is among the most researched SWEBOK categories, with over 180 papers to examine as part of this review of the SotA. Existing reviews such as Wang et al.[119] or Zhang et al.[120] show that testing with LLMs has seen an exponential increase in the number of publications within the last years. Unit-Testing being the most explored type of tests (followed by System tests). Generating tests from code is, so far, the most explored test case generation techniques. A few but increasing number of papers now target other test case generation techniques like mutation testing, search-based testing (including fuzzing), specification-based testing and metamorphic testing. Often in these cases, the LLM is supported by or supports existing methods (for instance, generating metamorphic relationships, or mutation strategies). The vast majority of research papers focus on testing functional requirements.

In practice, LLM-generated tests are typically emitted into existing, mainstream testing stacks rather than new research harnesses, e.g., JUnit for JVM projects, pytest for Python, and Jest for JavaScript/TypeScript unit tests, with system/E2E tests commonly targeting Selenium/WebDriver, Playwright, or Cypress where applicable.

Tooling consensus is therefore more about an approach (generate-from-code, then run/fix iteratively in the project's native framework) than a single standard tool; widely used assistants such as GitHub Copilot and dedicated generators like Diffblue Cover reinforce this "LLM layer over existing frameworks" pattern.

A key gap for GreenCode is that, even where testing is well-covered, the sustainability link remains weak: most work targets functional correctness, while energy/performance regression testing and CI/CD-native "green regression" gates are still under-integrated.

Performance and Energy Testing Frameworks

Performance and energy testing frameworks are critical for quantifying software sustainability. CodeCarbon[121] and PowerAPI[122] allow developers to capture the energy footprint of workloads during development and CI/CD runs. Similarly, Intel RAPL counters[123] provide fine-grained energy measurements on CPUs, while Android Battery Historian supports energy profiling for mobile systems. Despite this, integration into DevOps pipelines is rare, meaning that energy regressions often go unnoticed. Research prototypes, such as GreenScaler[124], demonstrate the feasibility of automated energy regression detection, but adoption remains limited. Embedding such tools into mainstream CI/CD workflows is a significant opportunity.

Efficiency-First Code Optimisation

Limited research has specifically addressed code maintenance with an emphasis on energy optimization, of 180+ papers just 16 focussed on this topic. Among the 16 selected papers, the majority (11) focused on code generation, while only three addressed code translation; the remaining two were review articles. Most studies concentrated on Python (9 out of 16 papers), with other target languages including C++ (3), C (1), Java (1), and JavaScript (1). Whereas most studies measured the efficiency of the code generated by existing pretrained LLMs, 3 papers discuss techniques for fine-tuning LLMs for generating efficient code: Duan et al.[125] combine supervised fine-tuning with reinforcement learning to optimize LLM code output, Huang et al.[126] propose a self-improving code generator that leverages runtime profiling for iterative refinement, and Peng et al.[127] integrate profiling-guided prompting and energy-aware tuning strategies for sustainable code generation.

The subject is covered in further detail in the annual SotA review of static code analysis maintained by the project consortium.

System/Runtime Optimisation Platforms

Dynamic optimisation adapts code execution during runtime to improve efficiency. Traditional examples include the JVM HotSpot JIT and GraalVM[128], which optimise bytecode execution on-the-fly. In distributed systems, autoscaling frameworks such as Kubernetes Horizontal Pod Autoscaler

(HPA) reduce waste by matching resource allocation to demand. More recently, carbon-aware schedulers (e.g., Microsoft's Carbon Aware SDK[129]) optimise workload execution timing to coincide with periods of renewable energy availability. However, AI-driven approaches that couple runtime adaptation with software-level energy optimisation are nascent. This represents an opportunity for hybrid solutions that link code efficiency with platform-level sustainability.

Commercial and Open-Source Code Optimisation Ecosystem

A variety of industrial and open-source tools address code quality and efficiency, but their sustainability focus is inconsistent. Commercial products such as AWS CodeGuru, SonarQube/Sonar AI CodeFix, Veracode Fix, and Snyk Code deliver automated performance, quality, and security insights. Open-source projects such as EcoCode[130], Scaphandre[131], and Green Software Foundation's Carbon Aware SDK[129] support energy measurement and green scheduling. Research-driven projects like EnergyVis[132] provide visualisation frameworks for energy profiling. However, these ecosystems remain fragmented and rarely interoperable, limiting adoption. A unifying approach that bridges commercial and open-source sustainability tooling would have significant impact.

Evaluation

LLM Code Generation Benchmarks

The following table lists widely used benchmarks for evaluating LLM code generation across multiple programming languages, reflecting the shift from single-language (often Python-only) evaluation toward more realistic, polyglot software engineering settings. The listed benchmarks include function-level generation suites (e.g., HumanEval variants and MultiPL-E) as well as workflow-/repository-oriented benchmarks (e.g., multilingual SWE-Bench) that better approximate real maintenance tasks. Together, they help compare model capability, robustness, and generalisation across languages and problem types.

Name	URL	Languages
HumalEval-XL	https://github.com/FloatAI/humaneval-xl	python, java, javascript, c#, go, kotlin, perl, php, ruby, scala, swift, typescript
Aider Polyglot	https://github.com/Aider-AI/polyglot-benchmark	++, Go, Java, JavaScript, Python, Rust
SWE-Bench multilingual	https://www.swebench.com/multilingual.html	C, C++, Go, Java, JavaScript, TypeScript, PHP, Ruby and Rust
MultiPL-E	https://github.com/nuprl/MultiPL-E	22 languages

Code Translation Benchmarks

This table summarises benchmarks designed specifically for code translation, where the primary evaluation challenge is preserving functional equivalence under different language semantics, standard libraries, and idioms. The datasets captured here span broad multi-language translation (CodeTransOcean), repository-level translation tasks (RepoTransBench), and higher-structure translation such as class-level cases (ClassEval-T). These benchmarks are useful for assessing whether LLM-based translation supports modernisation workflows beyond line-by-line conversion,

particularly when correctness must be validated via compilation and tests rather than surface similarity.

Name	URL	Languages
CodeTransOcean	https://github.com/WeixiangYAN/CodeTransOcean	Python, C, C++, VisualBasic, Go, PHP, Java, C#, Swift, R, Rust, Fortran, Ada, Perl, COBOL, Lua
RepoTransBench	https://github.com/DeepSoftwareAnalytics/RepoTransBench	Python to Java
ClassEval-T	https://github.com/wLinHoo/ClassEval-T	Python, Java, C++

Code Efficiency Benchmarks

The following table compiles benchmarks that move beyond correctness to focus on code efficiency, typically via runtime- or resource-oriented signals (e.g., execution time, algorithmic efficiency, or performance under fixed inputs). While most efficiency benchmarks still use performance proxies rather than direct energy measurement, they provide a practical starting point for evaluating whether generated code is "good" not only in function but also in operational cost. For GreenCode, these benchmarks are especially relevant as candidates for augmentation with energy/carbon instrumentation overlays and repeatable protocols that connect efficiency outcomes to measurable sustainability impact.

Name	URL	Languages
Mercury	https://github.com/Elfsong/Mercury	Python
EffiBench-X	https://github.com/EffiBench/EffiBench-X	Python, JavaScript, C++, Java, Go, and Ruby
COFFE	https://github.com/JohnnyPeng18/Coffe/	Python
ENAMEL	https://github.com/q-rz/enamel	Python
EvalPerf	https://github.com/evalplus/evalplus	Python

GreenCode: State of the Art Review

Limitations and Gaps

Although AI-assisted software engineering is advancing rapidly, relatively little research treats code efficiency and energy consumption as first-class objectives. A recent survey of more than 180 papers on AI for code-related tasks found only 16 studies that directly addressed energy or efficiency, mostly focused on code generation and largely evaluated on Python, with limited attention to enterprise and legacy languages (e.g., COBOL, Java, C/C++). Most work assesses the efficiency of code produced by pretrained models rather than training or adapting models explicitly for efficiency, with only a small number of exploratory approaches (e.g., RL-assisted fine-tuning, profiling-guided self-improvement, and energy-aware prompting).

The following Limitations and Gaps table summarises the main shortcomings through a GreenCode (energy + performance + quality) lens. It highlights that energy cannot be reliably inferred from code alone, optimisation effects often fail to generalise across hardware/OS/workloads, and energy-labelled training data remains scarce. It also captures the barriers to industrialisation: evaluation still prioritises correctness over efficiency, AI-driven changes can add technical debt and reduce explainability, and many tools remain snippet- or IDE-centric rather than supporting measurement-in-the-loop workflows at repository and system scale. Finally, it notes adoption risks, prompt sensitivity, non-determinism, and unclear governance boundaries, which motivate GreenCode's closed-loop approach (measure > optimise > validate) with stability checks and sustainability-aware benchmarks.

Limitation / Gap	Description	Example / Implication	Application to GreenCode
Limited understanding of energy from code alone	Current AI models infer behaviour from syntax and patterns, not actual runtime energy consumption.	Code that appears efficient may behave inefficiently due to runtime, I/O, or hardware effects.	GreenCode must couple AI suggestions with empirical measurement and validation.
Weak generalisation across environments	AI-generated optimisations may not transfer across hardware, OS, runtimes, or workloads.	An optimisation valid in cloud CI may regress energy usage in production.	GreenCode should validate optimisations across representative execution contexts.
Sparse energy-labelled training data	Few datasets annotate code with reliable energy or carbon measurements.	Models optimise for proxies (complexity, style) rather than true energy.	GreenCode can help generate and curate energy-annotated corpora through benchmarking.
Evaluation benchmarks focus on correctness, not efficiency	Existing benchmarks (HumanEval, MBPP) ignore energy and performance.	AI tools appear successful despite producing inefficient code.	GreenCode can contribute energy-aware benchmarks and evaluation criteria.
Risk of AI-induced technical debt	AI-generated fixes may improve one metric while degrading maintainability, readability, or correctness.	Automated refactoring introduces subtle regressions.	GreenCode should include human-in-the-loop review and regression testing.

Limited explainability of AI decisions	Many AI tools cannot justify why a change improves efficiency.	Developers mistrust or blindly accept AI recommendations.	GreenCode can emphasise explainable routing, pattern attribution, and traceability.
Fragility on large, heterogeneous legacy systems	AI tools perform well on isolated snippets but struggle at repository or system scale.	Legacy COBOL or polyglot systems remain partially optimised.	GreenCode should focus on system-level workflows, not isolated code generation.
Over-reliance on static optimisation	AI suggestions are often applied statically without runtime feedback.	Optimisations may be invalid under real workloads.	GreenCode can close the loop between AI, runtime profiling, and re-optimisation.
Unclear responsibility boundaries	It is unclear when AI should act autonomously versus assistively.	Fully automated fixes may be unsafe in regulated or safety-critical systems.	GreenCode can support graded autonomy and governance-aware deployment modes.
Copilot-driven siloed optimisation	Current AI-assisted development tools are primarily embedded at the individual developer or IDE level, encouraging isolated, localised problem solving.	Energy or performance optimisations are applied to snippets without considering system-wide effects or interactions.	GreenCode can promote system-level optimisation by coordinating AI actions across repositories, components, and teams.
Limited support for workflow- and agent-based optimisation	Most AI tooling focuses on single-step code generation or refactoring rather than multi-stage, feedback-driven workflows.	Complex optimisation tasks requiring profiling, refactoring, validation, and iteration remain largely manual.	GreenCode can orchestrate agentic workflows that combine analysis, optimisation, and validation loops.
Language and ecosystem bias in AI code research	The majority of AI-assisted code efficiency research focuses on Python, with limited coverage of other widely used languages.	Findings may not generalise to enterprise systems written in COBOL, Java, C/C++, or mixed-language stacks.	GreenCode can target underrepresented languages and heterogeneous legacy codebases.
Underrepresentation of small and energy-efficient models	Research and tooling disproportionately focus on large language models, despite their high energy cost.	Potential gains from small models, task-specific models, or MoE architectures remain underexplored.	GreenCode can prioritise small, specialised models aligned with energy efficiency goals.
Prompt-level energy sensitivity in practice	Prompt tweaks can flip energy outcomes, even when code changes appear equivalent.	“Green prompting” can be unreliable across models/hardware.	Formalise prompt patterns + validate with measurement-in-the-loop.
Inconsistent optimisation outcomes in production workflows	AI-assisted “optimisation” can be non-deterministic across runs and contexts.	Teams struggle to operationalise and govern AI refactoring at scale.	Add stability checks, confidence scoring, and rollback mechanisms.

Many commercial code-assistance platforms prioritise IDE or near-real-time feedback (e.g., static analysis and AI suggestions surfaced during development). While valuable for developer experience, continuously invoking heavyweight analysis or LLM inference at edit time can be operationally costly

and difficult to govern at scale. A clear opportunity is to shift a significant portion of sustainability- and efficiency-oriented analysis to batch and review-time workflows (e.g., pull request, nightly, or release-candidate gates), where inference can be amortised across changes, measurement boundaries and baselines can be applied consistently, and proposed fixes can be validated with regression checks before expert review and acceptance. This mode also generalises naturally to legacy estates, enabling large-scale modernisation and optimisation programmes with a human-in-the-loop retained for oversight and accountability.

More broadly, most AI-for-SE solutions remain concentrated on QA and security outcomes. These are essential, but sustainability and energy efficiency are under-served and are likely to become a more explicit dimension of technical debt and governance. Over time, "debt" discussions are expected to expand beyond traditional categories (quality, test, documentation, architecture) to include sustainability-facing concerns such as energy/efficiency debt and green compliance debt, particularly as measurement and reporting requirements mature and optimisation becomes easier to operationalise.

Non-Priorities

Some directions, such as attempting to build a general-purpose replacement for IDE copilots, pursuing "perfect" energy prediction from static code in the absence of measurement, or competing with large commercial QA/security platforms on breadth, would dilute effort without resolving the core barriers to sustainability optimisation, as such GreenCode will not focus on these.

The non-priorities table below clarifies the activities and problem framings that are out of scope, so that the project concentrates on workflow-integrated, measurement-in-the-loop optimisation with repeatable validation and governance-ready evidence.

Area	Reason to De-prioritise	Implication
Generic IDE refactoring (readability, naming, formatting)	These are already well-covered by mainstream IDEs (IntelliJ, VS Code, Eclipse) and open-source tools (Prettier, ESLint, ReSharper).	Competing here would duplicate existing functionality without adding sustainability value.
Pure correctness / functional testing with LLMs	The field is crowded with hundreds of papers and tools (unit test generation, fuzzing, mutation testing). The sustainability link is weak.	Limited differentiation; GreenCode should instead focus on energy-aware testing.
General-purpose AI code assistants (like Copilot/CodeWhisperer clones)	Big players (Microsoft, Amazon, Google) dominate this market with rapid model improvements.	Building a competing assistant would be resource-heavy and low-impact; instead GreenCode should layer sustainability intelligence on top of these assistants if addressing them at all.
Obsolete small language models	Some SLMs reviewed in research (Durán et al. 2024) are already outdated and not practical for deployment.	Effort spent optimising obsolete models would not provide lasting advantage. Focus should be on efficient, modern SLMs tuned for energy-aware tasks.

Snapshot-only benchmarking of correctness	Benchmarks like HumanEval, MBPP, CodeXGLUE already cover correctness; repeating this adds little.	GreenCode should focus on benchmarks for energy/performance, not correctness alone.
Security-only tooling	Tools like Snyk, Veracode, and GitHub Dependabot already provide mature solutions.	Security is important, but GreenCode's differentiation lies in sustainability, not duplicating mature security workflows.
Highly speculative one-shot AI app generation	The report itself notes "mixed results" and high technical debt risk from one-shot app generation.	Chasing "full AI app generators" is risky and distracts from the sustainability angle where GreenCode has unique value.
Training complex MoE models from scratch	Training MoE requires massive compute resources, complex distributed setups, and careful balancing to avoid expert collapse. This is outside GreenCode's likely resource scope.	GreenCode risks over-investing in infrastructure-heavy R&D instead of delivering applied sustainability tools. Focus should be on leveraging pre-trained MoEs or small/efficient models.
Low-level MoE training research (e.g., expert collapse, routing stabilisation)	Research-heavy topics like hierarchical routing stability, collapse prevention, and convergence tuning are well-covered by academia and big labs (DeepMind, OpenAI, Google).	Competing here offers little differentiation; GreenCode should consume these advances rather than replicate them.
Hardware-level MoE optimisation frameworks	System-level work on load balancing, distributed inference, and GPU-level optimisations is more relevant to cloud providers and chip vendors.	GreenCode should focus instead on software-level applications of MoE (energy-aware code optimisation), not hardware R&D.
Generic code generation benchmarks (e.g., correctness-only benchmarks)	MoEs are already being benchmarked extensively for code correctness and performance by academic consortia.	GreenCode adds little value here — it should instead define sustainability-oriented benchmarks (energy/performance metrics).
Security- or purely functional MoE applications	Using MoE purely for security fixes, correctness, or bug detection is already well addressed by other AI tools.	GreenCode should avoid diluting focus and instead specialise on energy efficiency + maintainability.
Edge deployment of large MoEs	Deploying large-scale MoEs on IoT/mobile is impractical due to memory and routing overheads.	GreenCode should focus on lightweight SLMs or distilled models for edge/IoT sustainability rather than trying to squeeze full MoEs onto constrained devices.
Always-on, real-time IDE inference as primary delivery model	Bulk/review-time modes reduce inference burden and carbon/cost.	Prioritise batch/review-time analysis + human review gates over continuous IDE inference.

Summary and Future Opportunities

Recent evidence from ongoing systematic literature reviews further highlights that many of the most practically relevant opportunities for AI-assisted software sustainability remain underexplored. In particular, the literature on code modernisation using large language models remains limited, with recent reviews identifying only a small number of relevant studies, the majority of which focus on

code translation rather than on deployment strategies, architectural restructuring, or long-term maintainability. This gap is especially significant for legacy and large-scale enterprise systems, where sustainability gains are more likely to arise from coordinated, bulk analysis and modernisation of entire codebases rather than from isolated code generation tasks.

At the same time, the increasing use of copilot-style tools has led to highly siloed, developer-centric optimisation practices, creating new forms of AI-induced technical debt at team and organisational level. Emerging research indicates that iterative, workflow-based and agentic approaches—combining steps such as code translation, unit test generation, error correction, and control-flow or call-graph analysis—are often more effective than single-pass generation. However, such approaches are not yet widely supported by current tooling. Similarly, while small and specialised language models show promise for energy-efficient software engineering, existing benchmarks largely focus on functional correctness and performance, with limited consideration of energy consumption, and prior energy studies are often based on now-obsolete models. These gaps point toward the need for integrated, workflow-oriented AI systems capable of operating at repository and system scale, rather than at the level of individual code snippets.

Opportunities

The limitations identified in this section also reveal a set of high-leverage opportunities where GreenCode can materially advance the SotA. In particular, there is clear value in moving from "suggestions" to closed-loop optimisation: coupling AI-generated refactors and configuration changes with instrumented measurement, confidence grading, and regression validation for correctness, performance, security, and sustainability outcomes.

Additional opportunities include building sustainability-aware benchmarks and datasets, developing batch/review-time optimisation workflows that scale to legacy estates, and creating reusable optimisation patterns and guardrails that can be applied consistently across languages and domains.

The opportunities table below summarises these directions and how they translate into actionable project focus areas.

Future Opportunity	Description	Example / Implication	Application to GreenCode
Workflow- and agent-based optimisation pipelines	Shift from single-pass code generation to multi-stage, feedback-driven optimisation workflows.	Combine profiling, refactoring, testing, and validation iteratively.	GreenCode can orchestrate agentic workflows spanning analysis, optimisation, and validation.
Automated application of sustainability fixes	Apply energy and efficiency improvements using the same automation paradigms as code quality and security fixes.	Sustainability fixes become part of standard remediation pipelines.	GreenCode can auto-apply validated energy optimisations with human oversight.
Bulk analysis and modernisation of legacy codebases	Enable repository- and system-scale optimisation rather than snippet-level changes.	Legacy systems gain efficiency without full replatforming.	GreenCode can perform bulk modernisation across large, heterogeneous codebases.

Reconciliation of AI-generated code at team scale	Detect and rationalise inconsistencies and inefficiencies introduced by siloed copilot usage.	Pay down AI-induced technical debt across teams.	GreenCode can analyse and harmonise AI-generated code across repositories.
Energy-aware code maintenance and refactoring	Extend AI optimisation beyond generation to long-term maintenance activities.	Refactoring decisions consider energy and performance trade-offs.	GreenCode can prioritise energy-efficient maintenance actions.
Repository-level documentation and system understanding	Generate documentation at repository or system level rather than isolated snippets.	Improves maintainability and supports sustainability decisions.	GreenCode can produce energy-aware system documentation.
AI-assisted sustainable coding	Embed sustainability heuristics in AI-assisted code generation.	LLM suggests efficient libraries/structures by default.	Curate prompt libraries + adaptive models; expose via MCP/workflows.

GreenCode: State of the Art Review

Sustainable and Green AI

Contributing Partner(s): TWT, FH Aachen, Digital Tactics

Background

The rapid expansion of AI, particularly deep learning and LLMs, has made AI workloads a significant contributor to computing-related energy demand. “Green AI” refers to processes aimed at reducing the energy consumption and carbon footprint of AI across its lifecycle, from model development (training and tuning), deployment (inference), to supporting pipelines (data processing, evaluation, monitoring). In contrast to “Green IT” which focuses on infrastructure efficiency alone, Green AI represents a tight coupling between model behaviour and operational energy consumption: model architecture, parameter count, token throughput, batch size, hardware choice (CPU/GPU/accelerator), and deployment topology, all of which can substantially alter the energy usage per unit of useful work.

A key challenge for software sustainability is that AI workloads create new optimisation surfaces that do not exist (or are less dominant) in traditional software systems. For example, prompt-level strategies can change token counts and latency, while architectural techniques such as quantisation, pruning, and mixture-of-experts (MoE) alter compute intensity and memory access patterns.

Sustainable and Green AI requires an integrated view: measurement and reporting, benchmarking and comparability, model-level efficiency, system-level efficiency, and operational strategies (e.g., carbon-aware scheduling).

GreenCode aims to make best use of automated agentic AI processes, to optimise the energy consumption of software. To ensure it has a net benefit it must therefore use AI processes effectively and sustainably. As a result driving the development and use of Green AI is a significant internal consideration for the project as well as being a topic that can benefit all partners in their future work with AI systems.

Current State of the Art

AI/ML measurement, carbon tracking, and reporting

A growing ecosystem of academic and industrial tools supports training-time and inference-time footprinting for AI/ML pipelines. Commonly cited examples include Carbontracker[133], CodeCarbon, and the Experiment Impact Tracker [134], which integrate into Python-based workflows and can be used to estimate energy and carbon emissions for training runs and experiments.

An important state-of-the-art shift is from “reporting after the fact” toward operational energy observability. Tools and approaches increasingly expose energy/CO2e signals as time-series metrics that can be consumed in standard monitoring stacks (dashboards, alerting, tracing), enabling teams to detect regressions and tie AI workload changes to operational outcomes. In cloud-native environments this is reflected in container- and pod-level attribution approaches (e.g., Kubernetes-focused exporters), while host/process tooling continues to be used for calibration and deeper investigation. This trend matters because it aligns sustainability telemetry with the way production teams already manage reliability, performance, and cost.

However, AI measurement spans multiple "layers" (framework, process/container, infrastructure, and grid carbon factors). For this reason, Green AI tooling increasingly combines:

- **Energy estimation or measurement** (device/process/container attribution where possible), and
- **Carbon estimation**, usually by combining energy estimates with regional carbon-intensity signals.

Rather than repeating the energy tooling landscape here, we focus on what is distinct about Green AI: measurement boundaries vary across framework, process/container, infrastructure, and grid carbon factors, and "energy per run" are only comparable when hardware, batch size, and workload descriptors (e.g., token length and throughput) are reported consistently. Detailed coverage of measurement methods, calibration, cloud limitations, and tool capabilities (including RAPL-based approaches and container-level profilers) is provided in the Energy Measurement, Metrics and Attribution section.

Benchmarks and comparability for Green AI

Recent efforts to establish reproducible baselines for energy-aware AI and computing have emerged from the AI research community. While MLPerf has historically emphasised performance benchmarks for training and inference(AI workload efficiency, not software-engineering task benchmarks), newer initiatives extend benchmarking approaches to include energy and carbon impacts, and to treat energy as a first-class metric. Helmholtz AI's[135] sustainable AI benchmarking work is positioned around measuring both accuracy and energy consumption across model architectures and hardware platforms, emphasising transparency in reporting and fair comparison of optimisation strategies, particularly relevant for energy-aware AI workloads.

In practice, the hardest comparability problem is not selecting a metric but controlling the experimental description. For LLM-style workloads, small changes in prompt length, generation parameters, batching, caching, or concurrency can significantly change throughput and energy per unit of work. As a result, "green" benchmarking is increasingly framed as a protocol: a reproducible workload definition plus hardware and serving configuration disclosure, with enough detail to support re-runs and regression testing over time.

Despite this momentum, comparability remains uneven in everyday AI evaluation practice. For example, some model assessments report efficiency primarily in terms of inference time and VRAM consumption, explicitly not measuring energy. This creates a gap between "performance-centric" evaluation and "energy/carbon-aware" evaluation, and limits the ability to validate claims about sustainability improvements.

For GreenCode, this implies the need for a consistent evaluation protocol for AI-based tools, ideally including:

- workload definition (task type, dataset, prompt lengths / token budgets),
- hardware configuration (device, accelerator type, power limits),

- throughput/latency metrics (tokens/s, requests/s),
- and at least one sustainability metric (energy per inference/run, energy per token, carbon per run).

Model-level efficiency techniques

Model-level Green AI research focuses on reducing the compute and memory footprint needed to reach a given level of task quality. Techniques widely referenced include quantisation, pruning, and architectural choices such as mixture-of-experts (MoE), which activates only a subset of model components per inference (as discussed in the AI tooling for Code Optimisation section of this review).

In parallel, there is renewed interest in smaller and more specialised models (including small language models) for specific software engineering tasks such as code change summarisation, classification, and refinement, often supported by benchmarks that span multiple programming languages. However, there is still inconsistent inclusion of energy measurement in these studies, and at least one noted analysis of small language model energy consumption is now obsolete due to rapid model turnover.

A further practical direction is “small-first” model selection and routing: using smaller or specialised models for routine tasks (classification, summarisation, extraction, lint-like checks) and reserving larger models for genuinely hard cases. This aligns with the broader move toward modular pipelines and agentic workflows, where different steps can be assigned different model classes and energy budgets. For GreenCode this is particularly relevant because the optimisation pipeline itself must demonstrate net benefit, so model choice becomes part of the sustainability design space.

System-level efficiency techniques for training and inference

System-level techniques focus on improving the efficiency of executing a fixed model under realistic deployment constraints. In production, energy consumption is strongly influenced by utilisation, scheduling, and distribution decisions (e.g., batching, load balancing, or placement decisions in a cluster).

State-of-the-art system-level practice therefore concentrates on improving utilisation and reducing wasted work in serving: batching and request shaping to improve accelerator utilisation, careful placement and load balancing in clusters, and serving configurations that minimise unnecessary token generation. These levers are often more deployable than architectural model changes because they can be applied to a fixed model and iterated rapidly, but they also increase the importance of measurement boundaries and repeatable baselines when claiming sustainability gains.

This system-level perspective aligns with a broader measurement reality: different measurement tools expose different parts of the stack (CPU counters, GPU telemetry, container-level estimates, Windows process attribution), and cloud environments often restrict access to the lowest-level counters, leading to reliance on provider telemetry or estimation. This makes it especially important to define measurement boundaries when claiming energy improvements from serving strategies.

Operational Green AI and carbon-aware scheduling

Operational approaches aim to reduce *emissions* (not necessarily energy) by shifting workloads in time or location based on carbon-intensity signals. The Carbon Aware SDK is a representative approach: it supports carbon-intensity-aware workload scheduling for CI/CD and orchestration contexts, complementing direct energy measurement tools.

Operational approaches are most compelling when they integrate into existing automation points such as CI/CD, batch analytics, and periodic model evaluation/retraining. In these contexts, shifting workload execution based on carbon-intensity signals can reduce emissions with minimal impact on users, provided that SLAs and governance constraints are respected. This reinforces a broader SotA direction: sustainability controls are increasingly implemented as policy-driven orchestration decisions rather than as one-off manual interventions.

For industrial settings, even when tactics are known, adoption can be limited by perceived trade-offs, costs, and resistance, motivating stronger tooling support and clearer evidence on benefits.

AI and Machine Learning Workloads and Industrial "Green AI" Practice

The rapid growth of AI has spurred specialized research in energy-aware model design and deployment. Beyond prompt-level optimizations for large language models[86, 87], techniques such as quantization, pruning, and mixture-of-experts architectures provide substantial energy reductions[80, 86]. Training and serving workflows increasingly incorporate measurement and profiling (e.g. CarbonTracker[133], EnergyVis[132]) and are supported by emerging benchmark suites such as MLPerf-Energy[136] and domain initiatives such as Helmholtz AI[135], which together help establish more comparable efficiency baselines across training and inference tasks.

In industrial contexts, Green AI[137] is increasingly framed as a system and operations problem rather than a purely model-level choice. Tedla et al. (2024)[138] introduced EcoMLS, a self-adaptive architectural approach for designing energy-efficient ML-enabled systems that dynamically adjusts resource allocation and model complexity at runtime based on workload and energy constraints, reducing power consumption without compromising accuracy. De Martino et al. (2025)[139] examined green architectural tactics across 168 ML-enabled projects and found that practices such as optimised libraries and memory-efficient designs are already common due to their integration into popular ML frameworks, while explicitly energy-saving tactics remain rare. Reported barriers include organisational resistance, perceived performance trade-offs, and additional costs, reinforcing the need for education, incentives, and tooling that makes energy outcomes measurable, repeatable, and safe to adopt at scale.

Limitations and Gaps

The main barriers to making "Green AI" repeatable and governance-ready in practice are described in the table below. It highlights inconsistent measurement boundaries and metrics, limited comparability across tools and deployment contexts (especially in cloud environments), and strong context dependence of optimisation outcomes and trade-offs (quality, latency, cost, energy). It also captures ecosystem challenges such as rapid model turnover and limited lifecycle transparency,

reinforcing the need for closed-loop “measure > optimise > validate” practices with clear confidence bounds.

Limitation / Gap	Description	Why it matters	Implication for GreenCode
Energy not treated as a first-class evaluation metric	Many AI evaluations still report latency/throughput and memory (e.g., VRAM) but omit direct energy measurement or consistent energy estimates.	Sustainability improvements cannot be validated or compared reliably; efficiency claims remain ambiguous.	Define a GreenCode evaluation protocol that reports energy-per-run / energy-per-token alongside accuracy, latency, and cost.
Fragmented tooling and inconsistent measurement boundaries	AI energy/carbon tools operate at different layers (framework, process, container, cluster) and use different assumptions (device scope, idle allocation, etc.).	Results are not comparable across teams/environments; “same model” can produce different footprint claims.	Provide a unifying adapter/reporting layer that standardises boundaries and metadata (hardware, batch size, token length, utilisation).
Limited fidelity in managed/cloud environments	Access to low-level counters (CPU/GPU) may be restricted; users rely on coarse telemetry or estimation.	Undermines reproducibility and weakens attribution to model vs infrastructure effects.	Support “best available” measurement modes with explicit confidence levels and cloud-specific integration patterns.
Optimisation outcomes are context-dependent	Techniques (prompt strategies, batching, routing, quantisation) can reduce or increase energy depending on workload, model, and serving stack.	Best practices don’t transfer cleanly; risk of regressions and unintended energy increases.	Build closed-loop measure → optimise → validate workflows and energy regression tests in representative conditions.
Trade-offs across quality, latency, cost, and energy	Lower-energy configurations can degrade quality or increase latency; the “best” point depends on use-case constraints.	Single-objective optimisation is misleading; adoption fails without explicit trade-off handling.	Implement multi-objective scoring and policy-based selection (e.g., minimum quality threshold + energy minimisation).
Rapid model turnover makes findings stale	Model releases evolve quickly; older efficiency results may not generalise to current model families.	Evidence bases decay fast; practitioners rely on outdated conclusions.	Treat evaluations as continuously updated assets; automate re-benchmarking and versioned reporting.
Lifecycle opacity beyond operational energy	Training, data pipelines, and embodied impacts are hard to measure consistently and often omitted.	Optimising inference ignores large upstream footprints.	Support lifecycle reporting modes + “known unknowns” disclosure.
High energy cost of large AI models	Training and deploying LLMs can consume significant energy, potentially outweighing code efficiency gains.	Using large LLMs for minor optimisations may yield net-negative sustainability outcomes.	GreenCode should prioritise energy-efficient models and quantify net impact of AI-assisted optimisation.
Inconsistent LLM prompting results	Prompting suggesting energy-aware development can	Developers lack reliable methods to generate	GreenCode can test, refine, and formalise prompting

	sometimes reduce, but also increase, energy consumption.	efficient code with AI assistance.	strategies for energy-aware code generation.
Context dependence prompt optimisation	Energy savings from LLM prompting and ML system tactics vary by task, model, and hardware.	Some prompt modifications reduce energy use, others increase it.	GreenCode can benchmark across multiple contexts to provide reliable guidance.
Inconsistent AI optimisation outcomes	Prompting or green tactics for LLMs/ML can reduce or increase energy use depending on workload.	Hard to rely on AI-assisted optimisation in production.	GreenCode can test and formalise prompting heuristics across contexts.

Non-priorities

The non-priorities table below clarifies what GreenCode will not attempt to solve in the Green AI workstream in order to remain focused on practical, measurable impact. It excludes activities that would require universal or "perfect" accounting across all models and infrastructures, or that duplicate large-scale platform engineering in hyperscaler stacks, and instead keeps attention on interventions that can be evidenced, compared, and governed within defined measurement boundaries.

Area	Reason to De-prioritise	Implication
New AI hardware / chip-level optimisation	Requires specialist hardware engineering (accelerators, firmware, drivers) and long development cycles; outside a software engineering–led sustainability scope.	GreenCode should focus on software and system levers (model choice, serving efficiency, measurement, governance) rather than attempting hardware innovation.
Creating new global Green AI standards	Standards/benchmarks are already emerging; creating a new one risks duplication and slow uptake.	Align with existing standards and benchmarking efforts; contribute GreenCode protocols that map cleanly to them instead of inventing new ones.
Standalone "Green AI platform" disconnected from software engineering outcomes	Siloed footprint dashboards often do not change engineering behaviour unless integrated into SDLC/CI and decision workflows.	Prioritise integration into CI/CD, evaluation pipelines, and developer tooling so Green AI metrics directly drive actionable changes.
Carbon-aware scheduling without energy visibility	Carbon-intensity shifting can reduce emissions while masking absolute energy increases or performance/cost regressions.	Combine carbon-aware workload shifting with energy-per-work metrics and regression checks to avoid "greenwashing by scheduling."
"Maximum efficiency at any cost" optimisation	Aggressive energy reduction can undermine quality, latency, reliability, security, or maintainability—blocking industrial adoption.	Use policy-based, constraint-driven optimisation (e.g., minimum quality thresholds and latency/cost limits) with transparent evidence for recommendations.
Full lifecycle / embodied carbon accounting as a primary deliverable	Embodied carbon and supply-chain impacts are important but are data-intensive and methodologically complex,	Treat embodied impacts as contextual considerations; prioritise operational energy/carbon reductions and repeatable measurement first.

	often requiring external datasets and specialist LCA expertise.	
Training frontier/foundation models from scratch	Extremely compute intensive; distracts from GreenCode's measurable software/system levers.	Focus on evaluation/optimisation of existing models and practical deployment patterns.

Summary and Future Opportunities

The current state of Green AI shows strong progress in tooling and methods, but also clear gaps in comparability, operational integration, and industrial validation. GreenCode can contribute by treating sustainability metrics as first-class across the AI toolchain: (i) measurement and reporting for AI workflows, (ii) benchmark protocols that enable fair evaluation of optimisation claims, and (iii) validated optimisation levers spanning model choice (small/specialised vs large), architecture (e.g., MoE), and deployment strategies (load balancing, scheduling).

A distinctive opportunity is to operationalise Green AI as a closed loop: measure > optimise > validate > regress-test. The report already highlights that many sustainability tools work in silos and that energy regression detection is under-integrated in pipelines; GreenCode can unify these elements and make "green regression" testing and governance feasible in real industrial contexts.

Opportunities

GreenCode can advance Green AI in ways that deliver value both internally (reducing the cost and footprint of GreenCode's own analysis and optimisation pipeline) and externally (providing reusable methods others can adopt). It can focus on making energy/carbon measurement more repeatable and comparable, selecting and operating models more efficiently (right-sizing, quantisation, batching, routing), and embedding carbon/energy awareness into day-to-day deployment choices.

"Closing the loop" is also extremely important with respect to evidence capture and regression validation against quality, latency, and cost targets, so that Green AI becomes auditable and scalable, within GreenCode and across wider production environments.

Opportunity	Description	Example / implication	Application to GreenCode
Energy-aware benchmarking for AI code tools	Extend evaluation beyond time/VRAM to include energy/carbon metrics for AI-assisted software engineering tasks	Current evaluations often omit energy measurements	Build benchmarks that report energy-per-run/token and enable fair comparison
Unified AI footprint instrumentation	Consolidate AI measurement across framework,	Current tooling is fragmented across CodeCarbon/Scaphandre/Kepler/Windows tooling	Provide a unified measurement "adapter" layer and reporting format

	host/process, and orchestration layers		
Model routing and "small-first" optimisation	Prefer small/specialised models or sparse (MoE) approaches where adequate	Lower AI overhead relative to optimisation gains	Add a GreenCode policy layer to route tasks to the most efficient acceptable model
MoE efficiency validation harness	Evaluate MoE routing/load-balancing strategies for energy and throughput in realistic serving stacks	Reported energy reductions depend on system-level design	Integrate MoE energy profiling into deployment tests and validate claims
Carbon-aware scheduling integration	Use carbon-intensity signals to shift non-interactive workloads	Emissions reductions via timing/location shifts	Integrate Carbon Aware SDK-style scheduling hooks into CI/CD and batch workloads
Industrial adoption enablement	Address organisational barriers through evidence, governance, and low-friction tooling	Green tactics often under-adopted due to perceived trade-offs	Provide cost–benefit evidence, safe defaults, and governance-aware modes
Integration of green tactics into ML frameworks	Extend sustainability features via framework/plugin ecosystems.	PyTorch/TensorFlow integrations with memory-efficient ops.	Provide extensions/plugins for popular ML frameworks.

GreenCode: State of the Art Review

Sustainable Software Engineering Practices and Developer Impacts

Contributing Partner(s): TWT, FH Aachen, Digital Tactics

Background

The accelerating expansion of digital systems and transformation means that sustainable software engineering is becoming an increasingly important area of research and industrial practice. Poor quality and inefficient software leads to many direct and indirect environmental impacts including excess energy usage, infrastructure over-provisioning, accelerated generation of electronic waste and additional support and maintenance burdens. The growing prevalence of AI and large-scale cloud deployments has further intensified these challenges, as energy-hungry ML models and data centres dominate modern computing infrastructures.

Several organisations have started to address these concerns at a systemic level. The Green Software Foundation (GSF), established in 2021, coordinates cross-industry efforts by major technology firms to standardise sustainable software practices, offering resources such as design patterns and the Software Carbon Intensity (SCI) specification110. Similarly, the Software Sustainability Institute (SSI) provides guidance and training aimed at embedding sustainability into software research and practice111, while organisations such as the Green Web Foundation advocate for a carbon-aware internet infrastructure112.

These initiatives reflect a growing recognition of the developer's role in sustainability, highlighting that effective solutions require not only infrastructure-level optimisation but also improved awareness and practices at software design and development stages. Unfortunately, their efforts being largely socially focussed (standards, education) are limited by weight of numbers (size of the software industry vs. the reach of the educators, the GSF's own substantial membership accounts for <1% of the total industry) and additional tools and methods of engagement and outreach are needed to make an impact at scale.

Current State of the Art

Green IT and Architecture-Level Sustainability Practices

In a practitioner-focused study, Verdecchia et al. (2021)[140] provide guidance for green IT and software engineering by identifying a set of technical, social, and organisational solutions to reduce the energy footprint.

Based on interviews and focus groups with stakeholders from data centres and cloud providers, the authors categorise solutions across three timeframes:

- Short-term solutions, such as moving to the cloud (despite energy-measurement opacity), energy-aware software optimisations, and the adoption of green energy sources, which are readily available.

- Near-term solutions that focus on dynamic resource allocation, and workforce development, i.e. sustainable skills training.

- Long-term solutions that explore hardware breakthroughs, including the adoption of photonic computing[141].

The paper also discusses the potential of green testing methodologies, where, for instance, AI-assisted testing achieved double digit reductions in energy consumption in industrial testing environments.

Other studies have focused on how software architecture and design decisions can support sustainability goals in industry. Lago et al. (2020)[142], apply the Sustainability-Quality Assessment Framework (SAF) to guide architectural design in four industrial case studies, including energy provisioning via IoT (FlexIOT) and data-sharing systems. SAF enables practitioners to map quality attributes, such as adaptability, efficiency, and data quality, to sustainability concerns.

A key insight from these projects is the importance of social sustainability factors which is often overlooked in software practices. Moreover, adoption challenges remain, particularly the need for training in sustainability, lack of quality models, and integration between SAF decisions and traditional architectural designs.

Green Maintenance and Lifecycle Management

While green testing has received some attention100, sustainable software maintenance remains an underexplored area of the state of the art. Legacy systems often accumulate inefficiencies over time ("energy technical debt"), where outdated dependencies, redundant code, or unoptimised libraries gradually increase resource usage. Capilla et al. (2022)119 argue that energy consumption should be incorporated into technical debt management frameworks, allowing teams to prioritise refactoring tasks that have both functional and environmental benefits. Longitudinal studies of industrial systems show that periodic refactoring to adopt energy-efficient libraries, update compilers, and remove dead code can yield measurable energy savings120. However, formal methodologies and tools for measuring and mitigating "green debt" are still lacking, limiting the systematic adoption of sustainable maintenance practices.

AI in the SDLC for Sustainability

In the context of machine learning systems, Trinh et al. (2024)[143] conducted a systematic review on integrating AI within the software development life cycle, emphasising sustainability, particularly energy efficiency and electronic waste reduction. Their guidelines for designing sustainable software have direct implications for industrial practices, especially in companies adopting AI for automated testing and requirements engineering.

Recent advances in prompting techniques for LLMs highlight opportunities for energy-aware code generation102. However, results remain inconsistent, with some energy savings offset by increases in other scenarios. To move beyond these limitations, research is shifting towards embedding sustainability heuristics within development environments. Prototypes are emerging that provide real-time recommendations for energy-efficient data structures, library functions, or loop designs. By analogy with security linters, such systems could guide developers in producing code that balances maintainability, performance, and energy consumption. This represents a promising direction for

bridging the skills gap, reducing reliance on ad-hoc prompting strategies, and normalising sustainability as a measurable aspect of code quality.

Prompt Optimisation for Coding LLM's

In the context of large language models (LLMs), Cappendijik et al. (2025)[144] proposed a framework for profiling and optimising LLM behaviour during code generation, enabling developers to balance accuracy and energy consumption in production environments.

Five LLMs were tested on three Python problems from LeetCode and prompts were modified to request energy-optimised solutions, encourage library function use, or prefer for-loops over while-loops.

Their results show that prompt modifications sometimes reduce energy use, with the for-loop prompt showing the most frequent improvements. However, effects were inconsistent, and in some cases energy consumption increased significantly. The study concludes that prompting can influence energy efficiency but is highly context-dependent, requiring further research across more models, languages, and hardware.

Energy-Aware Coding Practices and Developer Tooling

In a report on the potential of green coding, Junger et al. (2024)[145] provide a review of green software engineering practices and tools, explicitly addressing their use in industrial settings. The authors identify a few industrial case studies, such as analysis of personal health record software, which measured energy consumption and proposed optimisation strategies. They also emphasise the skills gap among IT professionals in applying sustainable software engineering methods, findings that are also shared by Heldal et al. (2024)[146] and Lammert et al. (2023)[147].

The report highlights a range of industry-developed tools actively used to monitor and optimise energy consumption, including Green Metrics Tool, CodeCarbon, Scaphandre, and Kepler. While these tools exist the ecosystem remains fragmented as they are designed for discrete contexts such as ML training or container-level monitoring97. Unlike security or observability domains, where consolidated stacks and standards have emerged, there is no widely accepted "green developer toolkit."

This fragmentation makes adoption challenging, as teams must first be educated and then stitch together disparate tools and reconcile inconsistent metrics. Industrial case studies to build upon are limited and standardisation efforts such as the Green Software Foundation's SCI specification110 (while important) require stronger integration into mainstream development toolchains (e.g., IDEs, CI/CD systems, cloud observability platforms) to achieve widespread impact.

Developer Skills, Training, and Workflow Enablement

A consistent limitation across the literature is the insufficient integration of sustainability competencies into software engineering education and professional training. While frameworks exist

to guide sustainability skills, such as those proposed by Heldal et al. (2024)98, current industry practice often leaves responsibility for optimisation to infrastructure teams rather than embedding it within daily development tasks. The Software Engineering Body of Knowledge (SWEBOK), widely adopted as a reference for curricula, has yet to incorporate sustainability as a core area of competence113. This omission contributes to a lack of awareness among practitioners about the impact of design choices on energy consumption and carbon intensity.

Closing this skills gap will require updating training programmes, integrating sustainability into formal engineering standards, and developing IDE-level feedback mechanisms that allow developers to anticipate if not observe the energy impact of their coding decisions in real time.

Technical approaches to energy-aware testing and CI/CD (including examples such as EcoCI and pipeline-integrated energy regression detection) are covered in Energy-Aware Computing: Energy-Aware Testing and CI/CD.

Developer-Facing Benchmarks and Validation Practices

While case studies demonstrate encouraging sustainability improvement results in contexts such as IoT and cloud storage107,108, the generalisability of these findings remains limited. Software sustainability lacks shared benchmarks and testing suites that provide transparent and repeatable comparisons.

This creates difficulties for organisations seeking to measure trade-offs between performance and energy use across different architectures or frameworks. Establishing widely recognised benchmarks for sustainable software systems would allow companies to demonstrate improvements, justify investments, and align with emerging reporting requirements such as CSRD, all of which are currently difficult to achieve.

Developer Attitudes to Sustainability and Green Software Engineering

Developer interviews conducted as part of this SotA review indicate that, despite growing activity in special interest groups, energy efficiency is not yet a routine driver in day-to-day software development; when sustainability benefits occur they are typically incidental, arising from performance tuning, good engineering hygiene, and avoiding obvious waste, rather than being explicit targets with defined acceptance criteria.

Developers most often described “sustainable software” in pragmatic terms: using fewer resources and building systems that are maintainable and long-lived, with broader social dimensions mentioned far less frequently.

The main barriers are time pressure and weak incentives: optimisation is perceived as extra effort, customer willingness to pay is unclear, and sustainability work often lacks management sponsorship. As a result, practitioners emphasised that adoption depends on low-friction, workflow-native tooling that offers clear explanations and quantified impact reporting, delivered through familiar touchpoints such as IDEs and CI pipelines, and able to scale beyond individual developers to team-level practice. Attitudes to AI are cautiously positive—developers see value in automation and

guidance, but concerns about trust, compliance/IP risk, and whether AI tooling could cost more energy than it saves reinforce the need for governance controls and, in many contexts, local/on-prem deployment options.

These interview findings align with wider evidence that adoption is constrained by a persistent skills and awareness gap.

Surveys [146, 147] consistently report that most developers lack formal training in energy-aware programming, and while industry initiatives (e.g., the Green Software Foundation's patterns and guidance, the Software Sustainability Institute's training activities, and Green Web Foundation advocacy) provide important knowledge transfer, uptake remains uneven.

The gap is amplified by the field's historic emphasis on performance-first development, with the energy–performance trade-off often underexamined in engineering teams, and by tooling limitations: although many tools and methods exist, they are frequently tailored to specific research contexts and are not always sufficiently scalable, customisable, or seamlessly integrable into existing enterprise development workflows.

Limitations and Gaps

The Limitations and Gaps table below summarises why sustainable software engineering remains difficult to adopt consistently in practice, despite a growing body of guidance and tooling. It highlights a combined deficit of skills and incentives (limited training/awareness), fragmented tooling and metrics (hard to integrate into CI/CD and day-to-day workflows), and a lack of repeatable industrial validation and benchmarks to justify investment and compare improvements. It also captures longer-horizon challenges that matter for GreenCode's mission, green maintenance/"energy debt" in legacy estates, persistent perceptions of energy–performance trade-offs, organisational resistance, weak SDLC-wide integration (e.g., Definition-of-Done), and limited mechanisms for green regression and post-deploy drift control, all pointing to the need for evidence-backed, workflow-integrated approaches.

Limitation/Gap	Description	Example/Implication	Application To GreenCode
Lack of awareness and training	Many developers lack knowledge of sustainability principles; SWEBOK does not yet include sustainability competencies.	Developers unaware of energy costs of design choices, leading to inefficient defaults.	Embed sustainability in training materials and consider the provision of IDE-level feedback to developers.
Tooling fragmentation	Tools such as CodeCarbon, Kepler, and Scaphandre are not integrated into mainstream developer workflows.	Hard for teams to reconcile inconsistent metrics or add sustainability to CI/CD pipelines.	Develop or integrate a unified toolkit into the GreenCode platform.

Limited industrial validation	Case studies exist (IoT, tourism, cloud storage) but results lack generalisability across domains.	No widely adopted equivalent of MLPerf for software sustainability benchmarks.	GreenCode can serve as an industrial testbed and contribute benchmark data.
Lack of green maintenance strategies	Energy technical debt accumulates over time in legacy systems.	Inefficient libraries and dependencies persist long after initial design.	Position GreenCode as a framework to assess and manage "green debt" over time.
Performance trade-offs	Developers and organisations often perceive energy efficiency as conflicting with performance.	Teams may avoid energy-saving tactics due to fear of latency or throughput degradation.	GreenCode can provide evidence and metrics to demonstrate "win-win" cases where both performance and energy improve.
Organisational and cultural barriers	Resistance to change and concerns over costs limit adoption of sustainable practices.	Green architectural tactics are rarely applied in ML-enabled systems due to perceived overheads.	GreenCode can highlight cost-benefit analyses and provide incentives for adoption.
Neglect of social sustainability	Human and organisational sustainability aspects are overlooked compared to technical factors.	SAF case studies show adaptability and social dimensions are often ignored.	GreenCode could extend its framework to include awareness of social/organisational impacts alongside technical ones.
SDLC-wide sustainability integration gaps	Practices often focus on runtime fixes, not end-to-end SDLC governance.	Sustainability not present in DoD/acceptance criteria; upstream choices persist.	Add lifecycle evidence flows and "definition of done" hooks.
Green regression + post-deploy drift control	Energy impact can drift after release due to config, traffic, dependencies.	Efficiency regressions go unnoticed for months.	Integrate regression monitoring + release gates + rollback triggers.
Lack of lifecycle coverage	Most tools focus on runtime, not full SDLC (design → testing → maintenance).	Misses upstream/downstream energy costs.	Extend GreenCode analysis to lifecycle metrics and refactoring prioritisation.
Green testing underdeveloped	CI/CD pipelines rarely include energy profiling or regression testing against sustainability metrics.	Energy-efficient design is undone by "energy-blind" testing and deployment.	Extend GreenCode to include automated green testing modules.

Non-Priorities

The Non-priorities table below clarifies what GreenCode will not attempt within the sustainable software engineering workstream in order to stay focused on actionable outcomes. It excludes efforts that would require broad cultural transformation without tooling support, attempts to create universal "one-size-fits-all" sustainability rules across all organisations and workloads, or duplicating established compliance frameworks and generic training programmes.

Instead, GreenCode prioritises interventions that can be operationalised and evidenced, embedding sustainability signals into normal engineering workflows with measurable, repeatable improvements, rather than treating sustainability primarily as a standalone awareness or policy exercise.

Area	Reason to De-prioritise	Implication
Highly domain-specific optimisation or niche programming languages	Narrow impact compared to cross-domain practices.	GreenCode should prioritise Python, C/C++, Java, and Rust before covering niche languages.
Social sustainability dimensions	Important, but less directly measurable in early technical prototypes.	GreenCode should acknowledge but focus initially on technical sustainability metrics.
Organisation-wide language rewrites as the default lever	In most organisations language is constrained by skills/legacy; full rewrites are rarely feasible.	Provide guidance and targeted migration only where justified (hotspots, lifecycle events).
Pure awareness campaigns without workflow/tooling integration	Training alone rarely changes practice without "in the flow" tooling/metrics and incentives.	Prioritise workflow-integrated interventions (CI gates, IDE hints, dashboards, DoD).

Summary and Future Opportunities

Sustainable software engineering is increasingly recognised as important, but it is not yet embedded as a routine engineering discipline. In practice, sustainability outcomes are often incidental, arising from performance work and general "good engineering", because teams lack clear incentives, training, comparable metrics, and workflow-integrated tooling to make energy and carbon a first-class acceptance criterion.

Progress depends less on new principles and more on operationalisation: consistent measurement boundaries, actionable guidance surfaced in familiar touchpoints (IDE/CI), and governance-ready evidence that supports trade-offs across quality, performance, cost, and sustainability.

For GreenCode, this reinforces the need to close the loop from measurement to optimisation to regression validation, enabling organisations to treat "energy/efficiency debt" as a managed part of software lifecycle and legacy modernisation rather than an optional aspiration.

Opportunities

GreenCode can most effectively accelerate adoption of sustainable software engineering by improving its own optimisation workflow and by producing reusable practices for the wider ecosystem. Focussing on making sustainability easy to act on: integrating measurement and guidance into CI/CD and developer tooling, translating principles into concrete refactoring and architectural patterns, and enabling green regression (detecting energy/performance drift) alongside functional and security checks are key opportunities. Strengthening the evidence base through shared benchmarks, impact reporting that supports business trade-offs, and repeatable governance-ready artefacts (e.g., confidence grading, audit trails) so sustainability becomes a managed engineering outcome rather than an aspirational guideline would be valuable.

Opportunity	Description	Example/Implication	Application To GreenCode
Unified sustainability toolkit	Integrating measurement, optimisation, and	Equivalent to an "observability stack" for energy efficiency.	GreenCode can position itself as this consolidated toolkit.

	reporting tools into a single framework.		
Green testing in CI/CD	Embedding energy profiling into testing pipelines.	Energy regression tests run alongside unit tests.	Add a GreenCode CI/CD plugin.
IDE-level sustainability feedback	Real-time developer guidance on energy costs of code patterns.	Similar to linters flagging anti-patterns; IDE highlights inefficient loops or data structures.	Surface sustainability signals through existing developer touchpoints (e.g., CI reports, code review annotations, dashboards), with optional IDE integration where it demonstrably reduces friction and improves adoption.
Green maintenance frameworks	Incorporating energy into technical debt management.	Prioritising refactoring tasks with environmental impact.	Extend GreenCode analysis to track "green debt" across versions.
Developer skills and incentives at scale	Make sustainability "doable" via training + nudges + team incentives, not just awareness.	Moves practice beyond voluntary individual effort.	Bundle playbooks, team-level dashboards, and "definition of done" hooks.

GreenCode: State of the Art Review

Energy-Aware Computing

Contributing Partner(s): ISEP, IrRADIRE, Trier University of Applied Sciences, Digital Tactics

Background

Energy-aware computing has evolved from low-power design in embedded systems and mobile devices toward large-scale applications in datacentres, cloud, and AI workloads. Early approaches emphasized hardware-level optimization, such as dynamic voltage and frequency scaling (DVFS) and power gating[37], but recent research increasingly highlights the software layer as a critical lever for reducing energy demand[249]. Distinguishing between "green IT" (infrastructure-level efficiency), "green software engineering" (development practices and design patterns), and "energy-aware computing" (runtime, compiler, and code-level interventions) is essential to define the scope of this field [140].

Growing regulatory and industry frameworks have accelerated the focus on energy-aware computing. Initiatives such as the EU Corporate Sustainability Reporting Directive (CSRD)[218] and ISO/IEC standards for sustainable IT[221] require organizations to report and optimize the energy footprint of their digital services. Green software practices, such as energy-aware computing aim to counter this by making software part of the solution rather than the problem. Fundamentally, an energy- or carbon-aware system does more when greener energy sources can be leveraged, and less when the energy or carbon emissions are higher.[148] This means designing software and systems to monitor energy conditions (such as grid carbon intensity, workload efficiency, or other similar metrics) and adjust their behaviour accordingly, for example delaying non-urgent batch jobs or choosing a cloud regions according to energy mix and green energy availability. By incorporating such logic, developers can significantly reduce the emissions associated with their systems.[149]

Equally as important is promoting considerations on energy-awareness during software and system development and design processes. This mainly involves dynamically adapting computational operations and system load based on real-time, estimated or forecasted energy characteristics based on power profiling, workload classification, or other system-level energy modelling methods.

This area of research is rapidly evolving as related approaches have shown significant promise for mitigating the energy intensity (and thereby reducing emissions and operational energy usage) in datacentres, cloud computing, and large-scale compute environments.[150] [151]

Further, it is necessary to not only extend traditional performance-driven computing with energy considerations, but to further connect strategies for energy optimization across all system layers, meaning that it should not be addressed in isolation within individual components but instead through integrated, system-wide coordination in alignment with broader system-level goals.[152]

Current State of the Art

At the runtime and operating system level, energy-aware computing techniques aim to reduce the energy consumption of software systems without requiring changes to application logic, by influencing how workloads are scheduled, executed, and placed on underlying hardware. A number of mature mechanisms already exist to support such energy-aware execution. Modern Linux kernels,

for example, include energy-aware scheduling (EAS) to optimise workload placement across heterogeneous cores [153].

At the infrastructure layer, container orchestration platforms such as Kubernetes increasingly expose cgroups and quality-of-service (QoS) controls that allow energy usage constraints to be enforced alongside CPU and memory limits. Comparable capabilities are also available on other major operating systems, including the Windows Energy Estimation Engine and macOS Energy Impact metrics, which provide developers and operators with visibility into application-level energy costs [154].

Together, these mechanisms demonstrate that energy-aware computing is technically feasible across platforms, even if such capabilities are not yet systematically exploited by software systems or development workflows.

Beyond programming language choice, concrete coding practices also exert a measurable influence on software energy consumption. Empirical studies have shown that patterns such as busy-wait loops, excessive polling, and unnecessary recursion waste CPU cycles and increase energy demand[130]. Likewise, memory-inefficient data structure choices amplify energy costs associated with allocation and garbage collection. Conversely, techniques such as caching frequently accessed values, reducing unnecessary object creation, and adopting lazy evaluation where appropriate have been demonstrated to reduce computational overhead [155].

These findings highlight that software-level design and implementation decisions play a significant role in shaping runtime energy behaviour, particularly when considered in conjunction with underlying runtime and operating-system mechanisms. However, these practices are not consistently taught or reinforced in mainstream curricula, and most IDEs lack feedback mechanisms to guide developers toward energy-aware design.

Energy-Aware IT Infrastructure Optimization and Reconstruction

Datacentres are significant contributors to high energy consumption; however, a root cause of energy inefficiency lies in the software and IT infrastructure design choices. For example, green computing patterns highlighted that inefficient resource utilization, poor workload distribution, and suboptimal placement of services contribute to unnecessary energy consumption [155]. Hence, optimizing IT infrastructure is crucial to reducing environmental impact while maintaining system performance and reliability.

In the following, IT infrastructure optimization is viewed as the sustainable reconfiguration of IT infrastructure. Sustainable reconfiguration involves the adaptation of the current IT infrastructure to improve its sustainability, using historical data and future trend forecasts. This process consists of two main parts:

1. Sustainable reconfiguration actions: These are the actions that contribute to improving the sustainability of the IT infrastructure
2. Optimization approaches: These are methods that use historical data and future trends to select and choose reconfiguration actions through deep reinforcement learning and genetic algorithms for example.

GreenCode: State of the Art Review

This section explores both parts, starting with sustainable reconfiguration actions, such as VM container relocation, followed by examining generic reconfiguration and optimization approaches for IT infrastructures across on-premises, cloud, and hybrid deployments, with and without a specific focus on sustainability.

Sustainable Reconfiguration

Sustainable reconfiguration actions aim to increase the overall sustainability of the IT infrastructure, primarily by decreasing energy consumption and reducing the number of required physical resources (resulting in less embodied carbon). These actions focus on software-level and logical changes and do not involve physical hardware upgrades, such as replacing cooling equipment with newer versions or upgrading GPUs to modern ones, even though these upgrades can yield substantial energy savings. However, hardware upgrades are only considered if workload migration to an existing newer hardware configuration is possible.

Based primarily on Bharany[Error! Bookmark not defined.], but also considering other sources, three areas for the reconfiguration of IT infrastructures can be identified:

1. Resource Power Management;
2. Server consolidation, workload placement, and migration, and;
3. Thermal management (heating & cooling)

Firstly, resource power management which deals with the power management of the physical resources in the IT Infrastructure. This approach is necessary as not all components are used equally. Studies have shown that cloud servers in datacentres are active only 10% to 30% of the time, while they remain idle around 70% of the time[Error! Bookmark not defined.]. To address this inefficiency, underutilised resources can be either completely powered down or their energy consumption can be reduced via Dynamic Frequency Scaling (DVFS). DVFS works by dynamically adjusting the voltage and clock frequency of processors, thereby reducing power consumption.

The second area concerns physical machine consolidation, workload placement, and migration.

Physical server consolidation addresses the overprovisioning and underutilization of physical hardware by reducing the number of active physical servers through virtualization techniques such as VMs and containers. This approach enables multiple applications to share the same physical infrastructure, overcoming the inefficient one-to-one application-to-server model. Such consolidation decreases direct energy consumption and reduces associated cooling requirements, physical space needs, and embodied emissions through reduced hardware procurement [156,157,158].

To realize the benefits of physical server consolidation, a strategic initial workload placement is required to avoid hardware underutilization from the start.

However, after some time, inefficiencies can sneak into the IT Infrastructure, which requires the migration of workloads to different machines. The goal of the migration is to restore higher utilization rates, assign workloads to more energy-efficient hardware suited for their specific tasks, and optionally enable the shutting down of over-provisioned servers that are no longer needed. These techniques must balance migration costs against energy savings, as the live migration consumes energy and creates temporary performance impacts.

The third and final reconfiguration area addresses thermal management (heating & cooling), because cooling systems can account for 30-40 % of a facility's electricity use. Against this, the EU requires that all new data-centre builds from 2026 achieve a Power Usage Effectiveness (PUE) ≤ 1.2, while the current European average is about 1.6 and many legacy sites still exceed 2.0[159][160]. For example, placing additional workloads on an already highly utilised server may avoid powering up a new machine, but could potentially generate thermal hotspots requiring increased cooling energy.

Reconfiguration Approaches

Our focus here regards improving an existing IT infrastructure, we do not consider the initial placement or scheduling of workloads, as such in the following sections we will cover general reconfiguration approaches, post initial cloud deployment optimisation, and energy-aware optimisation approaches.

An early general reconfiguration approach to IT infrastructures was proposed by Frey et al.[161]. Their approach aims to migrate an existing software system to a cloud platform. This migration process aims to find a cloud deployment option (CDO) that balances response time, costs, and SLA violations. A CDO comprises a target cloud environment, suitable VM instance types, and runtime configuration rules for dynamic resource scaling. To find the optimal CDO for a given software system architecture, they proposed a genetic algorithm that explores the CDO design space and identifies a Pareto-optimal solution.

Khan[162] proposes an approach for post-deployment optimization of multi-cloud environments, focusing on improving performance, availability, and resource utilization while minimizing costs. Their optimization approach consists of four main tasks: data gathering, future load prediction, optimization, and infrastructure adaptation.

Current data is collected from the infrastructure, including resource usage metrics such as CPU load, RAM usage, inbound and outbound traffic, costs, and infrastructure changes.

Based on this data, a prediction component utilizing Facebook Prophet[163] forecasts the future CPU load of the virtual machines (VMs) to identify those that can be shut down. Using the current infrastructure state and these predictions, a planning component proposes actions to achieve a new optimized state. This component employs hierarchical task networks for the optimization process and generates the following actions: optimizing the network via caching, optimizing storage by compressing data, relocating data to cold storage, or cleaning up compressed data. For VMs, possible actions include turning them on or off based on predicted load, relocating them to a different region, reducing their replica count, or deallocating them completely. Finally, an execution component implements these generated plans to adapt the current IT infrastructure.

The following section presents energy-aware optimization methods originally developed for datacentre management that may be transferable to IT infrastructures. We explore two main categories:

1. Consolidation strategies involving migration and load balancing, and shortly
2. Holistic datacentre optimization using reinforcement learning

To address the inefficient utilization of physical resources in cloud environments, which results in high energy consumption and low quality of service, Yao et al.[Error! Bookmark not defined.] propose an energy-efficient load-balancing strategy based on virtual machine consolidation (LBVMC). Their algorithm

considers multi-dimensional resource utilization, including CPU, memory, and bandwidth, and consists of the following three parts. First, based on current and predicted load, physical machines are classified into two categories: source Physical Machines (PMs), from which VMs should be moved to avoid overloading, and destination PMs, to which VMs can be migrated. In the second part, the migratable VMs are selected using a resource-weighted function that accounts for the multi-dimensional load.

Finally, the VM placement algorithm evaluates resource fitness, which refers to how well the resource profile of the VM matches the available capacity of a physical machine. This step also considers the similarity of the VM's demands to those of the VMs already on the target PM, which helps to avoid future overloading and performance degradation caused by resource contention.

In line with the work of Yao et al.[Error! Bookmark not defined.], Rezakhani et al.[Error! Bookmark not defined.] propose a similar consolidation algorithm. The goal of their approach is to migrate VMs from overloaded hosts to reduce their energy consumption and from underloaded hosts to allow them to be shut down, thereby saving energy overall. Similar to the work of Yao et al., their algorithm consists of several stages. Initially, an artificial neural network classifies each host as normal, overloaded, or underloaded. Based on this classification, a reinforcement learning agent selects the VMs to be migrated from the overloaded physical machines. Finally, the VMs selected by the RL agent, along with all VMs on the underloaded physical hosts, are moved to normal hosts, which are selected via a modified version of the best-fit-decreasing algorithm.

Both Yao et al. and Rezakhani et al. assume that shutting down or powering up a physical host is an instantaneous process. Furthermore, they implicitly presuppose that "on" and "off" are the only possible states for a host and that the transition between them is unaffected by Power Mode Transition (PMT). To address this, they propose a Two-Level Heuristic-based Virtual Machine Consolidation (TLHVMC) algorithm. The algorithm consists of Time Window-based Centralised Power Model Control (TWCC) policies to control the power modes of each host, while the second component, the Greedy Strategy-based hybrid heuristic placement (GSP) algorithm, is used to improve VM placement. The TWCC component suspends hosts that have just been emptied and those that remain idle after the consolidation process is complete.

Additionally, it wakes up sleeping physical machines based on a moving average to avoid Service Level Agreement (SLA) violations. Furthermore, the algorithm detects overloaded physical hosts via a threshold and selects migratable VMs based on the shortest migration time. The GSP algorithm then determines the placement of these VMs. Therefore, the GSP algorithm first sorts VMs by their CPU needs and categorizes host servers as normal, underutilized, or idle. The algorithm then attempts to place the movable VMs on a normal host and only falls back to underutilized or idle hosts when no other option is feasible.

In a recent review, Kahil et al.[164] examined how reinforcement learning (RL) approaches can be utilized to optimize the energy efficiency of datacentres in various aspects. They showed that RL can be utilized to optimize cooling systems, including ICT processes such as task scheduling, resource allocation, and VM placement and consolidation, as well as datacentre network traffic control and both cooling and ICT systems. They thereby identified that most current studies rely on simulated environments and lack real-time validation. Additionally, more robust multi-scale metrics are needed to reflect the energy improvement accurately. Finally, they identified that multi-agent deep reinforcement learning

algorithms are a promising direction for further optimizing the increasingly complex cooling and ICT systems.

Infrastructure Reconstruction via IaC

Infrastructure as Code (IaC) is a practice that automates the provisioning and management of IT infrastructure using code instead of manual configuration steps. Modern application environments consist of various components, such as operating systems, databases, and storage systems, that require regular setup, updates, and maintenance, especially during development, testing, and production phases. Manual management of this infrastructure is not only time-consuming but also prone to errors, especially in large system landscapes. IaC enables the description of the entire required infrastructure, including computing power, storage, and network configuration, in executable code. This code can be flexibly adapted, scaled, duplicated, deleted, and versioned, making infrastructure reproducible, scalable, and consistently manageable. This significantly enhances efficiency and reliability, especially in dynamic or cloud-based environments. Key concepts of IaC include the utilization of cloud technologies such as virtualization and software-defined resource handling [165].

Current Infrastructure as Code (IaC) implementations differ significantly in their approach to defining and managing infrastructure. Imperative IaC solutions, such as Ansible[166] (using YAML and Python modules) or Chef[167] (using a Ruby-based DSL), provide a step-by-step description of how the desired infrastructure should be created. Developers have complete control over the process, as the order of steps is explicitly defined. While this approach offers flexibility, it requires a detailed understanding of the processes and is more prone to errors in complex scenarios. In contrast, declarative IaC paradigms, as implemented in tools like Puppet[168] (using a proprietary declarative DSL), Terraform[169] (using HashiCorp Configuration Language), AWS CloudFormation[170] (using JSON or YAML), or Azure Resource Manager[171] (using JSON), allow developers to simply specify the desired state of the infrastructure. For instance, they can define which resources should be available and with which properties. The IaC tool then automatically calculates and executes the necessary steps to achieve this state. This method is often more robust, easier to maintain, and particularly well-suited for repeatable deployments. Recent trends in IaC extend beyond purely declarative approaches. Tools like Pulumi[172], AWS Cloud Development Kit[173] (CDK), and CDK for Terraform[174] enable developers to define the desired infrastructure state using general programming languages such as Python, TypeScript, or Go.

While Cloud Development Kit approaches ultimately generate declarative code (such as CloudFormation or Terraform HCL), which may reach their limits, Pulumi executes the operations directly. This enables a closer integration of code execution and provisioning logic, allowing downstream configuration values of one resource to be used to configure others. Consequently, Pulumi results in a more integrated and dynamic user experience that can effectively manage more complex dependencies [175, 176].

A significant challenge in using Infrastructure as Code (IaC) is managing the increasing heterogeneity in deployment automation for modern, distributed applications. These applications often consist of multiple components deployed in diverse environments and utilizing various deployment technologies. Consequently, managing this complex infrastructure becomes challenging, as it may not be feasible to use a single IaC tool. For instance, virtual machines in private clouds are provisioned using Terraform, while middleware components are installed using Puppet, and containerized services

are orchestrated via Kubernetes. The multitude of tools employed hinders a holistic view of the application, making subsequent management tasks, such as backup, scaling, and updates, significantly more difficult, especially when these processes span tool boundaries. To address this issue, Harzenetter et al.[177] propose an integrated management system that automates the creation of an instance model. This model encapsulates the current state of the running application in a standardized format using the TOSCA standard (Topology and Orchestration Specification for Cloud Applications).

The information for this model is extracted directly from the APIs of the deployment technologies employed. The instance model serves as the foundation for the automatic generation of executable management workflows, enabling coordinated management of the application regardless of the tool used for initial deployment. Notably, Terraform, a declarative infrastructure provisioning technology, is particularly well-suited for initial virtual machine provisioning. However, Terraform lacks the capability to store information about the software components installed on these virtual machines. To gain a comprehensive understanding of the application, Terraform integrates additional information from tools like Puppet or specialized plugins. Moreover, during state-changing operations, changes that impact a component's state, the system ensures that the relevant deployment technology is informed about these changes. This prevents unintended overwriting of target states enforced by the tools[Error! Bookmark not defined.].

Programming Language and Runtime Impacts on Energy Efficiency

Empirical studies consistently show that programming language choice significantly affects energy consumption. Pereira et al. (2017) [249] benchmarked 27 widely used languages across a set of computational problems and observed up to a 79× difference in energy consumption between the least and most efficient implementations.

Research comparing compiled (e.g., C, Rust) and managed languages (e.g., Java, C#) highlights orders-of-magnitude differences in performance-per-watt[243, 249]. While the Real-Time Specification for Java (RTSJ) aimed to support energy-aware execution, mainstream virtual machines do not implement it, leaving only niche offerings like JamaicaVM[178].

Emerging languages such as Rust and Go, designed with memory safety and performance trade-offs, are increasingly studied for energy-aware applications. At the runtime level, advances in garbage collection tuning, concurrency models, and memory-efficient data structures remain central to optimization[128].

Rust and Energy Efficiency

Programming language choice has a measurable effect on energy consumption. Comparative studies of compiled versus managed languages consistently show significant differences, with compiled languages such as C and Rust generally outperforming interpreted or virtual machine-based environments[249]. Rust, in particular, offers memory safety guarantees without relying on garbage collection, resulting in lower runtime overhead and improved performance-per-watt in many contexts. Its design enables fine-grained control over resource management, making it attractive for energy-sensitive applications such as embedded systems and high-performance computing. Although

Rust adoption in enterprise remains limited compared to Java or Python, empirical results highlight its potential as a language aligned with sustainability goals.

Garbage Collection and Runtime Impacts

Managed languages like Java and C# rely on garbage collection (GC) for memory management, but GC strategies significantly influence energy efficiency. Frequent or poorly tuned collections can cause spikes in CPU utilisation, leading to higher power draw [179]. Research into energy-aware garbage collection demonstrates that adaptive GC policies, which balance throughput with reduced collection frequency, can cut energy usage without major performance penalties. Despite these advances, mainstream runtimes rarely expose GC-energy trade-offs directly to developers, limiting practical adoption. This remains a key runtime-level challenge for energy-aware computing.

Energy-Aware Testing and CI/CD

The integration of energy awareness into software testing and continuous integration and deployment (CI/CD) pipelines is an emerging research direction at the intersection of DevOps and sustainable software engineering. Conventional CI/CD workflows primarily prioritise functional correctness, reliability, and performance, while sustainability considerations are rarely treated as first-class quality attributes. Recent research nonetheless demonstrates the feasibility of embedding energy measurement and profiling into automated CI workflows, enabling teams to monitor energy consumption during build and test phases and to detect energy-efficiency regressions in a manner analogous to performance regression testing[124].

Early tooling provides indicative evidence of practical pathways. For example, EcoCI[180] demonstrates how energy-aware CI pipelines can produce developer-facing feedback that supports iterative optimisation and prevents regressions. In parallel, AI-assisted testing approaches have shown promise in limited industrial and experimental settings, reducing energy consumption through strategies such as optimised test selection, prioritisation, and execution ordering[181]. These approaches align well with existing testing goals, maintaining coverage and confidence while reducing unnecessary computation, but remain far from standard practice.

Despite this progress, the evidence base remains sparse, tooling maturity is low, and adoption in production DevOps environments is limited. Open challenges include agreement on standardised and comparable metrics, reproducible measurement under variable CI infrastructure conditions, scalability across heterogeneous workloads and platforms, and low-friction integration into mainstream DevOps ecosystems. As a result, energy-aware CI/CD is still under-explored relative to its potential impact and represents a clear opportunity for GreenCode-aligned methods that combine reliable measurement, automation, and actionable developer guidance.

Hardware-Software Co-Design

Recent work demonstrates that energy efficiency is maximized when hardware and software are co-optimized. Techniques such as heterogeneous scheduling across CPUs, GPUs, and TPUs, near-memory computing, and runtime-aware placement of workloads are widely adopted in research prototypes[85, 86]. Compiler-level optimizations for ARM[182] and RISC-V[183], as well as hardware counter–driven adaptive scheduling, enable more granular energy savings than traditional static policies153, .

Limitations and Gaps

Energy-aware computing remains difficult to operationalise at the software layer, even where measurement is possible. As the table highlights, most OSs and runtimes still treat energy as an implicit side effect rather than a managed resource, with few control surfaces exposed to developers (e.g., opaque GC/JIT/scheduling behaviours) and limited mechanisms to enforce energy budgets at runtime.

There is also weak portability of energy behaviour across platforms, dominance of static/offline tuning over adaptive optimisation, and a lack of well-defined runtime trade-off models between energy, latency, and throughput. Finally, the table points to foundational ecosystem gaps, limited compiler support for energy objectives, noisy/uncertain measurements, fragmented telemetry across layers, and the under-utilisation of known language/runtime energy differences in real engineering decisions.

Limitation / Gap	Description	Example / Implication	Application to GreenCode
Energy not a first-class runtime resource	Most operating systems and runtimes treat energy as an indirect side-effect rather than a managed resource.	Schedulers optimise for latency or throughput, not energy.	GreenCode can promote energy as a controllable runtime objective via policy and recommendations.
Limited control surfaces for software	Developers and applications have little direct influence over energy-related runtime decisions. GC, JIT, and scheduling policies are opaque or coarse-grained. In practice, resource monitoring, budgeting, and enforcement mechanisms are rarely exposed at the programming-language level or integrated into mainstream development environments, meaning that energy and resource control is typically applied post hoc through profiling and operational tooling rather than being expressible directly within application code.	GC, JIT, and scheduling policies are opaque or coarse-grained.	GreenCode can surface tunable runtime and OS-level parameters.
Poor portability of energy behaviour	Energy efficiency does not transfer predictably across platforms.	Optimisations valid on x86 may fail on ARM or edge devices.	GreenCode can build comparative platform-aware energy models.

Static optimisation dominates	Many energy-aware techniques rely on static configuration or offline tuning.	Systems cannot adapt to workload or context changes.	GreenCode can support adaptive, runtime-aware optimisation loops.
Lack of energy-aware enforcement mechanisms	Systems can observe energy but rarely enforce limits.	Applications exceed energy budgets without consequence.	GreenCode can integrate enforcement strategies via orchestration and runtime policies.
Unclear trade-off modelling at runtime	Energy–performance trade-offs are poorly formalised at execution time.	Decisions default to performance even when marginal gains are costly.	GreenCode can provide quantified runtime trade-off guidance.
Limited integration of energy into compilers	Compilers rarely optimise explicitly for energy.	Generated binaries prioritise speed or size.	GreenCode can feed energy models into compilation and optimisation passes.
Runtime optimisation under uncertainty	Energy measurements are noisy, delayed, or approximate.	Optimisation decisions may be unstable or incorrect.	GreenCode can use probabilistic and confidence-aware adaptation strategies.
Fragmented energy signals across layers	Energy signals are exposed in inconsistent ways across OS, hypervisors, containers, and runtimes.	Hard to correlate energy impact with software decisions across the stack.	GreenCode can unify cross-layer telemetry and attribution models.
Programming language impacts poorly considered	Large empirical differences exist between languages in energy use, but rarely influence developer decisions.	Developers may select Python for speed of development, despite far higher energy costs compared to C or Rust.	Provide guidance or automated suggestions for language and runtime trade-offs.
Programming language impacts neglected	Language/runtime choice significantly affects energy use but is rarely accounted for.	Python dominates despite higher energy costs; Java RTSJ unsupported in mainstream JVMs, with only niche implementations like JamaicaVM available; profiling tools not production-ready.	GreenCode can provide comparative guidance and build abstractions that normalise measurements across languages/runtimes.

Non-Priorities

GreenCode will not focus (i) hardware-only benchmarks that are informative but not directly actionable for developers, (ii) building new low-level power profilers from scratch where mature signals already exist and the real gap is integration and standardised "measurement modes," and (iii) attempting large-scale consumer device profiling, where device heterogeneity makes results hard to reproduce and compare. The intent is to leverage existing measurement foundations and invest instead in repeatable methods and pipelines that GreenCode can validate and operationalise across server, edge, and cloud contexts.

Area	Reason to De-prioritise	Implication
Hardware-centric innovations (e.g., photonic computing, chip DVFS)	These advances are outside GreenCode's remit and depend on vendors.	Focus should remain on software-level leverage.
Niche runtimes (e.g., JamaicaVM)	Minimal industry adoption makes them impractical as a foundation.	Concentrate on mainstream Java/Python/Rust/Go ecosystems.
End-user device profiling (PCs, phones)	Highly heterogeneous, hard to standardise across billions of devices.	Prioritise server, edge, and cloud workloads first.
Generic carbon policy analysis	Regulation is context but not the project's direct contribution.	Treat standards (CSRD, ISO) as drivers rather than deliverables.
Alternative languages with marginal adoption (e.g., Haskell, niche DSLs)	Alternative languages with marginal adoption	Focus on mainstream languages/runtimes.

Summary and Future Opportunities

Energy-aware computing research demonstrates that substantial energy savings are possible by treating energy as a managed resource across the stack, but the state of practice remains fragmented and difficult to operationalise. Developers typically lack direct, portable control over energy behaviour because key decisions are mediated by compilers, runtimes, operating systems, and cloud platforms, while telemetry is inconsistent and attribution across layers is weak. As a result, many approaches remain hardware- or platform-specific, rely on offline tuning, and struggle to generalise across workloads and deployment contexts.

For GreenCode, this reinforces the need to focus on pragmatic, cross-layer integration: standardised measurement modes and baselines, confidence-aware attribution from system signals to software artefacts, and closed-loop optimisation workflows with regression validation, so energy-aware decisions become repeatable, comparable, and governable in real engineering environments.

Opportunities

GreenCode can effectively advance energy-aware computing into practical, repeatable engineering practice, both to reduce the footprint of GreenCode's own optimisation workflow and to provide reusable approaches for others by focussing on making energy a more explicit and actionable runtime concern by improving cross-layer telemetry integration (hardware > OS/runtime > container > application), developing portable measurement and baselining protocols with confidence bounds,

and enabling closed-loop optimisation that links energy signals to concrete code and configuration changes with regression validation.

The table also captures opportunities to translate research insights (e.g., scheduling, DVFS, batching, algorithmic choices, language/runtime behaviours) into deployable patterns and decision support that teams can adopt without specialist expertise, including CI/CD “green regression” gates and governance-ready reporting.

Opportunity	Description	Example / Implication	Application to GreenCode
Runtime-level energy control interfaces	Expose energy as a first-class controllable resource alongside CPU, memory, and I/O.	Applications or orchestrators can request energy budgets or power caps.	GreenCode can surface runtime and OS-level energy controls via unified APIs and recommendations.
Energy-aware scheduling and workload placement	Optimise task scheduling based on energy efficiency, hardware heterogeneity, and carbon intensity.	Shift workloads to efficient cores or lower-carbon time windows.	GreenCode can integrate scheduler hints and carbon-aware execution strategies.
Adaptive runtime policies (GC, JIT, memory)	Dynamically tune runtime behaviour for energy efficiency rather than peak performance.	Less aggressive GC or JIT compilation reduces power spikes.	GreenCode can recommend or auto-configure runtime profiles optimised for energy.
Energy budgeting and enforcement mechanisms	Enforce energy constraints at container, VM, or application level.	Prevent runaway workloads that exceed energy or carbon budgets.	GreenCode can integrate with cgroups, Kubernetes QoS, or serverless limits.
Energy-aware container and serverless optimisation	Reduce abstraction overhead and inefficiencies in containerised and FaaS environments.	Consolidate microservices or functions to reduce orchestration overhead.	GreenCode can identify energy-inefficient deployment patterns and suggest reconfiguration.
Hardware–software co-design feedback loops	Use hardware telemetry to inform software optimisation decisions.	Software adapts to DVFS states, cache behaviour, or memory bandwidth constraints.	GreenCode can close the loop between hardware signals and software refactoring.
Energy-aware compilation and code generation	Incorporate energy models into compiler optimisation passes.	Compiler chooses lower-energy instruction sequences over fastest ones.	GreenCode can integrate compiler-level insights into its optimisation pipeline.
Cross-platform energy portability models	Understand how software energy behaviour changes across hardware and platforms.	Same code behaves differently on ARM vs x86 or edge vs cloud.	GreenCode can build comparative energy profiles across platforms.

Runtime adaptation under uncertainty	Adapt execution despite noisy or partial energy measurements.	Decisions made with probabilistic energy models rather than exact values.	GreenCode can support confidence-aware runtime optimisation strategies.

GreenCode: State of the Art Review

IT Architecture and Infrastructure Mining, Reconfiguration and Reconstruction

Contributing Partner(s): University of Augsburg, Trier University of Applied Sciences

Background

ISO/IEC/IEEE 42010:2022 (Software, systems and enterprise, Architecture description) defines architecture as the "fundamental concepts or properties of an entity in its environment and governing principles for the realization and evolution of this entity and its related life cycle processes."

In the context of IT, IT-Architecture outlines the fundamental structure and characteristics of the relationships between systems and applications within an organization's IT portfolio[184]. Building upon the structure provided by IT-Architecture, IT Infrastructure comprises the concrete instances of hardware, software, and service components that support the realization of business applications[185]. To make it more tangible, the IT Infrastructure consists mainly of these three elements:

1. Hardware elements, such as servers, CPUs, and networking equipment;
2. Discrete software elements, including full software elements such as VMs, containers, operating systems, software applications and parts of software elements like code snippets, software routines, and functions;
3. Communication flows between software elements.

Building and maintaining an up-to-date IT architecture and IT infrastructure model is challenging due to their dynamic changing nature, lack of detailed documentation, and heterogeneity of systems[186]. To address these challenges, IT architecture and IT infrastructure mining techniques are employed to automatically create and maintain an accurate and up-to-date model of the IT architecture and IT infrastructure. On the one hand, IT architecture mining focuses on extracting and reconstructing architectural knowledge from IT systems to perform analysis such as service response time, change impact, and dependency analysis. On the other hand, IT infrastructure mining detects, maps, and analyses the infrastructure components, their configurations, and dependencies laying the foundation for further monitoring, optimization and reconstruction via IaC.

IT infrastructure mining and monitoring processes build the foundation that enables the sustainability analysis of an IT architecture with appropriate KPIs and patterns. For instance, discovery and monitoring can identify available and unused but wasteful resources, such as in the case of Twitter which rediscovered an idle GPU cluster[187]. Next, the analysis results can be used to optimise the IT infrastructure. One optimization, for example, is VM consolidation and optimized placement of applications, which can reduce energy consumption by up to 22%[188]. Finally, the monitoring and optimisation results can be employed to reconstruct and reconfigure the existing IT infrastructure.

Current State of the Art

In the following, an overview of current approaches for IT infrastructure mining, monitoring, analysis optimisation, and reconstruction is given.

IT Infrastructure Mining and Monitoring

In academic literature several approaches have emerged for IT infrastructure mining. An early crawler-based approach for discovering the complete enterprise IT was proposed by Binz et al.[189]. Their

approach introduces a discovery framework architecture that automatically identifies and maintains the components of an Enterprise Topology Graph (ETG). An ETG represents a snapshot of an enterprise IT landscape, and contains detailed technical information about each component, such as processes, services, software, infrastructure, and their relationships and dependencies[190].

At the core of the proposed framework of Binz et al. is the discovery layer, which iteratively builds the ETG by crawling the existing IT infrastructure using specialized plugins. A plugin is a software component that retrieves information from a specific data source, such as a web server. The execution order of these plugins is determined by a scheduler, and the final ETG is refined by a reconciliation component to ensure data quality. The overall coordination of the discovery process is handled by a discovery manager, which also allows users to specify the starting point of the crawl. This approach has the advantage that various data sources can be integrated via plugins and, therefore, remains expandable.

In the related domain of Enterprise Architecture Management (EAM), Farwick et al.[191] and Holm et al.[192] each propose a method to create an Enterprise Architecture (EA) model automatically. EAM aims to capture the relationship between business processes and underlying IT architecture in an EA model. This model is then utilized to analyse and optimize the IT architecture to align with the overall business strategy. To keep the model up-to-date, Farwick et al. propose creating and continuously updating EA models by obtaining information (semi)-automatically from humans and different technical interfaces, namely web services and business process management tools. Similar to Farwick et al., Holm et al. automate the creation of EA models by scanning the network. They divide the information obtained from a network scan into three categories: System information in the form of MAC and IP addresses; user information including active hosts; and software information such as the operating system, its version, applications running on it, and the applications' protocols, types, and versions.

In a more recent approach Harzenetter et al.[193] propose a method to dynamically create an instance model of the running IT infrastructure capturing details such as component configurations, runtime parameters, and their deployment context. An instance model is generated in three steps in their approach. First, the instance information retriever extracts information about the running applications by querying the APIs of the underlying deployment technologies. This retriever offers a plugin interface to support different deployment technologies and to be expandable. Each plugin extracts information for its specified deployment technology, such as Kubernetes or Puppet. Second, the technology-specific data are converted into a technology and vendor-agnostic model format, namely Topology and Orchestration Specification for Cloud Applications (TOSCA). Finally, Harzenetter et al. utilize the crawler approach of Binz et al. to further enrich the model with additional data. The resulting model contains technology-agnostic deployment data and application-specific information that enable the generation of management activities, such as backup creation and dependency updates.

In addition to academic approaches, several tools have been developed in the industry for IT infrastructure discovery and monitoring. Discovery focuses on identifying and mapping the components and their relationships within the IT infrastructure, while monitoring extends this by continuously collecting runtime data from the discovered components, such as performance metrics, availability, and usage patterns. Common IT infrastructure mining/discovery and monitoring tools are listed in below. The tools are compared based on their license, ongoing support, primary user interface, and target business size. Additionally, information is provided about Individual strengths and

whether the tool considers sustainability aspects in any form, e.g. captures energy consumption or estimates CO2e emissions.

GreenCode: State of the Art Review

Tool	License	Support	Primary User Interface	OS Support	Target Business Size	Sustainability Assessment	Strengths	Type
Qbilon[194]	Proprietary	active	web	discovers: cloud & on-premises	small, medium	none native, but identifies unused and oversized asset	Graph-based,multi-source connectors	Discovery / IT-landscape mining
Device42[195]	Proprietary	active	web	discovers: cloud & on-premises	small, medium, large	Optional Power & Environmental module: live PDU power, temp, rack kW	Mature DCIM + CMDB, broad integrations	Discovery / IT-landscape mining
ServiceNow Discovery[196]	Proprietary	active	web	discovers: Linux, windows, mac, cloud, network, mainframe	medium, large	ESG reporting possible	-	Discovery / IT-landscape mining
BMC Helix Discovery[197]	Proprietary	active	web	discovers: cloud & on-premises	medium, large	none native	-	Discovery / IT-landscape mining
Faddom[198]	Proprietary	active	web	discovers: cloud & on-premises	small, medium, large	none native	-	Discovery / IT-landscape mining
Virima[199]	Proprietary	active	web	discovers: cloud & on-premises	small, medium, large	none native	-	Discovery / IT-landscape mining
Lansweeper[200]	Proprietary	active	web	discovers: IT, OT, IoT, and Cloud	small, medium, large	none native	-	Discovery / IT-

								landscape mining
Nagios[201]	Open Source GPL v2.0	active	web	monitors: windows, Linux	small, medium, large	no native support, could be integrated via plugins / extensions	open-source, community addons, established (25+ years)	monitoring
Zabbix[202]	Open Source AGPL-3.0 license	active	web	monitors: windows, Linux, IoT, cloud	small, medium, large	no native support, could be integrated via plugins / extensions	open-source, established (20+ years)	monitoring
SolarWinds[203]	Proprietary	active	web	monitors: windows, Linux, mac os, cloud	small, medium, large	supports EnergyWise protocol to obtain energy data of network devices no native support for energy / CO2e estimation for servers and applications	established (20+ years),	monitoring
Datadog[204]	primarily proprietary SaaS parts Apache-2.0	active	web, mobile	monitors: windows, Linux, cloud, IoT, mac os	small, medium, large	integration with HardwareSentry to obtain data on power consumption electricity costs and CO_2 emission, not directly application / process specific	parts are open-source, sustainability integration for hardware, supports NVIDIA DCGM Exporter	monitoring
Paessler PRTG[205]	Proprietary	active	web	monitors:, Windows (server), some features via SNMP probes, cloud	small, medium, large	No native sustainability/CO_2e support; can be extended via custom sensors or scripts, has support to query data from PDUs and UPSs	established, customizable	monitoring

Icinga[206]	GPL 2.0	active	web	monitors: windows, Linux, cloud	small, medium, large	No native support for energy/CO2e; possible via plugins/integrations	open-source, extensible via plugins	monitoring
Dynatrace[207]	Proprietary	active	web	monitors: Windows, Linux, cloud	small, medium, large	supports interfaces like IPMI to obtain energy consumption of servers, Dynatrace Carbon Impact app to estimate energy and CO2e of running resources	security focused, full-stack, sustainability focus	monitoring
Netdata[208]	GPL-3.0+	active	web	monitors:: Windows, Linux, macOS, cloud	small, medium, large	No native sustainability/CO2e assessment for applications, however supports Sense Energy Monitoring for IoT / home energy monitoring	Real-time, high-resolution, open-source	monitoring
New Relic[209]	Proprietary	active	web	monitors: Linux, windows, mac os, cloud	small, medium, large	integration with HardwareSentry to obtain data on power consumption electricity costs and CO_2 emission, not directly application / process specific	established, Full-stack observability	monitoring
Elastic Observability[210]	Elastic License 2.0	active	web	monitors: Linux, windows, mac os, cloud	small, medium, large	No native support; can ingest energy/CO2e data via custom plugins	Unified logging/metrics/traces	monitoring
Kepler[211]	Apache License (version 2.0) and GNU	active	CLI, API, dashboards via Grafana	monitors: Kubernetes Cluster	small	Specialized for energy monitoring in Kubernetes; tracks power/CO2e at container level	open-source, specialized for energy monitoring	monitoring

	General Public License, Version 2 or the BSD 2 Clause license							
Scaphandre[212]	Apache-2.0 license	active	CLI, API, Grafana dashboards	monitors: Linux (main), some support for Windows via WSL	small	Specialized for energy monitoring; tracks power/CO2e at process and VM level	open-source, specialized for energy monitoring	monitoring
Amazon CloudWatch[213]	Proprietary	active	AWS Management Console, CLI, API	Linux, Windows,	small, medium, large	AWS provides operational telemetry that can support proxy-based sustainability signals, but provider-reported emissions data is typically delivered via separate carbon accounting dashboards and exports (e.g., AWS Customer Carbon Footprint Tool in the Billing console) rather than the monitoring layer itself.	Deep AWS integration, scalable metrics, strong alerting & automation	monitoring
Azure Monitor[214]	Proprietary	active	Azure Portal, CLI, API	Linux, Windows	small, medium, large	Cloud emissions reporting is provided via dedicated sustainability tooling (e.g., Emissions Impact Dashboard and Azure	Tight Azure integration, powerful log analytics, rich dashboards	monitoring

						Carbon Optimization data/insights), while Azure Monitor remains primarily an operational telemetry platform.		
Google Cloud Monitoring[215]	Proprietary	active	Google Cloud Console, CLI, API	Linux, Windows	small, medium, large	Carbon Footprint API, sustainability recommendations	Fast setup, OpenTelemetry support, integrated tracing & profiling	monitoring
Alibaba CloudMonitor[216]	Proprietary	active	Alibaba Console, API	Linux, Windows	small, medium, large	Green Computing Reports	Cost-effective, Prometheus support, strong Alibaba ecosystem integration	monitoring

Limitations and Gaps

Infrastructure mining and architecture reconstruction rarely translate into sustainability outcomes today. As the table shows, most discovery/monitoring tooling still treats sustainability as secondary (optimising for availability, performance, and cost first), with limited native integration of energy telemetry APIs and weak linkage between architecture views and energy hotspots at component/flow level. It also highlights trust and deployment barriers, energy estimates are rarely validated/certified, optimisation strategies often assume homogeneous datacentres rather than hybrid enterprise estates, and many tools lack concrete action primitives for reconfiguration. Emerging constraints that matter for modern systems include: insufficient modelling of GPUs/accelerators, the need for multi-objective optimisation (SLA/cost/energy/cooling), and governance/access constraints on telemetry collection in real organisations.

Limitation / Gap	Description	Example / Implication	Application to GreenCode
Sustainability treated as a secondary concern in infrastructure tooling	Most infrastructure discovery and monitoring tools prioritise availability, performance, or cost over energy and sustainability.	Optimisation focuses on datacentre placement rather than reducing hardware, software, and service consumption.	GreenCode can promote sustainability as a first-class optimisation objective.
Lack of first-class integration of energy telemetry APIs	Infrastructure tools rarely natively integrate APIs such as RAPL, DCMI, NVML, or NVMe interfaces.	Energy metrics are inferred indirectly or omitted entirely.	GreenCode can integrate low-level energy APIs directly into infrastructure models.
Weak linkage between architecture models and energy hotspots	Architecture views typically lack energy annotations at component and flow level.	Energy-intensive subsystems remain invisible in system-level analyses.	GreenCode can augment architecture graphs with energy hotspot detection.
Absence of certified and trustworthy energy estimation mechanisms	Energy estimates produced by tools are rarely validated or certified.	Results may not be trusted for decision-making or reporting.	GreenCode can support validation, calibration, and certification of energy estimates.
Datacentre-centric optimisation strategies	Many optimisation techniques are designed for homogeneous datacentres rather than heterogeneous IT infrastructures ***Error! Bookmark not defined.***	Enterprise and hybrid infrastructures cannot fully exploit existing approaches.	GreenCode can adapt optimisation strategies to broader IT contexts.
Limited action-level optimisation primitives	Tools identify inefficiencies but offer few concrete reconfiguration actions.	Opportunities like caching or communication locality are missed.	GreenCode can propose actionable infrastructure-level optimisations.
Insufficient modelling of accelerators and	GPUs and accelerators are often ignored or oversimplified in	AI workloads dominate energy usage without visibility.	GreenCode can extend models to include GPUs

heterogeneous resources	infrastructure models *Error! Bookmark not defined.*		and specialised hardware.
Lack of multi-objective, multi-agent optimisation frameworks	Current approaches struggle to balance SLA, energy, resource usage, and cooling.	Local optimisations conflict at system level.	GreenCode can explore multi-agent, multi-objective optimisation strategies.
Telemetry governance and access constraints	Collecting infra telemetry can be blocked by security, privacy, and organisational boundaries.	Incomplete models; blind spots in optimisation.	Provide least-privilege collection patterns + evidence-grade audit trails.

Non-Priorities

The directions below are intentionally out of scope to keep the workstream focused on GreenCode's differentiator: sustainability-aware enrichment, actionable recommendations, and evidence-backed optimisation. It de-prioritises building a full CMDB/DCIM or pursuing inventory completeness for its own sake, as well as one-off snapshot discovery that becomes stale in dynamic estates. It also avoids approaches that don't scale or don't drive action, manual documentation-first reconstruction, visualisation-first "pretty diagrams," and performance-only monitoring without sustainability annotations. Finally, we explicitly step away from vendor-locked modelling formats, physical facility engineering, and fully automated reconfiguration without governance, favouring incremental normalisation and explainable recommendations with safe rollout/rollback patterns.

Area	Reason to De-prioritise	Implication
Building a generic CMDB/DCIM/discovery product "from scratch"	Mature enterprise products already dominate discovery/CMDB/DCIM; GreenCode's differentiator is sustainability-aware enrichment + evidence, not replicating full ITAM stacks.	Focus effort on sustainability overlays (energy APIs, hotspot metrics, certified estimation, actionable recommendations) that can integrate with existing inventories.
Pure "inventory completeness" as the primary objective	Chasing 100% asset enumeration often yields diminishing returns and delays sustainability impact; for GreenCode, actionable sustainability signal matters more than exhaustive asset detail.	Prioritise "good-enough topology for optimisation" + continuous update loops over perfect inventories.
One-off/snapshot discovery outputs without reconciliation/change tracking	Static snapshots rapidly become stale in dynamic estates (containers, autoscaling, hybrid). This runs counter to the section's emphasis on continuously maintained models.	De-prioritise tools/methods that cannot support continuous model maintenance and provenance; prefer pipelines with reconciliation and drift detection.
Manual documentation-first architecture reconstruction (workshops/interviews as the main method)	Doesn't scale, is slow to update, and creates governance friction; also tends to miss runtime realities (deployments, dependency drift).	Use humans for validation/intent only; rely on automated evidence (APIs, logs, traces, IaC state) for the model backbone.

Vendor-locked, single-platform modelling formats (no translation/normalisation)	The section highlights heterogeneity and vendor/tool diversity; locking to one vendor model undermines portability and cross-layer reasoning.	Prefer vendor-agnostic intermediate representations (e.g., TOSCA-style normalisation) and adapters to/from popular sources.
"Pretty diagrams" / visualisation-first architecture mining	Visualisations can be valuable, but on their own they don't reduce energy/carbon; the bottleneck is actionable, validated insight (hotspots → interventions).	Invest in diagnosis + recommendation + verification loops first; visualisations are secondary outputs.
Performance/availability-only mining and monitoring (sustainability as an afterthought)	Mature APM/monitoring already covers performance/uptime; GreenCode's unique contribution is making sustainability a primary optimisation lens.	Avoid duplicating APM; instead extend mining outputs with energy/carbon annotations, hotspot metrics, and sustainability-aware reconfiguration actions.
Attempting to model physical facility upgrades (cooling plant redesign, PUE engineering) as a core deliverable	This crosses into facilities engineering and datacentre operations; important, but outside the software/IT mining scope and likely partner capability boundaries.	Keep scope on software/IT infrastructure representations and decisions that software teams can influence (workload placement, consolidation, configuration).
Fully automated reconfiguration with no governance/approval path	Infra changes carry risk (SLA, security, compliance). "Auto-change" approaches are often blocked organisationally.	Prioritise **explainable, ranked recommendations** + safe rollout/rollback patterns over hands-off automation.
Trying to "standardise everything" before delivering value	Large standardisation programmes slow down delivery and adoption; better to incrementally normalise around a minimum viable schema and expand with evidence.	Start with a minimal cross-layer schema + adapters; iterate based on what drives hotspot detection and optimisation.

Summary and Future Opportunities

IT infrastructures are contributors to the ICT sector's energy consumption, and so their optimisation presents an opportunity to reduce the environmental impact of the ICT sector. In order to achieve this, the existing infrastructure must first be discovered, monitored, and analysed for sustainability inefficiencies, such as energy hotspots. Based on this analysis, the IT infrastructure can be optimized for energy efficiency and finally reconstructed into a more sustainable state. The core problem, however, is that the existing tools and practices for these stages are fragmented. To create an end-to-end framework, several advances are required in the two key areas: first, tools for accurate energy estimation and hotspot detection in IT infrastructure, and second, multi-objective optimization approaches that balance cost, performance, and energy consumption across all layers, including hardware and cooling to applications and their communication.

As a result, three new contribution opportunities are opening up for the GreenCode project. Firstly, the creation of new mining and monitoring tools that treat sustainability as a primary focus. This would involve integrating energy-relevant APIs, such as RAPL, NVIDIA NVML, and IPMI 2.0, and based on this, creating new, certified energy estimation models that enable the precise estimation of energy consumption across all layers of the IT infrastructure. Secondly, based on this, new metrics for detecting sustainability hotspots can be derived, focusing on energy and related carbon emissions in the areas of hardware, software, and communication. Thirdly, new optimization approaches can be developed that balance cost, performance, and energy consumption across all layers of the IT infrastructure.

Opportunities

GreenCode can turn infrastructure mining into an end-to-end sustainability optimisation capability. By emphasising moving from static discovery to continuous, energy-annotated architecture reconstruction, and extending infrastructure representations, energy/carbon can become first-class attributes across systems, services, and flows. Unifying multi-layer topology (hardware > platform > application) and pairing it with actionable recommendation engines and scenario-based "what-if" analysis to support safe decision-making could also be beneficial.

Several opportunities directly address modern estate realities, energy-aware workload placement across infrastructures, accelerator-aware optimisation, multi-objective trade-off frameworks, and trustworthy/certifiable assessment, while also enabling cross-stakeholder transparency and dynamic resource allocation as governance-ready operational practices.

Opportunity	Description	Example / Implication	Application to GreenCode
Continuous, sustainability-aware architecture reconstruction	Move from static or periodic discovery to continuously updated architecture models enriched with energy data.	Architecture views remain accurate despite scaling, redeployment, or workload changes.	GreenCode can maintain live, energy-annotated architecture graphs.
First-class energy-aware infrastructure models	Extend architecture and infrastructure representations to include energy and carbon attributes.	Energy becomes visible at system, service, and flow level.	GreenCode can embed energy metrics directly into system models.
Unified multi-layer system representations	Integrate hardware, infrastructure, platform, and application topology into a single model.	Enables end-to-end optimisation rather than siloed improvements.	GreenCode can act as a unifying system-of-systems model.
Actionable reconfiguration recommendation engines	Translate mined insights into concrete, system-level reconfiguration actions.	Suggest workload relocation, consolidation, caching, or topology changes.	GreenCode can generate ranked, explainable optimisation recommendations.
Scenario-based "what-if" sustainability analysis	Simulate architectural changes before execution to estimate	Enables safer and more ambitious infrastructure changes.	GreenCode can support decision-time sustainability modelling.

	energy and performance impact.		
Energy-aware workload placement across infrastructures	Optimise placement based on interaction patterns, resource type, and energy efficiency.	Reduce communication overhead and unnecessary data movement.	GreenCode can guide placement strategies across datacentres and clouds.
Heterogeneous resource-aware optimisation	Incorporate GPUs, accelerators, and specialised hardware into optimisation strategies.	Prevents AI workloads from dominating infrastructure energy use unnoticed.	GreenCode can extend infrastructure models to accelerator-heavy environments.
Multi-objective infrastructure optimisation frameworks	Balance energy efficiency with SLA, cost, reliability, and resilience.	Avoids single-metric optimisation failures.	GreenCode can support trade-off-aware optimisation strategies.
Certified and trustworthy infrastructure energy assessments	Develop validated, auditable energy estimation methods at infrastructure level.	Builds trust for procurement, reporting, and governance.	GreenCode can support calibrated and certifiable infrastructure metrics.
Cross-stakeholder system transparency	Provide shared, sustainability-aware system views for architecture, operations, and sustainability teams.	Reduces misalignment between technical and sustainability decisions.	GreenCode can act as a shared decision-support platform.
Dynamic resource allocation	Workload-aware scaling and runtime resource adaptation.	Energy savings by adjusting compute allocation dynamically.	Integrate profiling + scaling advice into optimisation pipelines.

GreenCode: State of the Art Review

Standards and Certification for Green Software

Contributing Partner(s): Digital Tactics

Background

Today the ICT sector is estimated to be responsible for 2-4% of global GHG emissions, rising to ~14% by 2040 if unchecked[Error! Bookmark not defined.]. A computer or server "can only be as energy efficient as the software allows" [217] so regardless of physical optimisations that may be present in modern IT hardware (e.g. energy efficient CPU's/GPU's) or in facilities like datacentres (advanced cooling, energy management), if the software they are running is inefficient, the system will remain inefficient.

Poorly optimized code causes unnecessary processor load, data transfer, and even shortens hardware lifespan, not because the hardware is worn out but because more powerful hardware is required to operate the software, driving more frequent upgrades[217]. Thankfully though due to the rapid pace updates can be deployed at, if inefficient software is optimised then a step change beneficial impact in performance and energy efficiency can be realised on deployment.

Over the past decade, green software engineering has emerged as a discipline focused on reducing the carbon footprint of software through best practices[Error! Bookmark not defined.] . This is motivated by both environmental concerns and upcoming regulations.

Efficient software often runs faster and costs less to operate, dispelling the myth that sustainability must come at extra cost[Error! Bookmark not defined.]. Furthermore, software teams increasingly recognize their responsibility in mitigating climate change, especially as new laws require organizations to report the emissions linked to their digital products and services[Error! Bookmark not defined.] .

Standardizing what constitutes "green" software is important not only for consistency and credibility in market, but also to enable certification, compliance, and broad adoption of sustainable software practices.

GreenCode: State of the Art Review

Current State of the Art

Standardisation and Regulatory Alignment

Although the Green Software Foundation's SCI specification provides a starting point for measuring software carbon intensity110, adoption is limited and standards remain fragmented. Broader regulatory frameworks, such as the EU Corporate Sustainability Reporting Directive (CSRD), increasingly mandate disclosure of environmental impacts114, yet lack software-specific methodologies. Without clear and consistent reporting guidelines, developer teams and organisations cannot easily benchmark or demonstrate compliance. Future efforts must therefore prioritise harmonisation between industry-led specifications, academic methodologies, and regulatory frameworks, ensuring that sustainability metrics are both technically robust and policy-relevant.

Similar regulatory pressures are emerging internationally, including through the European Green Deal and in the United States through proposed SEC climate disclosure rules, and are particularly pronounced in public sector procurement, where sustainability and reporting requirements increasingly influence software and IT purchasing decisions [218, 219]

While recent standards such as the Software Carbon Intensity (SCI) specification represent a significant step forward in defining how the carbon impact of software systems can be measured and reported, their practical application remains challenging. Most standards focus primarily on *quantification and disclosure* rather than on *how software systems should be designed, implemented, or operated to reduce emissions*. As a result, compliance with emerging standards does not necessarily translate into meaningful improvements in software energy efficiency, particularly at the level of code, architecture, or runtime behaviour.

Global Standards and Frameworks

Several standards and industry frameworks now address software sustainability, either by repurposing existing ICT standards or by developing new guidance specific to software**Error! Bookmark not defined.** . In 2024 the Green Software Foundation's (GSF) Software Carbon Intensity (SCI) specification, a methodology to calculate a software system's carbon emissions per functional unit, was adopted as ISO/IEC 21031:2024, making it a formal international standard[220].

SCI now provides a globally recognized metric for software carbon footprint, enabling organizations to measure and track the carbon intensity of software systems in a consistent manner, and to compare different configurations or successive versions of the same system under controlled assumptions[220]. The SCI metric builds on established GHG accounting principles, aligning with frameworks like ISO 14064 (which governs quantification and verification of organizational GHG emissions) and the GHG Protocol[220]. In essence, ISO 21031:2024 fills a long-standing gap by defining how to measure software's contribution to carbon emissions, helping make software sustainability "measurable and manageable" rather than an afterthought[220].

Although SCI provides a common language for comparing software carbon intensity, it is largely agnostic to the internal structure of software systems. The metric does not prescribe how emissions should be reduced, nor does it directly integrate with software engineering artefacts such as code repositories, architectural models, or CI/CD pipelines. This limits its usefulness as an engineering tool

and positions it primarily as a reporting and benchmarking mechanism rather than a driver of optimisation.

In parallel, general environmental management standards are being applied to software. ISO 14001 (Environmental Management Systems) provides a framework that organizations can extend to cover software-related impacts [Error! Bookmark not defined.]. The software development process can also draw on ISO 14062 (guidance on eco-design) to integrate environmental considerations in design and engineering [Error! Bookmark not defined.].

Traditional software quality standards like ISO/IEC 25010 already include attributes such as "performance efficiency", "resource utilization", and even the degree to which a system mitigates risk to the environment[Error! Bookmark not defined.]. Another crucial set of standards come from the ITU (International Telecommunication Union): ITU-T L.1410 provides methods for ICT product life-cycle assessments[Error! Bookmark not defined.], and ITU-T L.1420 offers an energy and GHG assessment methodology at the organization level [Error! Bookmark not defined.] .

Together with ISO 14044 (life-cycle assessment for products)[Error! Bookmark not defined.], they enable a holistic evaluation of software's indirect environmental impacts across its lifecycle (development, deployment, usage, and disposal of supporting hardware). In fact there are now around 150 ICT sustainability standards and recommendations (many from ITU-T) that upcoming policies may draw upon for regulating digital technology [Error! Bookmark not defined.]. This shows that the "greening" of existing standards is well underway, even as new, software-specific standards are introduced, but while this is the case they remain far from widescale implementation.

At the infrastructure and system benchmarking level, additional standards complement software-focused sustainability metrics. The ISO/IEC 30134 series defines widely used data centre sustainability key performance indicators, including Power Usage Effectiveness (PUE), Carbon Usage Effectiveness (CUE), and Water Usage Effectiveness (WUE) [221]. For system-level efficiency benchmarking, the SPEC SERT suite extends traditional server performance evaluation by incorporating workload-level energy measurements[222]. In the context of artificial intelligence, benchmarks such as MLPerf-Energy have recently introduced energy-aware extensions that capture both performance and energy efficiency during model training and inference, reflecting the growing energy impact of AI workloads[223].

GreenCode: State of the Art Review

Benchmarks & Standards Snapshot (software-relevant)

The table below summarises the most relevant software- and system-level standards and benchmarks currently used to assess energy efficiency and environmental performance across different layers of the software stack.

Name	Classification	Scope	Primary Function	Energy Coverage	Comparability	Maturity / Status
Software Carbon Intensity (SCI) / ISO/IEC 21031:2024	International standard (metric framework)	Software system	Calculation and reporting of software carbon intensity	Direct (estimated via energy & carbon factors)	Within-system (across configurations/iterations)	ISO standard
Green Software Foundation – Measurement Guidance (non-SCI)	Industry framework (guidance)	Software systems	Guidance on sustainability metrics and reporting practices	Indirect / methodological	Not intended for comparison	De facto, evolving
ISO/IEC 30134 (PUE, CUE, WUE) [221]	International standard (infrastructure KPIs)	Data centre / infrastructure	Facility-level energy, carbon, and water efficiency KPIs	Indirect (context/proxy)	Cross-facility	ISO standard
SPEC SERT Suite[222]	Benchmark suite	Server / system	Comparative evaluation of system-level energy efficiency under defined workloads	Direct (measured energy)	Cross-system (controlled workloads)	De facto industry benchmark
MLPerf-Energy	Benchmark suite	AI training & inference systems	Joint benchmarking of AI performance and energy efficiency	Direct (measured energy)	Cross-system (benchmark-specific)	De facto industry benchmark
Carbon Aware SDK	Developer tooling / operational framework	Deployment & operations	Carbon-aware scheduling / workload shifting using grid carbon signals	Indirect (grid carbon intensity)	Not intended for comparison	Production-ready tooling

MLCO$_2$[224] and Hugging Face emissions reporting [225]	Transparency & reporting framework (voluntary)	AI model training & publishing	Voluntary estimation and disclosure of training emissions (often via model cards)	Indirect (estimated energy & carbon via CodeCarbon and similar)	Not intended for comparison	Widely used, voluntary
DIMPACT	Measurement framework (industry-led)	Digital services / media supply chains	Calculation of environmental impacts across digital value chains	Indirect (model-based)	Not intended for benchmarking	Industry initiative
ISO/IEC 25010 (Performance Efficiency characteristics)	International standard (software quality model)	Software product quality	Defines performance efficiency attributes (time/resource utilisation)	Indirect (proxy)	Not intended for comparison	ISO standard
ISO 14040 / ISO 14044 (Life Cycle Assessment)	International standard (LCA methodology)	Products & services (cross-sector)	Lifecycle-based environmental impact assessment method	Indirect (methodological)	Depends on study design	ISO standard
AFNOR SPEC:– General framework for frugal AI	Pre-standard / specification (guidance)	AI systems	Guidance to measure/reduce AI environmental impact across lifecycle	Indirect (principles & practices)	Not intended for benchmarking	Pre-standard (AFNOR SPEC)

Initiatives and Organizations

Initiatives led by organisations such as the Green Software Foundation, the Software Sustainability Institute, and the Green Web Foundation have played a critical role in raising awareness, establishing shared terminology, and promoting best practices. However, their impact is constrained by their emphasis on guidance, education, and voluntary adoption. Given the scale and heterogeneity of the global software industry, standards and principles alone are insufficient to drive widespread behavioural change without complementary tooling that embeds sustainability directly into engineering workflows.

The Green Software Foundation

The GSF was established in 2021 and has become a key driver in coordinating industry efforts. It has grown to dozens of member organizations (including major tech firms like Google, Intel, Microsoft, and Salesforce) dedicated to advancing sustainable software practices[Error! Bookmark not defined.]. They have published open resources such as Green Software Design Patterns and a Carbon Aware SDK. The

Carbon Aware SDK is an open toolkit that enables applications to run workloads at times or locations where electricity is cleaner (i.e. with lower carbon intensity) [220], an example of carbon-aware computing in action.

They target three main principles in their work: Software energy efficiency reflecting the use of less power for the same work; Hardware efficiency reflecting the use of less physical equipment; and Carbon awareness which reflects scheduling and load shifting to use greener energy[Error! Bookmark not defined.]. They also offer training (e.g. a Green Software Practitioner course) and foster knowledge exchange in the community.

Through such initiatives, industry is attempting to supplement formal standards with practical tools and best practices, accelerating what companies can do now while standards bodies catch up.

However, while their work is admirable in terms of coordinating large players in software engineering and deriving some consensus on standards, its members account for a small proportion of the total software industry[226] and outside of specialist communities, a significant proportion of the software industry is not impacted or engaged by their work.

Significant effort needs to be applied to make a real difference through education and standardisation, and it may be the case that, like security, accessibility and other topics, full (or even close to full) engagement is never achieved.

This is where tools like GreenCode will make a key difference in the application of green software standards. By abstracting the responsibility for the application of green software principles away from the specific developer and onto the system at large the need for every developer to be adequately educated is removed. It is beneficial of course if the developer is educated on these topics but the requirement and burden of this is reduced. GreenCode will create educational feedback loops and materials for developers based on the action it has taken on their codebases also spreading the reach of the green software message into areas where it may not otherwise naturally occur.

The Software Sustainability Institute

The Software Sustainability Institute (SSI), predates the GSF by over 10 years and was the world's first dedicated organization for academic research software sustainability. It has become a global leader in promoting better practices. Based at UK universities including Edinburgh, Manchester, and Southampton, and funded by major research councils (EPSRC, ESRC, BBSRC, UKRI, JISC, NERC), the SSI has received more than £17 million to date in cumulative funding.

Its activities span training, consultancy, and community building: it has supported more than 210 fellows, trained over 8,000 researchers, and partnered with initiatives like The Carpentries to scale national training capacity. Crucially, SSI pioneered the Research Software Engineer (RSE) movement, co-founding the UK RSE Association and supporting the formation of the Society of Research Software Engineering, thereby transforming the recognition and professionalization of research software work.

At SSI has advanced sustainability in research software by embedding sustainability as a recognized quality attribute, promoting reproducibility and maintainability, and developing frameworks to assess

and improve software health. Its workshops and reports (e.g., WOSSS19) help set agendas for sustainable software research, while its consultancy and collaborative projects provide real-world validation across disciplines and countries.

Like the GSF the SSI's impact lies in cultural change: normalizing the idea that research software is a first-class research output requiring long-term stewardship. In this sense, SSI not only shapes academic discourse but also provides infrastructure and policy influence.

The Green Web Foundation

Founded in 2006, the Green Web Foundation (GWF), predates both the SSI and GSF. It is a non-profit organization dedicated to decarbonizing the internet by promoting transparency and accountability in web hosting and digital infrastructure. Its most widely used service is the Green Web Dataset, an open and continuously updated directory of web hosts and services that are verified green providers. This dataset underpins the popular "green check" tool, which allows users and developers to quickly verify whether a website runs on sustainable infrastructure. GWF also offers APIs and browser extensions that integrate green checks into everyday workflows, thereby normalizing carbon-aware decision-making for developers, organizations, and end-users alike.

Beyond hosting verification, the Foundation has broadened its activities to education, community-building, and tooling for carbon-aware software development. Initiatives include training programs on sustainable web practices, open-source libraries for carbon-aware operations, and partnerships with organizations like the Climate Action Tech community.

It plays a unique role by bridging grassroots developer communities and sustainability advocates with industry efforts, complementing broader frameworks like the Green Software Foundation. At the state of the art, GWF provides practical, lightweight interventions—APIs, badges, and open data—that make sustainability actionable for the wider developer ecosystem, particularly SMEs and civil society groups that are often outside the scope of larger corporate consortia.

The Sustainable Games Alliance

An often-overlooked area of the IT and software sector is the video gaming. Like TV and film production it tends to be categorised under creative industries but is at its heart as much enterprise software as it is creative endeavour. Moreso in some cases given the demands and expectations of its audience.

The Sustainable Games Alliance (SGA) is a member-governed non-profit cooperative, founded by game industry veterans and environmental researchers, with the mission of making the games sector a sustainability leader by establishing ambitious, industry-specific standards that go beyond mere compliance[227] [228].

Recognizing the inadequacy of one-size-fits-all frameworks, the SGA has developed the SGA Standard—a comprehensive, open-access toolkit tailored to the unique needs of game production.

Based on the GHG Protocol, the standard simplifies compliance with diverse regimes (e.g., EU CSRD/ESRS; California's SB 253/SB 261; and emerging UK Sustainability Reporting Standards (UK SRS)

based on ISSB/IFRS S1/S2 through actionable templates, benchmarking, emissions assessment methodologies (including Scope 1, 2, and refined Scope 3 calculations), and sector-relevant data collection guidance [228] [229]. This initiative is informed by ongoing research e.g. from the EU-funded STRATEGIES project, which ensures the standard's robustness and credibility through expert review [228].

SGA occupies a pivotal role in the state-of-the-art for sustainable software by bridging the sustainability gap within a high-impact creative industry. It provides tailored compliance pathways, allowing developers to benchmark, report, and reduce emissions effectively and cost-efficiently. Its focused scope on industry-specific emissions (like employee commuting, home-working energy use, digital distribution, and gaming hardware life cycles) enables meaningful interventions that traditional general-purpose ESG frameworks miss [229].

By aggregating data across members, promoting shared methodologies, and grounding the approach in research-backed validation, SGA is pioneering an actionable, scalable, and sector-aligned model for embedding sustainability into the operational DNA of game development studios worldwide.

IEEE Communities on Sustainable Computing and Green ICT

The Institute of Electrical and Electronics Engineers (IEEE) has been instrumental in formalizing sustainability within computing through its technical societies, conferences, and standards initiatives. Two major communities lead this space: IEEE Sustainable Computing (within the IEEE Computer Society) and IEEE Green ICT Initiative. Together, they provide an academic and industrial platform for advancing research, education, and standardization on environmentally responsible ICT.

The IEEE Green ICT Initiative, launched in 2015, coordinates across IEEE societies to address ICT's environmental footprint, covering areas such as energy-efficient networking, green cloud computing, and lifecycle assessments of digital systems. Meanwhile, IEEE Sustainable Computing focuses more narrowly on algorithmic efficiency, low-power architectures, and software-hardware co-design, producing publications and organizing tracks at flagship conferences like IEEE e-Energy and IEEE Sustainable Computing.

These IEEE efforts are notable in the state of the art because they connect fundamental research, standardization, and policy influence. They are closely involved in the development of IEEE standards (e.g., IEEE 1680 series on environmental assessment of electronics) and collaborate with ISO and ITU on international green ICT metrics. Importantly, IEEE communities also frame sustainability as a cross-cutting concern: embedding energy efficiency and environmental performance alongside traditional concerns like reliability, performance, and security. This positions IEEE as a key knowledge broker—translating research breakthroughs into globally recognized standards and disseminating best practices across academia, industry, and government. While their reach is strongest in scholarly and professional communities, their outputs (conferences, standards, publications) directly influence the broader ICT sustainability ecosystem.

Certification Schemes

Certification schemes and assurance mechanisms play a complementary role to standards, benchmarks, and measurement frameworks by providing independent validation of sustainability claims rather than defining how energy or environmental performance is calculated or compared. Whereas standards and benchmarks establish metrics, methodologies, and controlled evaluation conditions, certification schemes focus on verifying compliance against defined criteria, supporting trust, procurement decisions, and regulatory alignment.

In the context of green software, certification and assurance initiatives typically operate at the level of organisations, services, or products, and often rely on underlying measurement standards or lifecycle assessment methods without prescribing specific software engineering techniques. This subsection covers formal ecolabels, assurance bodies, and closely related pre-standardisation efforts that shape emerging certification practices, while remaining distinct from the benchmarking and tooling ecosystems described earlier.

Blauer Engel (Blue Angel)

Formal certification programs for green software are still nascent, but a few pioneering schemes have emerged. In Germany, the government-backed Blue Angel (Blauer Engel) ecolabel, the world's first environmental product label, introduced criteria for "Resource and Energy-Efficient Software Products" (award criterion DE-UZ 215) in 2020 [217,230].

This Blue Angel certification assures that a software product meets stringent requirements for low energy consumption, frugal use of hardware resources, and user-transparency. For example, Blue Angel-certified software must demonstrate that it uses hardware in an especially efficient manner, saves energy, and can extend hardware lifespan by remaining performant on older devices[217]. It also forbids practices like excessive data collection, advertising bloat, or unnecessary network use that would waste energy[230] . Initially limited to desktop applications, the criteria were recently expanded (in 2024) to cover mobile and server software as well[217].

Being first, the Blue Angel serves as a de facto international benchmark for green software products. However, adoption has been limited so far, the first software to earn it was the KDE Okular PDF reader in 2021[217], and only a handful of products (including an open-source cloud server and a web analytics tool) have followed[217]. The updated criteria aim to encourage broader uptake, offering more flexible testing methods and requiring ongoing annual re-measurement of energy use to ensure software updates remain efficient[217]. Observers note that making such sustainability achievements visible via labels like Blue Angel is important to drive awareness and accountability in the software industry[217].

Numérique Responsable

Another notable certification is France's "Numérique Responsable" (Responsible Digital) label, launched in 2019 by the Institut du Numérique Responsable with government support[231]. Rather than certifying a specific software product, this label recognizes organizations (companies, public agencies, etc.) for implementing a broad responsible IT strategy covering environmental, social, and economic aspects of digital technology[231].

Participating organizations undergo an audit and commit to continuous improvement in areas like energy efficiency of IT operations, electronic waste reduction, digital inclusion, and so on [231]. The French NR label has multiple levels and is becoming a national benchmark for sustainable IT management, with large enterprises and even cities achieving certification [231,232]. While its scope goes beyond just software development, it highlights that at an organizational level, sustainable software practices are part of a larger digital responsibility framework [231]. Companies like CGI and Pierre Fabre in France have publicized obtaining this label as proof of their commitment to greener IT[232].

General Framework for Frugal AI (AFNOR SPEC)

The "General framework for frugal AI", published as an AFNOR SPEC, represents a recent and notable effort to address the environmental impacts of artificial intelligence systems through structured guidance rather than formal certification. As a pre-standardisation specification, the AFNOR SPEC does not constitute a binding standard or certification scheme, but instead provides a shared reference framework intended to inform future standards, best practices, and assessment methodologies.

The specification focuses on the concept of *frugality* in AI, encompassing not only energy consumption during model training and inference, but also broader considerations such as data efficiency, hardware utilisation, deployment choices, and lifecycle impacts. Importantly, the AFNOR SPEC emphasises the need to consider environmental impacts across the full AI system lifecycle, rather than treating energy efficiency as a narrow performance optimisation problem. In the context of green software, this framework highlights the growing convergence between AI governance, sustainability objectives, and software engineering practices, and illustrates how pre-standardisation efforts can shape the evolution of future certification schemes and regulatory expectations for environmentally responsible AI systems.

Carbon Trust (Certification and Assurance)

The Carbon Trust is a well-established, independent organisation that provides carbon measurement, assurance, and certification services across a wide range of sectors, including digital and ICT systems. In the context of software and digital services, the Carbon Trust plays a key role in translating high-level sustainability reporting requirements into verifiable claims, supporting organisations in measuring, disclosing, and validating the environmental impacts of their products and services.

Carbon Trust certifications and labels are typically grounded in lifecycle assessment (LCA) methodologies and internationally recognised standards, and are often used to demonstrate credibility to customers, regulators, and procurement bodies. While the Carbon Trust does not prescribe specific software engineering practices, its schemes exert indirect influence on software development and operation by incentivising organisations to adopt measurable, auditable approaches to emissions reduction. As such, Carbon Trust certification functions as an important market signal and assurance mechanism, complementing more technically detailed software-level metrics and engineering frameworks by providing external validation of sustainability claims.

DIMPACT

DIMPACT is an industry-academic collaboration, led by the University of Bristol, that provides a standardized methodology and toolkit for assessing the carbon footprint of digital media services. Originally developed with partners including BBC, Netflix, ITV, and Sky, DIMPACT focuses on the

emissions associated with content delivery, advertising, publishing, and video streaming—sectors with rapidly growing digital footprints. Rather than being a certification in itself, DIMPACT provides a boundary-spanning accounting framework that translates complex supply chains and infrastructure dependencies into measurable greenhouse gas emissions. By offering a transparent, sector-specific approach grounded in the GHG Protocol, DIMPACT has been recognized as a *de facto* benchmarking tool in the media industry. Its existence highlights how sustainability standards are emerging through coalitions of industry stakeholders around shared measurement challenges, setting useful precedents for other digital domains, including software development and cloud-based services.

Community Labels

Apart from official labels, community-driven badges have appeared. For example, the Green Web Foundation maintains a directory of websites hosted on renewable energy and offers a badge for websites that are "running on green energy." Similarly, tools like EcoGrader and Website Carbon Badge allow web developers to assess and display the carbon footprint of their sites.

These are not formal certifications, but they contribute to a culture of transparency. They also complement emerging Web-specific standards, the W3C's Sustainable Web Design guidelines (drafted in 2023) provide best practices for creating low-carbon web applications, covering everything from efficient UX/design to green hosting choices[Error! Bookmark not defined.]. This kind of guidance may eventually translate into certification criteria for web services.

Policy and Regulatory Developments

Thus far, no country has explicit energy-efficiency regulations for software products[217], but policies are trending toward including software in broader climate accountability. The EU's Digital Product Passport initiative and Green Deal discussions hint at incorporating digital services into ecodesign and circular economy regulations in the future. Importantly, the EU's Corporate Sustainability Reporting Directive (CSRD) (effective 2024) mandates large companies to report detailed environmental data, including IT usage impacts and product-use phase emissions[Error! Bookmark not defined.].

Software-related emissions (e.g. from cloud usage or end-user devices) would fall under scope 3 (indirect) emissions which firms must quantify[220]. In fact, studies find that in ICT companies, applications and software services account for 67-93% of total ICT GHG emissions, making it a crucial focus for reporting and reduction[220] . This regulatory pressure is driving companies to adopt the new measurement standards and tools. For instance, European firms like banks and IT consultancies have started to apply the SCI ISO standard internally to compute baseline emissions for their software portfolios[220].

They are also pairing it with code-level sustainability analysis, the Object Management Group (OMG) released an Automated Source Code Resource Sustainability Measure (ASCRSM) standard in 2023 to identify common coding issues that waste energy or resources[233]. By using SCI for high-level carbon accounting and ASCRSM for code-quality scanning, organizations can set targets per application and pinpoint inefficiencies in source code (such as algorithmic patterns that cause excessive CPU or memory use)[233] . These concrete methodologies and standards, backed by industry consortia and now ISO/IEC and OMG, represent the state-of-the-art toolkit for making software greener.

GreenCode: State of the Art Review

Academic and Competitive Benchmark

In academia, research on software energy optimization and measurement has flourished, yielding tools like energy profilers, IDE plug-ins for efficiency, and algorithms for energy-aware scheduling. While not formal standards, these research outputs often inform industry best practices. We also see competitions and benchmarks highlighting what is achievable. For example, the earlier Green Code Lab Challenge (2013-2015, later evolved into the international Design4Green competition) brought together student and professional teams to build the most energy-efficient web applications under time constraints[234].

Such contests illustrate techniques for drastic energy reduction in code and have helped raise awareness in the developer community. In high-performance computing, the Green500 ranking (a counterpart to the Top500 supercomputers list) has, since 2007, spurred competition for performance per watt in supercomputing, effectively a benchmark for software-hardware co-optimization.

The lessons from these arenas (e.g. using efficient algorithms, tuning code to hardware, avoiding computation that doesn't add user value) are increasingly being codified into mainstream guidelines (for instance, the Green Software Foundation's pattern catalogue captures many of these ideas for general software development[Error! Bookmark not defined.]).

Limitations and Gaps

The Limitations and Gaps table below summarises why standards and certification, while rapidly evolving, still struggle to drive measurable reductions in real software estates. It highlights fragmentation and scope constraints (overlapping schemes, niche certifications), weak market/legal drivers, and the practical difficulty of robust measurement and attribution, particularly in modern cloud and multi-tenant settings where allocation rules materially affect results. It also surfaces adoption and credibility risks, including limited SME accessibility and greenwashing potential where independent verification is weak, reinforcing the core gap between "being able to report" and "being able to optimise."

Limitation/Gap	Description	Example/Implication	Application To GreenCode
Fragmented Standards Landscape	Many overlapping standards, no unified baseline	Efforts remain "formative and fragmented"	GreenCode can align and integrate with SCI (ISO 21031) [220]to reduce fragmentation
Narrow or Specialized Scope	Certifications cover limited domains	Blue Angel initially only desktop	GreenCode can broaden applicability across mobile, cloud, AI
Lack of Legal/Market Drivers	No legal requirement for software efficiency	Sustainability de-prioritized unless voluntary	GreenCode can anticipate regulatory shifts (EU CSRD)
Measurement Complexity	Energy attribution is difficult	Tools immature; inconsistent results	GreenCode can focus on automated, robust measurement
Academic-Industry Gap	Research not adopted in practice	ASCRSM still niche	GreenCode can bridge by operationalizing research

Lack of SME Engagement	Most certification schemes are designed for large enterprises; SMEs and indie developers find them too complex or costly.	Small studios or open-source projects cannot realistically pursue Blue Angel or CSRD alignment.	GreenCode could provide a lightweight certification toolkit or automated reporting tailored for SMEs.
Risk of Greenwashing	Voluntary schemes without independent verification can lead to symbolic compliance rather than substantive improvement.	Large vendors "badge" products without real reductions.	GreenCode could provide evidence-backed metrics via automated measurement, reducing the risk of superficial claims.
Compliance action gap in green software standards	Existing standards primarily support the measurement and reporting of software-related emissions but provide limited guidance on how to achieve measurable reductions.	Organisations can be compliant with reporting requirements while making little or no progress in reducing energy use or carbon impact.	GreenCode can help bridge this gap by linking standardised metrics to concrete, evidence-based optimisation actions.
Conflicts between functional requirements and sustainability goals	Standards don't resolve real trade-offs in delivery contexts.	Sustainability is waived under performance/SLA pressure.	Provide trade-off patterns + decision records + evidence-based exceptions.
Multi-tenant attribution and shared-cost allocation	In shared cloud and platform environments, per-service energy/emissions attribution depends on allocation rules (idle sharing, contention, baseline assumptions) that materially affect results.	Two teams can report different "footprints" for the same service depending on whether they allocate idle energy by CPU time, memory reservation, request share, or fixed baselines.	GreenCode should define explicit allocation rules and confidence grading for multi-tenant settings, and link the resulting evidence to compliance reporting requirements described in the Standards and Certification section.

Non-Priorities

GreenCode should avoid the following items to remain focused on software-centred certification value. We de-prioritise duplicating mature hardware/facility schemes (EnergyStar, PUE, OCP), relying on offsets as a primary strategy, creating new ad-hoc metrics that increase fragmentation, and making non-software domains (embodied hardware, broader supply-chain topics) the core of a software certification effort. We also regard consumer-facing eco-label approaches and full regulatory alignment as premature for an early-stage platform, favouring instead interoperability with existing standards (notably SCI) while building practical evidence automation.

Area	Reason to De-prioritise	Implication
Hardware-only efficiency standards (Energy Star, PUE, OCP, EU Ecodesign)	Already mature; not the differentiator for GreenCode software certification.	Focus on software-level metrics and evidence.
Carbon offsets as the primary strategy	Offsets don't improve software efficiency and complicate assurance narratives.	Prioritise real reductions; offsets are out-of-scope for GreenCode deliverables.

Ad-Hoc Metrics	Fragmentation risk if inventing new metrics	Align with ISO 21031 (SCI)
Non-ICT sustainability domains (e-waste, supply chain, embodied hardware) as certification core	Data-intensive; outside software certification focus.	Acknowledge links; keep certification centred on software energy/carbon evidence.
Highly Experimental Academic Tactics	Some green architectural or ML tactics (e.g., EcoMLS adaptations) lack industrial proof and may be premature	GreenCode should avoid over-investing in unvalidated methods; instead, track them until mature
Non-software Sustainability Dimensions	Social sustainability or hardware lifecycle topics are important but outside a software certification focus	Keep scope narrowed to emissions and energy efficiency of software, while acknowledging links
Hardware Efficiency Standards	Already well-covered by existing programs (EnergyStar, OCP, EU Ecodesign).	GreenCode should focus on software-level efficiency, not duplicate established work.
Consumer-Facing Eco-Labels	High cost and limited credibility without regulatory backing.	Focus instead on developer and enterprise certification, where impact is larger and measurable.
Narrow Niche Certifications	Sector-specific schemes (e.g., for media streaming or games) may not generalize.	GreenCode should build a broadly applicable platform, while remaining interoperable with niches.
Full regulatory alignment (at this stage)	Regulation is in flux.	Track CSRD/SCI etc., but don’t over-invest in compliance tooling before core evidence automation is solid.

GreenCode: State of the Art Review

Summary and Future Opportunities

Looking forward, standards and certification schemes have the potential to act as powerful market mechanisms, particularly in procurement, regulation, and customer trust. However, the current standards landscape remains primarily focused on the measurement and reporting of software-related emissions, with limited support for translating compliance into concrete, system-level optimisation actions. Frameworks such as the Software Carbon Intensity (SCI) specification provide a structured methodology for assessing the carbon intensity of a software system and for comparing different configurations or successive iterations of the same system over time, rather than serving as a universal comparator across heterogeneous applications or deployment contexts.

As a result, a persistent gap remains between compliance and impact, where organisations may be able to calculate or disclose emissions without having clear, evidence-based pathways to reduce them. While existing initiatives and organisations have been effective in establishing shared principles and raising awareness, their emphasis on guidance, education, and voluntary adoption highlights the need for complementary tooling that embeds sustainability directly into software engineering and operational workflows.

GreenCode can play a complementary role by operationalising sustainability requirements at the level of software systems themselves. By linking architecture models, runtime behaviour, and empirical energy measurements to standardised reporting frameworks such as SCI, GreenCode enables certification to evolve from passive disclosure towards active, validated, and evidence-based optimisation.

Opportunities

GreenCode can turn the current standards landscape into an engine for real-world change presenting opportunities to (i) reduce fragmentation through standard consolidation and reference implementations, (ii) expand certification pathways beyond niche schemes, (iii) automate sustainability reporting alignment with ESG/regulatory pressures (e.g., CSRD), and (iv) shift certification from periodic audits to continuous improvement tooling embedded in CI/CD and operations. It also has the strategic opportunity to align product-level and organisation-level labels, connect software metrics into broader corporate frameworks, and use AI/automation to produce governance-ready evidence at scale.

Opportunity	Description	Example/Implication	Application To GreenCode
Standard Consolidation	Shape and unify emerging standards	ISO 21031 (SCI) could become universal metric	GreenCode can serve as reference implementation
Certification Expansion	Scale certification beyond niche	Blue Angel + potential new domains	GreenCode can support certification pathways
Integration into ESG	Mandatory sustainability reporting	EU CSRD (2024)	GreenCode can automate reports
Continuous Improvement Tooling	From static audits to CI/CD monitoring	Blue Angel requires annual remeasurement	GreenCode can embed runtime/DevOps tools

Organizational Label Alignment	Align with org-level schemes (e.g., Numérique Responsable).	Companies prefer org-wide vs product-only.	Ensure compatibility across product + org schemes.
Organizational Label Alignment	Growth of org-level certifications like France's Numérique Responsable	Companies may prefer org-wide over product-only certification	GreenCode can ensure compatibility with both software and organizational schemes
Cross-Sector Certification Convergence	Link software sustainability with other established certifications (ISO 14001, ESG audits).	Software sustainability included in corporate ESG reports.	GreenCode could bridge software metrics with broader corporate reporting frameworks.
AI and Automation in Certification	Using AI to dynamically assess compliance and generate sustainability reports.	Automated generation of CSRD-aligned disclosures.	GreenCode can be a compliance automation engine, saving companies time/cost.
Community Standards Influence	Shaping emerging standards in collaboration with GSF, IEEE, and national bodies.	GreenCode insights feed into ISO/IEC guidelines.	Opportunity to disseminate methods and findings into the standards pipeline.

GreenCode: State of the Art Review

Sectorial Review

This section outlines the state-of-the-art with respect to specific application domains and use cases.

SotA of Energy-efficient Computing for Automotive Systems

Contributing Partner(s): Kings College London, KAN Engineering, TWT

Background

Automotive systems represent a highly constrained and safety-critical computing environment in which energy efficiency has historically been driven by hardware and powertrain considerations rather than by software behaviour. However, the rapid transformation of vehicles into complex, software-defined systems—characterised by increasingly centralised electronic/electrical (E/E) architectures, high-performance compute platforms, and extensive sensing and connectivity—has significantly elevated the role of software in determining overall energy consumption. Modern vehicles now integrate dozens of electronic control units (ECUs) or domain controllers executing mixed-criticality workloads, ranging from real-time control functions to data-intensive advanced driver assistance systems (ADAS) and in-vehicle infotainment.

Energy efficiency in automotive computing is tightly coupled with stringent non-functional requirements, including functional safety (ISO 26262), real-time determinism, reliability, and thermal constraints. Unlike general-purpose or cloud computing environments, automotive software must often meet hard timing guarantees under worst-case operating conditions, limiting the applicability of aggressive dynamic power-management strategies commonly used elsewhere. As a result, energy optimisation has traditionally focused on static, design-time decisions such as hardware selection, task allocation, and scheduling policies, with limited visibility into fine-grained runtime energy behaviour attributable to software.

The shift toward software-defined vehicles, over-the-air (OTA) updates, and increasingly autonomous functionality has intensified the need for more systematic, software-aware energy optimisation approaches. Centralised compute platforms, heterogeneous processors (including CPUs, GPUs, and AI accelerators), and service-oriented software architectures introduce new opportunities for workload consolidation and adaptive resource management, while also increasing the risk of energy inefficiencies emerging from software design, integration, and evolution over a vehicle's lifetime. These characteristics place automotive systems at the intersection of embedded systems engineering, real-time computing, and energy-aware software design.

From a green software perspective, the automotive domain exemplifies many of the challenges that GreenCode seeks to address: long-lived, safety-critical software systems operating under strict performance and reliability constraints, where architectural decisions, runtime behaviour, and energy consumption are tightly coupled. Addressing energy efficiency in this context therefore requires holistic approaches that integrate software architecture, runtime execution, and empirical energy measurement, setting the stage for the state-of-the-art techniques and research directions discussed in the following sections.

GreenCode: State of the Art Review

Current State of the Art

The current state of the art in energy-efficient computing for automotive systems is characterised by a strong emphasis on conservative, design-time optimisation combined with tightly controlled runtime behaviour, reflecting the safety-critical and real-time nature of the domain. In production automotive environments, energy efficiency is typically addressed through architectural decisions, task allocation, and scheduling strategies implemented within established platforms such as AUTOSAR Classic and Adaptive, running on automotive-grade real-time operating systems and increasingly heterogeneous system-on-chip (SoC) platforms used for central vehicle compute and ADAS. Industry practice prioritises predictability, certification, and worst-case guarantees, with energy optimisation techniques carefully validated to ensure they do not compromise functional safety, timing constraints, or system reliability. As a result, the automotive SotA is defined less by aggressive dynamic optimisation and more by proven methodologies for consolidation, workload placement, and lifecycle-aware software evolution, informed by empirical measurement and extensive integration testing across long-lived vehicle platforms.

Quantifying Software-Level Energy Demands

Autonomous vehicles (AVs) are a rapidly developing class of intelligent transportation systems (ITSs) being deployed in selected cities. Connected vehicles continuously collect and transmit various types of data, such as speed, real-time location, acceleration, state of charge in electric vehicles (EVs), and diagnostics, from one vehicle to another. These data are essential for analyzing vehicle performance and road conditions, helping the drivers to make better decisions and managing the traffic conditions[235]. AVs hold significant potential to enhance road safety by reducing accidents caused by human error. They can also improve traffic efficiency and productivity, while minimizing environmental impacts through optimized driving patterns that lower fuel consumption and support sustainability[236].

Achieving high levels of autonomy requires AVs to comprehensively understand their environment by processing extensive data from sensors like cameras, radars, and LiDARs through a software stack heavily reliant on machine learning (ML) algorithms[236]. The data generated from an average hour of driving can reach up to 20 Terabytes, depending on sensor specifications and data rates[237]. Processing this massive volume of data in milliseconds necessitates significant computational power onboard the vehicle[235] [236]. This reliance on high-performance computing units integrated inside the vehicle, which deploy AI models and algorithms, acts as the "brain" of the AV [237].

Many review papers focus on the environmental benefits of AVs, such as optimized driving strategies, improved route selection, fewer stops, and platooning. However, they often overlook the substantial energy demands imposed by essential hardware systems, including sensors, computers, and cameras[235]. The power required by these components increases the energy consumption[235]. Previous studies have also not thoroughly examined the data processing requirements in these vehicles[235]. This computing power required onboard AVs has the potential to yield emissions comparable to those of all data centres today, based on a scenario with one billion AVs driving one hour per day with an average computer power of 0.84 kW[238]. Characterizing the carbon emissions from computing onboard AVs is a relatively new area of research[238].

The energy consumption of computing onboard AVs is part of the broader environmental footprint of digitization and Information and Communication Technologies (ICT) (Sudhakar, Sze and Karaman, 2022) [239]. While the final energy consumption and associated carbon footprint of ICT are relatively well-studied, other environmental factors and certain phases of the equipment life cycle (like raw material extraction, manufacturing, maintenance, and end-of-life) are less understood[239]. New connected equipment and services, including edge computing relevant to AVs, add complexity to these assessments[239].

Methods to Optimise Software Energy Demands

The increasing demand for higher autonomy levels, particularly Level 5 self-driving, requires enabling AI at the edge devices within the vehicle in an energy-efficient manner[237]. Challenges exist in applying and integrating technological innovations, despite advancements in sensor technologies, wireless communications, computing, and AI/ML algorithms, to achieve energy efficiency in autonomous driving services[237].

Approximation techniques and energy-efficient methods are being explored for autonomous driving services[237]. This includes communication-efficient approaches and software approximation techniques, such as low-rank approximation, pruning, quantization, and sparsification, which aim to reduce the parameters of statistical models for inference[237]. Energy-efficient deployment of AI applications on resource-constrained devices can also utilize allocation schemes, heterogeneity-aware mechanisms, and federated learning[237].

Edge AI approaches and vehicular frameworks focus on energy efficiency[237]. Datasets play a crucial role in developing machine/deep learning-based autonomous driving services and tasks[237]. Various datasets have been made available, often categorized by sensors used and potential driving applications, including commonly used ones like KITTI, Cityscapes, and PASCAL VOC [237]. Frameworks for autonomous driving are also being developed, categorized as driving task/assist oriented, independent application/service oriented, or compute-communication oriented [237].

Research trends in autonomous driving have shown primary focus areas such as perception (specifically object detection and segmentation), SLAM (Simultaneous Localization and Mapping), and vehicular communication[237]. While energy-efficient approaches are gradually increasing in popularity as research topics, the number of publications on these methods is still relatively less compared to other subjects[237].

Integrating Sustainable Practices into Vehicle Development Workflows

However, research gaps and open problems remain, such as data management and processing techniques on edge devices, categorizing autonomous driving use-cases by real-time and energy implications, and hierarchically categorizing autonomous driving tasks and their energy impacts [237]. A significant challenge is handling the high-volume data from vehicle sensors, especially during collaborative inference, as current edge frameworks may not propose modules for offloading or aggregating sensed data at the edge, which can lead to data floods and repetitive computation[237].

The environmental effects of digitization are categorized into direct effects (from the life cycle of equipment) and indirect effects (positive or negative outcomes from digitizing other sectors)[239]. Indirect effects can include efficiency gains or rebound effects[239]. A rebound effect occurs when a positive impact (like increased efficiency) leads to increased demand or consumption[239]. Different types of rebound effects exist, including direct, backfire, indirect, time, and macro-level rebounds[239]. Macro-level rebounds are often ignored in the literature[239]. Critically, assessments of indirect effects are complex because they are influenced by external factors like regulations, prices, and socio-cultural contexts, making extrapolations highly uncertain[239]. Existing literature tends to focus on positive indirect effects while potentially ignoring or minimizing negative ones[239]. Addressing these gaps requires further research and improved methodologies for assessing the complex interplay of direct and indirect environmental impacts of digitization, including the computational demands of autonomous vehicles[239].

Limitations and Gaps

In current automotive software engineering practice, formal analysis and assurance activities are heavily centred on safety and risk assessment, supported by well-established standards, processes, and artefacts. While this focus is both necessary and appropriate for safety-critical systems, it leaves sustainability largely outside the scope of systematic assessment. Although some of the data and models used in safety, timing, and performance analysis could potentially be leveraged to reason about energy consumption and environmental impact, existing assessment frameworks do not treat sustainability as a first-class concern. This raises an open question as to whether sustainability considerations can be integrated into current automotive assessment and assurance processes, or whether new, complementary frameworks are required to evaluate and evidence software sustainability alongside safety and risk.

Limitation / Gap	Description	Implication for Energy Efficiency	Relevance to GreenCode
Energy not a first-class optimisation objective	Automotive software optimisation remains dominated by safety, timing, and reliability requirements, with energy efficiency treated as a secondary or derived concern.	Energy inefficiencies persist when trade-offs are resolved conservatively in favour of worst-case guarantees rather than energy-optimal behaviour.	Highlights the need for approaches that integrate energy awareness without compromising safety or determinism.
Predominance of static, design-time optimisation	Most energy-related decisions are made during architecture design, task mapping, and scheduling, with limited adaptation at runtime.	Inability to respond to changing workloads or operating conditions leads to sub-optimal energy use over the vehicle lifecycle.	Motivates lifecycle-aware analysis and feedback mechanisms aligned with GreenCode principles.
Limited runtime energy visibility at software level	Fine-grained attribution of energy consumption to software components, tasks, or	Developers lack actionable feedback linking software behaviour to energy impact.	Reinforces the need for software-level observability and measurement integration.

	services is rarely available in production vehicles.		
Constrained applicability of dynamic power management	Aggressive DVFS, task migration, or dynamic consolidation techniques are often restricted by real-time and safety constraints.	Many energy-saving techniques used in cloud or consumer systems cannot be safely applied in automotive contexts.	Underscores the importance of domain-specific energy optimisation strategies.
Heterogeneous compute under-exploited for energy efficiency	Increasing use of CPUs, GPUs, and AI accelerators is driven by functionality rather than energy-optimal workload placement.	Workloads may execute on sub-optimal processing units from an energy perspective.	Points to opportunities for architecture- and workload-aware optimisation across heterogeneous platforms.
Fragmented toolchains and limited cross-layer integration	Energy analysis, timing analysis, safety verification, and software architecture are handled using largely separate tools and workflows.	Energy considerations are difficult to propagate across development stages and organisational boundaries.	Aligns with GreenCode's goal of unifying architecture, runtime behaviour, and sustainability analysis.
Limited support for long-term software evolution and OTA effects	Energy impacts of over-the-air updates and incremental software growth are rarely assessed systematically.	Energy consumption may increase over time as functionality accumulates.	Highlights the need for continuous assessment across the software lifecycle.
Lack of automotive-specific energy benchmarks	There are few widely accepted benchmarks for evaluating software energy efficiency under automotive workloads and constraints.	Comparability across platforms and solutions remains limited.	Supports GreenCode's emphasis on context-aware, system-specific evaluation rather than global benchmarking.

Summary and Future Opportunities

While autonomous driving offers numerous potential benefits, the energy consumption and environmental impact of the extensive computing and data processing required onboard AVs present significant challenges that need further research, particularly in developing energy-efficient hardware and software solutions and better understanding the broader environmental implications within the context of digitization.

Despite growing awareness of the energy and environmental implications of onboard computing in Electric / Autonomous vehicles, there remains a lack of systematic tools and methodologies to quantify and reduce the software-level energy footprint of ADAS/AD/EV applications. Current frameworks largely neglect the role of software optimisation and green coding practices in mitigating energy use at the algorithmic, subsystem, system and vehicle levels.

To address this gap, our work within GreenCode introduces a novel approach that integrates energy profiling, sustainability benchmarking, and software optimisation directly into the ADAS/AD/EV development workflow. By enabling fine-grained visibility into the energy consumption of AI models and software components, our contribution empowers developers and system architects to make

informed decisions that align with both performance and environmental goals. This represents a critical step toward embedding environmental sustainability into the core of ADAS/AD/EV systems and products.

Opportunities

The table below summarises the key opportunity areas for advancing energy-efficient computing in automotive systems, focusing on interventions that are compatible with safety, real-time constraints, and heterogeneous in-vehicle compute. It highlights where research and tooling can move from isolated optimisations to repeatable, engineering-workflow-friendly practices, spanning measurement and attribution, design-time trade-offs, and bounded runtime adaptation, alongside the enablers needed to operationalise these improvements at scale.

Opportunity Area	Description	Expected Impact on Energy Efficiency	Alignment with GreenCode
Energy as a first-class design objective	Integrating energy efficiency explicitly into automotive software architecture, scheduling, and trade-off analysis alongside safety and timing constraints.	Enables systematic reduction of energy consumption without undermining functional safety or determinism.	Directly supports GreenCode's aim to embed sustainability into core software engineering decisions.
Cross-layer energy observability	Development of tooling that links software components, tasks, and services to runtime energy measurements across OS, middleware, and hardware layers.	Provides actionable feedback to developers, enabling targeted optimisation and regression detection.	Aligns with GreenCode's focus on architecture–runtime–measurement integration.
Safety-aware runtime adaptation	Exploration of constrained, certifiable runtime adaptation techniques (e.g., bounded DVFS, mode-based scheduling) compatible with automotive safety requirements.	Allows limited but effective runtime energy optimisation under controlled conditions.	Supports GreenCode's principle of domain-aware, evidence-based optimisation.
Energy-aware workload placement on heterogeneous platforms	Systematic mapping of workloads to CPUs, GPUs, and AI accelerators based on both functional and energy characteristics.	Improves utilisation of heterogeneous compute resources and reduces unnecessary energy expenditure.	Reinforces GreenCode's system-level optimisation perspective.
Lifecycle-aware energy assessment for OTA updates	Evaluating the cumulative energy impact of over-the-air updates and incremental feature additions across the vehicle lifetime.	Prevents gradual energy degradation as software evolves post-deployment.	Strongly aligned with GreenCode's lifecycle and continuous assessment goals.
Integrated toolchains for energy, safety, and timing analysis	Closer integration of energy analysis with existing automotive toolchains for safety verification, timing analysis, and architecture modelling.	Reduces friction in adopting energy-aware practices within established workflows.	Advances GreenCode's vision of unified, developer-facing sustainability tooling.

Automotive-specific energy benchmarks and reference workloads	Development of representative workloads and evaluation methodologies reflecting automotive constraints and use cases.	Enables more meaningful comparison and validation of energy optimisation techniques.	Complements GreenCode's emphasis on context-aware evaluation over generic benchmarks.
Sustainability-aware assessment alongside safety and risk	Exploration of approaches to assess software sustainability in parallel with established automotive safety and risk assessment processes, either by extending existing frameworks (e.g. safety cases, V-model artefacts) or by introducing complementary sustainability assessment mechanisms.	Enables systematic consideration of energy and environmental impacts during design and validation, rather than relying on ad hoc or post hoc optimisation.	Directly aligned with GreenCode's goal of embedding sustainability into core software engineering and assurance practices.

GreenCode: State of the Art Review

SotA of Evaluating the Energy Efficiency of Embedded Platforms in Aerospace

Contributing Partner(s): ZAL

Background

The energy efficiency of embedded systems plays an increasingly important role in aviation. With the shift away from fossil fuels toward hydrogen or battery-electric systems, electrical energy consumption demands will also increase. This can affect the power consumption of electrical components in drones and other aerospace applications as much as it will commercial aircraft.

Optimizing software on embedded platforms (e.g., microcontrollers) in such a way that energy can be saved while maintaining the same system functionality is an important goal for the sector, enabling existing systems to be more efficient while also unlocking the potential for ever more complex software or even AI to operate in significantly energy constrained systems.

In principle, this can be achieved either by optimizing algorithms or by using dedicated hardware resources, but to quantify and understand the energy consumption of software as accurately as possible, appropriate measurements are necessary. Embedded platforms provide good access to hardware and power supplies and can be well controlled in terms of software and operating systems.

Current State of the Art

A state-of-the-art review was conducted to examine which methods and technologies have already been used to carry out similar measurements. The analysis of the state of the art is primarily intended to help optimize our own approach to measuring electrical power consumption or energy usage in these contexts.

The following section describes selected scientific studies evaluating the efficiency of different software components or algorithms. These and other works serve as guidance for the demonstrator to be developed within the project for automated energy consumption measurement.

ALEA: Fine-Grained Energy Hotspot Profiling for Embedded Software

The paper by Mukhanov et al.[240] introduces ALEA (Abstraction-Level Energy Accounting), a lightweight, statistical profiler for fine-grained energy consumption estimation at the basic block level of software. The aim is to help developers understand where energy is consumed inside programs and how to optimize code for energy efficiency beyond just execution time.

ALEA captures energy metrics at basic block level (a basic block is a straight-line code sequence with no branches). It uses random sampling to minimize performance overhead (<3.5%) compared to instrumentation-based profilers. It also uses general-purpose performance counters and external power measurement (does not require architectural modification). A linear regression model was build between performance counter values (e.g., instructions retired, cache misses) and energy consumption.

The runtime state was sampled as follows: At fixed intervals (e.g., every 10 ms), ALEA captures the currently executing basic block. It simultaneously logs performance counters and system-level power readings. It maps observed execution metrics to energy samples to learn the energy cost per block (regression analysis). The output parameters of ALEA are "Energy per basic block" (joules) and "Relative energy distribution over the codebase".

Two different case studies were evaluated, one of them was k-means clustering. ALEA showed that a small number of basic blocks consumed a disproportionate amount of energy (20% of execution time but 40% of energy). Developers used this insight to refactor memory access and loop bounds, reducing energy without affecting accuracy. Another case study involved the POP Ocean Model Kernel. ALEA identified energy-inefficient basic blocks in numerical loops involving floating-point operations. Using insights from ALEA, the developers applied loop unrolling and data layout transformations. It achieved up to 18% reduction in energy consumption with negligible performance overhead.

One key finding was that energy does not equal time: The profiler proves that optimizing for runtime does not always lead to optimal energy use. Some blocks consume more energy due to memory intensity or I/O stalls, not just execution cycles. Also, hotspots Are Not Always Obvious: Even in high-performance code, energy hotspots may reside in less obvious places (e.g., memory prefetch instructions or branch misprediction penalties).

Energy-Aware Runtime Algorithm Selection for Power-Constrained Embedded Systems

The study by Bunse et al.[241] explores how software-level choice of sorting algorithms impacts energy consumption, particularly in battery-powered, mobile and embedded systems. It proposes a runtime optimizer that dynamically selects between algorithm implementations to balance energy use versus performance.

For their evaluation, an AVR microcontroller (ATmega128) with external SRAM was used, connected over Bluetooth (BlueSmirf modules), used in a communication node that receives datasets, sorts them, and transmits results.

Different sorting algorithms were implemented: Bubble Sort, Insertion Sort, Quick Sort (iterative and recursive), Merge Sort (iterative and recursive), Heap Sort, Shell Sort, Selection Sort.

The energy consumption was estimated via battery voltage drop over time, used as an indirect indicator of system energy usage. Trend functions were empirically derived mapping input array size to estimated energy for 1,000 runs.

Energy vs. Performance Trade-offs: Insertion Sort is the most energy-efficient for small-to-moderate dataset sizes, despite poorer time complexity ($O(n^2)$). It is memory-efficient and in-place, thus saving power. Quicksort, although faster, consumes more energy due to increased memory and CPU utilization. For larger datasets where time is prioritized, Quicksort becomes preferable, especially when the energy difference falls within a small threshold[242]

Adaptive Algorithm Switching: A dynamic system that chooses sorting strategy at runtime based on data size and energy/performance trade-offs. Adaptive behavior matched or improved energy savings compared to energy-optimal non-adaptive variant, while maintaining better throughput than

purely energy-optimized often slow algorithms (Insertion Sort). Non-adaptive Quicksort provides high throughput but drains battery quickly; non-adaptive Insertion Sort extends battery life but limits throughput; the adaptive system strikes a middle ground based on policy thresholds.

In conclusion, an energy-aware strategy choice at runtime can outperform fixed implementation policies—vital for embedded/mobile systems where uptime and responsiveness matter. Right-sizing algorithm based on input size and desired trade-off yields significant gains in battery life without compromising performance unduly. Memory footprint and data movement matter more to energy use than raw algorithmic complexity; in-place, low-overhead algorithms can outperform higher-complexity ones under energy constraints. This is echoed in database join studies as well.

Empirical modeling with power measurement (even via voltage tracking) can deliver actionable cost functions for embedded systems. This approach can be extended to sorting or algorithm variants on platforms like the Intel NUC: derive energy models under controlled measurements, then dynamically select the implementation suited to data size and performance-utility needs. Highlights that pure performance optimization does not equal energy efficiency, underscoring the importance of custom energy profiling at the implementation level.

Bridging Research and Practice in Energy-Efficient Software Engineering

The article by Wysocki, Miciuła, and Plecka (2025)[243] presents a systematic literature review on methods for improving the energy efficiency of software and examines their actual application in industrial practice. The motivation for the study stems from the growing importance of sustainable software development, as software now represents a significant part of global energy use, both directly through computational workloads and indirectly via the large-scale infrastructures that execute it. The central research questions addressed by the authors were threefold: (1) what methods and practices for reducing the energy consumption of software have been reported in the literature, (2) what the current level of adoption of these methods in practice is, and (3) what barriers and opportunities exist for promoting more energy-efficient software engineering.

The review identifies and categorizes existing methods into five broad groups. The first group, resource management and parallelization optimization, focuses on controlling how computing resources are allocated and scheduled. Well-established techniques such as Dynamic Voltage and Frequency Scaling (DVFS) and thread shuffling are highlighted. These approaches can lead to substantial energy reductions, with some studies reporting savings of up to 56% when applied effectively. The HERMES framework, which combines voltage and frequency adjustment with workload tuning, was also discussed, though it delivered more modest improvements of 3–4%.

The second group of methods involves communication optimization, where the main goal is to reduce the energy overhead of data transfers and network interactions. This is particularly relevant in distributed and mobile computing environments where communication often consumes more power than computation. Although not elaborated in detail, the review emphasizes that optimizing communication patterns can yield significant benefits in systems where energy efficiency is constrained by I/O operations.

The third category, automatic energy tuning, relates to adaptive approaches that adjust software parameters at runtime to balance energy consumption and performance. These techniques rely on dynamic monitoring and intelligent control strategies to optimize energy efficiency under varying workloads. Such methods are especially promising in heterogeneous and cloud environments, where workloads can fluctuate significantly over time.

A particularly impactful category of techniques is approximation methods. Approximate computing reduces energy usage by deliberately relaxing precision or accuracy where exact results are not critical. Examples include EnerJ, a Java-based framework for approximate data types, which has demonstrated energy reductions between 10% and 50%. Other strategies, such as using single-precision instead of double-precision arithmetic, hybrid precision computations, or memoization (reusing previously computed results), can achieve even greater savings, with memoization showing reductions of up to 74% in certain cases. The review highlights that approximation offers some of the most significant opportunities for improving energy efficiency, if software correctness and quality requirements are not compromised.

The fifth and final group comprises programming practices, refactoring, and design patterns. Here the review emphasizes that good software engineering practices, such as minimizing I/O operations, preferring bulk operations over iterative calls, selecting energy-efficient data structures, applying loop transformations, and adopting the “race to idle” strategy, can collectively lower energy consumption. Refactoring tools and practices also contribute: automatic refactoring for mobile platforms led to energy reductions of around 5%, while broader refactoring efforts were found to offer further, though more context-dependent, improvements.

In addition to categorizing methods, the review sheds light on the current state of adoption in industry. Surveys of software developers and architects indicate that energy efficiency remains a low priority in practice. Only 18% of developers reported that they considered energy consumption during development, and just 10% had measured software energy use. Similarly, only 17% of software architects have addressed energy-related challenges in recent years. The main barriers identified include a general lack of awareness, limited client demand for energy-efficient solutions, insufficient tool support, and organizational priorities that focus more on performance, cost, and time-to-market. A follow-up survey conducted in 2023 by the authors reinforced these findings: while developers often lacked formal knowledge of energy-saving techniques, they showed considerable interest and willingness to adopt them if supported by appropriate incentives and education.

Finally, the review outlines several opportunities for improvement. Key enablers include increasing developer awareness through education and training, providing better tooling and profiling support to make energy consumption visible, and establishing incentives or client-driven requirements for energy-efficient software. The authors propose a conceptual framework that links these enablers with technical methods, highlighting synergies between academic research and industrial practice.

In conclusion, the systematic review by Wysocki et al. provides a comprehensive overview of the state of research and practice in software energy efficiency. It identifies promising technical

approaches, especially approximation techniques and resource management strategies—while highlighting the gap between research findings and their application in real-world software development. Addressing the awareness and adoption barriers identified in this study could significantly enhance the role of software in achieving broader sustainability goals.

Limitations and Gaps

The Limitations and Gaps table below outlines the main challenges that currently limit robust evaluation of energy efficiency on embedded platforms. It captures common sources of measurement uncertainty and non-comparability (differences in instrumentation, operating conditions, and workload design), the difficulty of attributing platform-level energy signals to specific software components, and the practical constraints of embedded environments (restricted access to counters, low overhead budgets, and high device heterogeneity). Together, these issues explain why results are often hard to reproduce or generalise, and why evaluation methods that work in controlled research settings frequently struggle to scale to real product development and certification contexts.

Limitation / Gap	Description	Implication for Energy Evaluation	Relevance to GreenCode
Energy evaluation dominated by hardware-level metrics	Energy efficiency is commonly assessed at the level of boards, chips, or power rails, with limited linkage to software execution.	Software-induced energy inefficiencies remain hidden, limiting actionable optimisation.	Highlights the need for software-centric attribution and analysis.
Limited software-level energy attribution	Fine-grained attribution of energy consumption to tasks, functions, or components is rarely available on embedded platforms.	Developers lack insight into how specific software design choices affect energy use.	Reinforces GreenCode's focus on linking software behaviour to energy outcomes.
Fragmented evaluation tooling and workflows	Power measurement, performance profiling, and software analysis are typically performed using separate tools and workflows.	Energy evaluation is costly, manual, and difficult to integrate into development cycles.	Aligns with GreenCode's goal of unifying analysis across layers and tools.
Lack of standardised evaluation methodologies	There is no widely accepted protocol for evaluating and comparing embedded platform energy efficiency across vendors or domains.	Results are difficult to reproduce or compare, limiting knowledge transfer.	Motivates GreenCode's emphasis on repeatable, evidence-based assessment.
Reliance on synthetic benchmarks	Many evaluations use microbenchmarks or artificial workloads that do not reflect real embedded applications.	Energy behaviour under real operating conditions is poorly understood.	Supports GreenCode's focus on runtime-aware and workload-representative analysis.

Limited consideration of RTOS and middleware effects	The energy impact of RTOS scheduling policies, middleware stacks, and inter-task communication is often under-analysed.	Significant energy costs may arise from system software rather than application logic.	Highlights an opportunity for runtime- and orchestration-aware evaluation.
Challenges evaluating heterogeneous embedded platforms	Increasing use of DSPs, GPUs, and accelerators complicates energy evaluation and comparison.	Workloads may be placed on sub-optimal processing units from an energy perspective.	Reinforces the need for system-level, heterogeneous-aware analysis.
Poor support for lifecycle and firmware evolution analysis	Energy impacts of firmware updates, configuration changes, and feature growth are rarely assessed over time.	Long-lived devices may become progressively less energy efficient.	Strongly aligned with GreenCode's lifecycle-aware sustainability objectives.
Limited traceability for certification or compliance	Energy evaluation results are often not produced in a form suitable for audit, certification, or procurement.	Limits the use of energy evidence beyond internal optimisation.	Supports GreenCode's role in generating traceable, auditable evidence.

Summary and Future Opportunities

Current systems are limited both by the accuracy of measurements and by a lack of automation. Our approach is therefore twofold: first, to enable measurements with the highest possible precision (i.e., high temporal resolution as well as high resolution of the recorded physical parameters, primarily the electric current). In addition, the entire measurement process is to be fully automated.

The underlying concept is that the user uploads the code to be tested into a designated repository (e.g., GitHub or GitLab), where the code is then automatically compiled for a specific target platform and executed on that platform. In parallel with the execution, the measurement of the electrical parameters is initiated. Upon completion of the measurement, the recorded data (time series) is likewise stored in a repository for subsequent analysis.

GreenCode: State of the Art Review

Opportunities

The table below summarises the main opportunities for improving how the energy efficiency of embedded platforms is evaluated in practice. It highlights directions for making measurement more accurate and comparable across heterogeneous devices and workloads, strengthening attribution from platform-level signals to software behaviour, and improving the repeatability of evaluation protocols so results can be trusted, reused, and applied within real engineering and certification workflows.

Opportunity Area	Description	Expected Impact on Energy Efficiency	Alignment with GreenCode
Software-level energy attribution on embedded platforms	Development of methods to attribute energy consumption to specific software components, tasks, or functions running on embedded platforms.	Enables targeted optimisation by linking software behaviour directly to energy use.	Directly supports GreenCode's goal of software-centric energy visibility.
Integrated hardware–software energy evaluation	Closer integration of hardware power models, on-chip sensors, and software execution models for holistic energy assessment.	Improves accuracy of energy evaluation and avoids isolated hardware-only or software-only analysis.	Aligns with GreenCode's cross-layer measurement philosophy.
Standardised evaluation methodologies for embedded energy efficiency	Establishment of common evaluation protocols, workloads, and metrics for embedded platforms across domains and vendors.	Improves reproducibility and comparability of energy efficiency claims.	Supports GreenCode's emphasis on evidence-based and repeatable assessment.
Runtime-aware energy evaluation under real workloads	Moving beyond synthetic benchmarks to evaluate energy consumption under representative, real-world embedded workloads.	Produces more realistic insights into operational energy behaviour.	Reinforces GreenCode's focus on runtime behaviour and empirical measurement.
Energy-aware evaluation of RTOS scheduling and middleware	Systematic assessment of how RTOS scheduling policies, middleware, and communication stacks influence energy consumption.	Enables optimisation of scheduling and communication choices for energy efficiency.	Complements GreenCode's interest in runtime and orchestration effects.
Evaluation support for heterogeneous embedded platforms	Improved methods to assess energy efficiency across heterogeneous embedded systems combining CPUs, DSPs, GPUs, and accelerators.	Enables energy-optimal workload placement and utilisation of specialised hardware.	Aligns with GreenCode's system-level optimisation perspective.
Lifecycle-aware evaluation of embedded software evolution	Assessing how firmware updates, configuration changes, and feature growth affect energy	Prevents gradual energy degradation in long-lived embedded systems.	Strongly aligned with GreenCode's lifecycle-aware sustainability goals.

	consumption over device lifetimes.		
Toolchain integration and developer-facing feedback	Integrating energy evaluation capabilities into existing embedded development toolchains and IDEs.	Lowers adoption barriers and encourages energy-aware development practices.	Supports GreenCode's aim to embed sustainability into everyday engineering workflows.
Certification- and compliance-ready evaluation evidence	Producing traceable, auditable energy evaluation results suitable for certification, procurement, or regulatory reporting.	Strengthens trust in energy efficiency claims for embedded products.	Positions GreenCode as a bridge between engineering evidence and sustainability governance.

GreenCode: State of the Art Review

SotA of Energy-efficient Computing for Web/SaaS Systems

Contributing Partner(s): Digital Tactics

Background

Modern web and Software-as-a-Service (SaaS) applications contribute significantly to global IT energy consumption and carbon emissions. Datacentres alone consumed an estimated 240-340 TWh in 2022 (about 1-1.3% of global electricity)[244], and the broader ICT sector (including networks and user devices) is responsible for roughly 1.5-4% of worldwide greenhouse-gas emissions[245]. This footprint is comparable to the aviation industry and is poised to grow as digital services expand[246].

With over half of Fortune 500 companies and many nations setting carbon-reduction targets by 2030, improving software energy efficiency has become an important aspect of corporate sustainability strategies. Web and SaaS systems, from enterprise applications to online services, are a key focus, since every user interaction (page load, API call, transaction) consumes energy across servers, networks, and devices. Enhancing the energy efficiency of software can therefore yield both environmental benefits and cost savings.

Traditional efforts to green IT often focused on hardware (improving server hardware efficiency, cooling, and using renewable power in datacentres), and indeed hyperscale cloud providers have made datacentre efficiency gains (e.g. improved PUE and 100% renewable energy commitments). However, software efficiency is emerging as an equally critical lever: as hardware improvements plateau, ever-growing software complexity can negate hardware gains[246]. Writing cleaner, more performant code and intelligent system design can dramatically cut the number of servers or CPU cycles needed for the same task, translating to lower energy use. For web/SaaS systems that scale to millions of users such as widely-used SaaS tools like Canva or Xero, or e-commerce platforms like eBay and shopify through to widely deployed website platforms like WordPress, even minor per-transaction savings can accumulate to substantial energy and carbon reductions. Reducing just 1 MB of data transfer in an online service handling 150 million orders per year could save on the order of 4 metric tons of CO_2 annually[247].

Energy-efficient computing for web and SaaS is about building applications that deliver the same functionality and user experience with fewer computational resources and less data movement. This not only lowers operational costs for providers (as energy bills and hardware requirements drop) but also helps organizations meet sustainability goals.

GreenCode: State of the Art Review

Current State of the Art

Today, software teams address energy efficiency through both engineering practices and specialized tools. Many optimizations align with existing goals of performance and cost reduction, since faster, leaner code inherently uses less energy. In this section we will discuss the key approaches defining the current state of the art.

Efficient Coding and Architecture

Thanks to education efforts by the Green Software Foundation and the Green Web Foundation amongst others, sustainability trained developers are increasingly factoring the energy impact of design choices into their work. This starts with selecting efficient algorithms and minimizing unnecessary computations and extends through to architectural decisions such as avoiding overly chatty microservices or excessive data duplication. Focusing on three areas, communication, computation, and user experience, can significantly cut software energy footprints.

Concretely, reducing data transfer (communication) by eliminating redundant payloads or merging services has big payoffs. Optimizing computation means trimming abstraction layers and avoiding bloated code, e.g. using efficient data structures, caching results, and removing features that are not valuable to users.

Rethinking user experience requirements can also save energy (for example, not everything needs ultra-low latency or real-time updates if a slightly slower update would allow more energy-efficient processing or server scheduling). These practices are increasingly recognized by industry as part of sustainable software design.

Choice of Programming Language and Framework

The stack used for a web backend or SaaS can greatly influence energy consumption. Recent empirical research confirms that different web frameworks and languages have varying efficiency. For instance, an experiment comparing popular backend frameworks (Django in Python, Laravel in PHP, Express in Node.js, and Spring Boot in Java) found that under high load, the Node.js and Java frameworks were far more energy-efficient than the PHP and Python ones[248]. Specifically, with large numbers of concurrent users, Express (Node) and Spring Boot handled requests with the lowest energy usage and fastest response times, whereas Django and Laravel consumed significantly more energy and had slower responses. The clear implication is that “heavier” frameworks or less optimized runtime environments can waste energy at scale.

Similarly, studies at the programming language level (e.g. Pereira et al.) have shown languages like C/C++ or Go tend to be more energy-efficient than languages like Python or PHP, in part due to differences in memory usage and execution speed[249]. While choosing a tech stack is usually driven by functionality, developer skill, or ecosystem, these results suggest that energy efficiency is an important attribute to consider, especially for large-scale SaaS deployments. It’s worth noting that efficient frameworks often correlate with better performance and hardware utilization, aligning sustainability with business performance needs.

GreenCode: State of the Art Review

Front-end Efficiency

Although the server side is a major focus, the front-end (client side) of web apps also impacts energy usage. Complex, heavy client-side scripts not only strain user devices (draining battery and drawing more power) but also cause more data transfer. Industry best practices for web sustainability echo long-standing web performance guidelines: use lightweight frameworks, optimize and lazy-load images, minimize JavaScript, and leverage static content where possible.

Sending fewer bytes and doing less work in the browser (e.g. avoiding overly feature-rich frameworks) can significantly reduce total energy. Tools like Google's PageSpeed Insights and sustainable web design checklists (e.g. The Green Web Foundation's guidelines) are now used to audit front-end bloat in terms of carbon impact, not just speed.

Cloud Infrastructure and Scaling Practices

The move to cloud-native architectures has enabled more dynamic and efficient use of computing resources for SaaS. Two notable trends here are containerization (Docker/Kubernetes) and serverless computing or "Function as a Service" (FaaS). Containers package applications in a more lightweight fashion than traditional virtual machines, allowing higher density of workloads per host. This improves utilization (fewer idle servers) and thus can reduce energy per application.

However, containers do introduce a modest overhead. Academic tests found Docker containers incur a slight increase in energy consumption compared to running the same workload on bare metal, largely due to I/O virtualization overhead [250]. In practice, this overhead is usually small relative to the gains from consolidation, but it highlights that container runtime efficiency (and avoiding unnecessary container layers) matters. Orchestration tools (Kubernetes, Docker Swarm) can aid efficiency by bin-packing workloads onto servers and shutting down unused instances.

Serverless computing pushes this further by abstracting servers entirely, code runs on-demand and scales to zero when not in use. This model inherently aims to eliminate idle capacity and maximize resource-use efficiency. Studies and industry reports indicate that well-architected serverless applications can significantly reduce energy consumption. For example, one 2023 empirical study showed serverless deployments reduced energy usage by up to 70% compared to equivalent always-on services [251].

The fine-grained scaling and pay-per-request model means you aren't powering resources that aren't actively doing work, a clear win for efficiency. Major cloud providers (AWS Lambda, Azure Functions, etc.) automatically handle multi-tenancy and resource optimization behind the scenes, benefiting from economies of scale. That said, serverless is not a silver bullet, it can have sustainability downsides if misused. Cold start overhead (spinning up containers for functions) and short-lived functions that don't fully utilize their allocated CPU time can lead to inefficiencies [251]. However, research into mitigating these issues such as IBM's EcoFaaS project show that smart scheduling can cut cluster energy use by up to 42% [252].

Many organizations are now exploring "sustainable serverless" design patterns: for instance, using caching to avoid repeated cold starts, and bundling small functions to reduce inter-function calls, all with the aim of maximizing the energy proportionality of the system (using resources only exactly as needed).

Tools for Measurement and Transparency

As developers have begun to wake up to the challenge of energy efficiency increasing numbers of tools to measure and track software energy usage at various levels are emerging. This is critical because "you can't improve what you don't measure." On the low-level end, developers and researchers use profiling tools such as Intel RAPL (for measuring CPU energy) and software like Perf, PowerStat, and PowerTop to measure the energy consumed by code running on a machine [248].

Academic projects like GreenScaler and JouleScope also help profile code energy usage for optimization. In the context of web apps, new tools integrate these measurements into development workflows. For example, GreenFrame (formerly Argos) is a tool that automates end-to-end testing of a web application (including front-end, backend, and database) inside instrumented Docker containers, and produces an energy/carbon report. GreenFrame's model, developed in collaboration with researchers at CNRS in France, converts metrics like CPU time, memory, and network I/O into an estimated energy consumption and even carbon emissions for a given user journey. This can be integrated into CI pipelines, meaning a development team can get feedback on how a code change affects the energy usage of the whole system, and even identify which component (frontend vs backend vs database) is the biggest contributor. Such tooling is analogous to performance testing or security testing in the pipeline, showing steps towards carbon-aware CI practices.

Similarly, the Green Web Foundation has introduced open APIs and libraries for estimating web carbon footprints such as CO2.js, an open-source JavaScript library that estimates the CO_2 emissions from data transfer and energy usage of websites and apps [253].

CO2.js uses established models (like the "Website Carbon" model and the Sustainable Web Design model) to translate bytes transferred and time used into carbon emissions, factoring in regional electricity carbon intensity. Developers can use it to display emissions per page view to users or to monitor their application's carbon budget [254]. Today CO2.js is becoming integrated into various website analytics and performance tools such as SiteSpeed.io's sustainability plugin and the dev tools interface of the Firefox web browser to make carbon metrics a first-class performance metric.

There are also web services like WebsiteCarbon.com and Ecograder that allow anyone to input a URL and get an estimate of energy and CO_2 per page view, raising awareness and encouraging optimization (e.g. compressing images to lower bytes and thus emissions). These tools, while based on models, have spurred competition among websites to be "low-carbon" just as they compete to be low-latency.

Industry Initiatives and Benchmarks

Leading tech companies and coalitions are actively defining standards and best practices. The Green Software Foundation (GSF), founded in 2021 by companies like Microsoft, Accenture, and ThoughtWorks, has published a Software Carbon Intensity (SCI) specification, essentially a standardized method to score an application's carbon efficiency which is now an accepted ISO standard[Error! Bookmark not defined.].

SCI considers the energy used by the software, the hardware it runs on, and the carbon intensity of the electricity (accounting for cloud region, etc.), yielding a single metric (a "carbon score per functional unit"). The idea is to enable structured comparison across different deployments or configurations of a software system, and to track improvements when code or deployment changes, provided that functional scope and underlying assumptions are clearly defined.

Alongside SCI, GSF curates a Patterns Catalogue of green software practices and offers training (a practitioner course) to spread knowledge. These efforts reflect an emphasis systematic approaches and education over ad-hoc optimisation, enabling sustainability to become a core software quality attribute (just like security or reliability).

Moving on from software specific measurement, cloud providers now provide carbon tracking tools for their customers. For example, AWS, Azure, and Google Cloud each have dashboards that show the emissions associated with a customer's cloud resource usage (often using regional data on energy mix). This is prompting SaaS companies to consider when and where their workloads run but is often criticised for inaccuracy and opaqueness around absolute direct emissions data).

Data like this across the software and the infrastructure has enabled the concept of carbon-aware computing to emerge. This involves scheduling compute jobs in regions or times when electricity is cleaner (lower carbon intensity), for instance, running batch jobs in a datacentre that currently has surplus renewable energy[255]. Microsoft's open-source Carbon Aware SDK[256] (released via GSF) and projects like Carbond [257,258] (an OS-level carbon-aware daemon) facilitate this by providing signals on grid carbon intensity that applications can use to adjust behaviour. While primarily used for batch processing and flexible workloads, such approaches could extend to SaaS, e.g. non-urgent data analytics might be delayed until renewable generation is high.

Finally, as initiatives like the Sustainable Web Design movement and organizations like the Green Web Foundation have started to mature they have begun to set informal benchmarks (e.g. target of <0.5 grams CO_2 per page view for a well-optimized site) and promote the concept of "carbon budgets" for websites (similar to performance budgets) to guide developers towards what "good" looks like. This is increasingly visible in the web industry's conferences and literature, showing that sustainability is slowly becoming a consideration on par with accessibility, data privacy and user experience in the minds of developers[259], though as with these peer topics, they can often be neglected by commercial development teams if the client does not mandate their consideration, or if legislation does not demand it.

Limitations and Gaps

The Limitations and Gaps table below summarises why energy-efficient web and SaaS engineering remains hard to operationalise at scale, despite growing awareness and guidance. It highlights persistent legacy inefficiencies in code and architecture, low developer awareness and incentives, and the ongoing difficulty of measuring and attributing energy/carbon across distributed, cloud-native systems (backend, network/transit, and client). It also captures the lack of standardised, comparable frameworks and the practical challenge of turning sustainability intent into repeatable engineering actions across heterogeneous stacks and deployment environments.

Limitation/Gap	Description	Example/Implication	Application To GreenCode
Legacy Inefficiencies	Many Web and SaaS systems still run on legacy code and infrastructure that were not designed with energy efficiency in mind, leading to wasteful resource usage. These older systems often consume more power than necessary to accomplish the same tasks.	A legacy monolithic application might continuously run unused services or use inefficient algorithms, drawing far more energy than a modern optimized microservices-based equivalent[260]. In fact, legacy IT infrastructure can account for over one-third of an organization's power consumption. These legacy inefficiencies persist due to a lack of incentives or knowledge to address them.	GreenCode will directly addresses legacy code optimization. Its pipeline will analyse existing codebases, detect energy-intensive code paths, suggest and auto-generate more efficient implementations, enabling legacy systems to gain sustainability benefits without total rewrites. Particularly important for enterprises that can't afford large-scale migrations.
Lack of Developer Awareness	Developers and architects often lack awareness or training regarding software energy efficiency[261], so it is rarely a priority during development. This knowledge gap means code may be written without considering energy impact, contributing to "software bloat" and higher power consumption over time.	Many teams focus on features and performance, assuming hardware improvements will cover inefficiencies; as a result, unnecessary computations or heavy libraries remain in production, wasting energy across millions of user interactions. Overreliance on ever-faster hardware has led to complacency about optimization.	GreenCode will provide insights and visibility by reporting on and generating training documentation from the changes it makes. Through this process it raises awareness, flags inefficient code patterns, and shows concrete savings in energy and CO_2, helping to educate teams through practical examples.
Difficulty Measuring Software Energy Use	It remains challenging to measure how much energy software consumes, especially in complex cloud or distributed environments. Breakdown of which components contribute most significantly to the applications' carbon footprint across backend, transit, and frontend is highly	Organizations may not realize which parts of their application are the most energy-hungry. For instance, gauging the energy impact of a new feature might require specialized hardware measurements or elaborate benchmarks, making it cumbersome to optimize and iterate on energy efficiency.	GreenCode needs accurate energy performance data to function and will generate this by automating stress testing and benchmarking while monitoring energy expenditure. This simplifies measurement for software owners and development teams by deferring the process to an independent assessment (similar to an

	context dependant. Without easy-to-use metrics or tools to isolate software power usage, teams struggle to identify inefficiencies or track improvements in energy use over the software lifecycle.		automated code review and penetration test in the security world). Developers will be able to receive clear, per-component estimates of energy and carbon impact for a codebase and any subsequent code change.
Lack of Standardized Frameworks	There are no universally adopted standards or frameworks for evaluating and guiding software energy efficiency within the SDLC. This lack of consensus makes it difficult to benchmark sustainability or certify software as "green," and developers have few official guidelines or metrics to follow during design.	With no common metric (analogous to an "Energy Star" rating for software), organizations rely on ad-hoc methods. Efforts like the *Software Carbon Intensity* (SCI) specification are emerging to fill this gap by defining a standard measure of software carbon emissions, but widespread industry uptake is still pending.	GreenCode will contribute to standards and certification by actively developing measurement methodologies and metrics that could become part of industry standards. By working on Green Software Certification it will provide organizations concrete ways to prove and communicate sustainable software practices.
Trade-offs with Other Software Qualities	Optimizing for energy efficiency can conflict with other priorities such as performance, reliability, security, privacy, accessibility or development speed. Teams often face trade-offs: for example, the fastest or most feature-rich solution might consume more power, forcing difficult choices between user experience (or service quality) and energy use.	Enabling aggressive energy-saving modes can slow response times or reduce reliability. Consequently, product owners might choose a design that uses more servers or high-performance settings to ensure speed and resilience, at the cost of higher energy consumption and carbon footprint.	GreenCode will quantify and help manage trade-offs. By measuring both energy and performance impacts, it will help developers make informed choices. Such as optimizations that maintain acceptable performance and user experience while reducing emissions.
Tooling Gaps	Energy profiling can be time-consuming. Mainstream developer tools and cloud platforms often lack built-in support for monitoring or optimizing the energy consumption of software. Without accessible profilers or energy analytics integrated into development workflows, energy use remains "out of sight, out of mind" during software design and testing.	Many developers have never used an energy profiler. For instance, a team might push inefficient code to production simply because their performance tools flag CPU and memory issues but provide no feedback on energy usage, causing them to miss opportunities to reduce power draw and operating costs. The time required to profile for energy use can further discourage developers from engaging with this task.	GreenCode aims to close the tooling gap through automated analysis, reporting, and optimization recommendations. This means sustainability checks can become as routine as security scanning, enabling green practices without requiring separate specialized tools.

Procurement	Current regulations also focus on reporting and transparency, they don't directly mandate efficiency improvements (though recent shifts in procurement approaches are starting to drive this kind of thinking as a way of supporting non-technical purchasers in addressing these requirements).	It has been recognised that bridging this gap will require greater demand from stakeholders (customers, regulators) for energy transparency, and embedding sustainability as a first-class concern in software engineering education and process, but measuring the deliverables are a challenge for non-expert procurement professionals.	GreenCode will help procurement officers and others responsible for purchasing software and IT solutions to make informed choices through its independent green software certification.
Containerised and serverless obfuscation	Because serverless abstracts the infrastructure, developers might be even less aware of resource usage (each function invocation seems ephemeral and cheap). This can lead to architectures that trigger multiple function calls where one could suffice, or transfer far more data between functions than needed, hidden inefficiencies that add up in energy expenditure.	There's a knowledge gap in industry on how to best optimize serverless for sustainability. Similarly with containers: organizations may spin up hundreds of microservices for modularity, but each carries overhead. The state-of-practice often prioritizes developer convenience and speed to market over energy optimization.	GreenCode may be able to recommend where serverless functions or microservices might be merged back together, following more traditional application architectures for lower financial and environmental cost and sustainability.
Rebound effects	It is hard to qualify the global system impact of optimisation. If we make a service 50% more efficient, the cost may drop, and more users might use it. Similarly, there might be increased scope for including new features within the same infrastructure and performance budget, potentially eroding the carbon savings.	Without careful planning, digital efficiency gains can lead to more consumption elsewhere[262]. These socio-technical factors form a gap in our understanding of true net impact.	GreenCode will help mitigate the impact of rebound effects that software and AI interventions can cause[263]. By ensuring software and AI climate interventions are themselves are optimised to be as green as possible the climate benefit of other interventions utilising them will be boosted.

GreenCode: State of the Art Review

Summary and Future Opportunities

It is possible to drastically cut software energy use but the current SotA falls short of a comprehensive solution. We lack broad adoption, precise measurement in many contexts, and fully developed processes to make green software routine. This is analogous to where security was decades ago, recognized as important, with some tools available, but not yet baked into every developer's workflow. GreenCode can begin to plug many of these gaps by creating an automated system that improves software performance with minimal impact on developers already busy workloads.

Opportunities

The Opportunities table below outlines where the field is most ready to move from principles to repeatable practice in web and SaaS systems. It focuses on embedding sustainability into existing delivery mechanisms, especially CI/CD "green regression" checks, verifiable measurement and reporting, and the emergence of certification/labels that create demand signals and accountability. It also highlights opportunities to scale impact through AI-assisted optimisation and workflow automation (particularly for legacy estates), enabling measurable reductions in energy/carbon without requiring teams to become sustainability specialists.

Opportunity	Description	Example/Implication	Application To GreenCode
CI/CD Pipeline Integration	Embed energy-efficiency checks and optimizations into Continuous Integration/Continuous Deployment pipelines. This involves automatically measuring the energy or carbon impact of code changes and alerting or gating builds when regressions occur. By integrating sustainability into DevOps, every code commit can be evaluated for its environmental impact alongside functionality and performance.	A CI pipeline might run energy consumption tests for each new feature, flagging a build if a change causes a noticeable increase in CPU or memory usage. Some leading organizations already employ such "green pipelines" that automatically optimize deployments for lower energy use and carbon footprint by default. This is a trend that could be expanded by GreenCode.	GreenCode's mission is to decarbonise the IT sector at scale, by integrating sustainability and energy efficiency into the SDLC. It could provide automated energy analysis tools that fit directly into CI/CD, delivering feedback on energy impacts alongside existing tests, making green software a routine part of modern DevOps workflows.
Green Certification & Labels	Establish certification programs or eco-labels to recognize software that meets defined sustainability criteria. A formal "green software" certification would signal that a Web/SaaS application has been optimized for low energy use and minimal carbon emissions. These	The Blue Angel eco-label in Germany is being extended to software products, certifying applications that achieve high resource and energy efficiency. In the future, a SaaS provider might proudly display a sustainability badge (akin to an Energy Star rating for	GreenCode could build on its measurement and optimization tools to underpin green certification programs by producing verifiable metrics and reports[264], enabling companies to earn sustainability

	labels can drive competition and provide incentives (and accountability) for companies to adopt greener development practices.	software) to demonstrate its application's low carbon footprint and attract environmentally conscious customers.	labels based on actual data rather than marketing claims[265].
AI-Assisted Code Optimization	Leverage AI to identify and fix energy inefficiencies in code. AI models can analyse large codebases to detect "code smells" or suboptimal patterns that lead to higher computational demand. By automatically suggesting more efficient algorithms or refactoring resource-heavy code paths, AI tools can help reduce software power usage without extensive manual effort.	An AI-driven analyser flags a frequently called function that uses an inefficient algorithm and suggests a more efficient approach, cutting that function's energy consumption up to 30%. Experiments have shown that AI can pinpoint hidden inefficiencies: Sogeti (part of Capgemini) demonstrated AI tools that detect and refactor code smells to reduce energy usage in data centre applications, and Groza et al. have shown similar in Android mobile applications[266]	GreenCode plans to create AI-driven optimization tools to drive sustainability and energy efficiency. By using AI to analyse patterns, detect energy code smells, and propose greener software, GreenCode can automate improvements that would otherwise require expert review—unlocking efficiency gains at scale.
Carbon-Aware Computing	Design software and scheduling systems to be "carbon-aware," i.e. to adapt their operation based on the real-time carbon intensity of the electricity grid. Carbon-aware computing involves shifting compute workloads to times or locations where the energy supply is cleaner (lower-carbon). By intelligently timing or geo-locating certain processes, Web/SaaS systems can reduce the carbon emissions associated with their energy consumption without necessarily reducing the amount of work done.	Microsoft, in partnership with others, developed a Carbon-Aware SDK that allows cloud applications to run tasks when and where the grid's carbon intensity is lowest. A SaaS platform might delay non-urgent batch jobs until nighttime when wind or solar energy is plentiful, or route requests to data centres in regions with greener energy, thereby cutting emissions significantly while maintaining service for users.	GreenCode can provide carbon signals for decision-making. By integrating regional carbon intensity data into its pipeline, it could enable apps to schedule workloads dynamically for lower emissions.
New Sustainability Assessment Frameworks	Develop and adopt robust frameworks for measuring and improving software sustainability. This includes standardized metrics and tools (e.g., a "software sustainability score") to evaluate an application's energy usage and carbon footprint in a consistent way. Such frameworks would provide architects and developers with clear targets and guidelines, making sustainability a	The Green Software Foundation's Software Carbon Intensity (SCI) specification offers a formula to calculate the carbon emissions per user operation of a software system. In the coming years, companies could incorporate SCI scores or similar metrics into their development cycle, for instance, requiring that a new feature stays within a certain energy budget, and	GreenCode could define and implement new frameworks. It is developing methodologies and metrics for measuring software energy and carbon impact that could feed into future ISO standards or sustainability frameworks. GreenCode can act as both a tool and thought leader

	first-class quality attribute in software design alongside performance, security, etc.	use these scores in reporting and internal benchmarking to drive continual energy efficiency improvements.	shaping how green software is assessed and certified.

GreenCode: State of the Art Review

SotA of Energy-efficient Computing for Media/Gaming

Contributing Partner(s): Digital Tactics

Background

Green software and sustainable IT are rising priorities for the gaming and media sectors. Over 3 billion people (one-third of humanity) play video games[267], and the electricity consumed by their consoles, PCs, and mobile devices exceeds that of some midsize countries[268]; In the United States alone, gaming-related energy use produces more annual greenhouse emissions than 5 million cars[269]; and globally video streaming contributes around 1% of global greenhouse gas emissions[270].

Designing and optimizing games, media production tools, and streaming services to minimize energy consumption and carbon emissions requires an understanding of the full production and consumption lifecycle from development (e.g. coding, rendering, build processes) to runtime (energy draw of games/apps on user devices, servers, and networks). Efficient code, optimized algorithms, and smarter use of hardware resources can directly translate to lower electricity use, reduced costs, reduced environmental impact and reduced heat generation which helps improve performance (by avoiding thermal throttling) a key ask of the gaming sector.

Hardware advances have made each generation of GPUs, consoles, and datacentres both more performant and energy efficient, but total power consumption continues to rise with increasing demand, both in terms of units of games and consoles sold per capita and in terms of the features required by gamers (super high-quality graphics, advanced gameplay experiences, responsive real time interactions, et cetera).

Similarly, UHD streaming video, and AI-driven media applications can strain both devices and cloud infrastructure. Without intervention, the “upgrade treadmill” (incessant pushes for higher fidelity and frequent hardware refreshes) lead to an unsustainable cycle of rising energy use and e-waste[271].

Awareness is growing that green software practices are vital to break this cycle. Industry groups and regulators have started to respond. The Sustainable Games Alliance (SGA) is developing a standard for measuring and reporting game industry emissions, aligned with frameworks like the GHG Protocol and new EU rules[272]. On the media side, studios and broadcasters are exploring virtual production and other tech-driven methods to cut down on physical resource use and travel. Additionally, sweeping regulations such as the EU’s Corporate Sustainability Reporting Directive (CSRD)[273] now legally require large companies, importantly including large digital companies, to report their carbon footprint and environmental impacts[274].

Similar disclosure rules are emerging in the UK (SDR)[275] and US (SEC climate disclosure rules, though these are being challenged and adapted at present)[276]. These policies create strong incentives for gaming and media companies to prioritize energy efficiency in software development and IT operations, as they must account for emissions across their whole value chain, including how users consume their digital products.

Improving software energy efficiency in gaming and media is very much a growing industry imperative promising win-win benefits: lower climate impact, compliance with regulations, cost savings, reduced waste and improved user experiences.

GreenCode: State of the Art Review

Current State of the Art

Gaming Industry: PCs, Consoles, Mobile, and Cloud Gaming

In gaming, energy efficiency efforts cover both hardware and software considerations. Player experience is key and most of gaming's carbon emissions come from the electricity consumed by their devices (consoles, PCs, and mobiles) during gameplay. This far outweighs the footprint of manufacturing or even the cloud servers for online services [277]. Front-end (user-side) optimisation is therefore a major topic for user experience as well as energy efficiency and we will focus our discussion here.

Efficient Console Hardware & Idle Power Management

The latest generation of consoles (PlayStation 5, Xbox Series X for example) are designed to be more energy-efficient per unit of performance than prior models. They also implement aggressive low-power modes. In 2023-2024, both Xbox and PlayStation released updates to dramatically cut standby/idle power, addressing the fact that consoles often sit idle or in menus. Xbox also introduced "carbon-aware" updates, scheduling game patch downloads for times when the local power grid is cleaner (higher renewable energy availability)[278].

These optimizations have significant impact [279,280]. Additionally, manufacturers have voluntarily agreed (through the EU's Games Console Voluntary Agreement [281]) to continue improving energy and resource efficiency, e.g. setting power caps for media playback, introducing auto-power-down features, and providing repairability to extend device life which is projected to save 41 TWh of electricity over the lifetime of the newest consoles.

Mobile and Handheld Gaming Optimization

Mobile games (smartphones, tablets, and hybrid devices like the Nintendo Switch) are inherently constrained by battery and thermal limits. This has forced developers to prioritize energy efficiency to avoid draining batteries or overheating. The Nintendo Switch uses under one-tenth[277] the power of a PlayStation or Xbox console while still delivering a compelling experience. Its success proved that gameplay innovation and fun do not always require maxing out hardware, inspiring rivals to consider efficiency. Mobile game developers routinely optimize frame rates, reduce background activity, and use energetically cheap graphics techniques to ensure smooth play without rapid battery drain. These practices represent a de facto state-of-art in mobile: energy efficiency is treated as a first-class performance metric because it directly impacts user experience (battery life).

Software Tweaks in Games to Save Energy

The patching of games with regular updates is a common practice and since 2023 major game studios have patched titles to eliminate unnecessary power draw without compromising player experience. For example:

- Halo Infinite developers discovered their game was fully rendering 3D scenes behind pause menus that players couldn't see, significantly wasting GPU cycles. They implemented a fix to dynamically lower graphics resolution when the game is paused, cutting that mode's energy

usage by 15% [282]. In its opening month alone this would have saved an estimated 38 GWh, the same monthly energy use as approximately 130, 000 European homes (estimate based on reported peak of 20M+ players and 211Wh draw of an Xbox series X during Halo Infinite gameplay, PC versions draw yet more energy).

- Fortnite introduced an update to reduce graphics quality in the matchmaking lobby (where players wait for a match to start). This change saves an estimated 73 GWh of electricity per year globally (roughly the combined annual energy use of 20, 800 European homes), with negligible impact on player experience [283].

- Elder Scrolls Online developers report that throttling graphics when players go idle or open menus led to a 5% drop in overall game energy consumption which for Xbox players alone means an approximately 1GWh/year reduction[284].

These cases have helped shine a light on the need for gaming studios to scrutinize every aspect of game loops (menus, loading screens, low-action scenes) for opportunities to scale down processing and save energy where user experience is not impacted. Microsoft's has since released tooling for Xbox developers to profile and measure their games' power usage in different scenarios [285] which enables them to identify energy hot-spots (like unneeded background rendering) for optimisation. However, sustainability still remains a secondary thought over user experience and speed of delivery today.

Game Engines and Optimization Technologies

Modern game engines (Unity, Unreal, etc.) are also under the microscope for energy efficiency. A comparison of Unity vs. Unreal Engine [286] found significant differences (up to 3-4× in certain physics or rendering scenarios) in energy consumption between engines. This implies that engine developers have room to improve and that choosing the right engine and settings can impact a game's carbon footprint. State-of-the-art engines now offer features that indirectly aid efficiency: for instance, dynamic resolution scaling and AI-based upscaling (like DLSS) allow maintaining high frame rates by rendering fewer pixels when possible, thereby reducing GPU workload and power draw. GPU manufacturers have also introduced driver-level features (e.g. NVIDIA's WhisperMode, AMD Chill) that cap frame rates or optimize graphics to save power on gaming laptops. These tools exemplify current best practice: giving users or developers the option to trade a bit of excess performance or fidelity for a large reduction in energy usage, especially useful on thermally limited or older devices.

Cloud Gaming

Cloud gaming services like NVIDIA GeForce Now, Xbox Cloud Gaming and similar defer the heavy rendering from user's machines to server GPUs in datacentres, with streaming video being sent to the user's device instead. While this shifts the gaming load from the user's device (potentially reducing their device's energy use), it adds substantial energy overhead in datacentres and network transit [287].

While energy and cooling are arguably better managed from a centralised perspective in datacentres, cloud gaming can consume far more total energy per hour than local device gaming, due to the inefficiencies of streaming and running a dedicated high-end GPU per stream. As such current efforts focus on the use of clean energy in datacentres, improving the efficiency of video encoding, and

sharing GPUs among multiple users when possible. In its current state GaaS is convenient but not energy-efficient, unless the datacentres involved are near-100% renewable-powered and highly optimized.

Gaming as a Service (GaaS) & Live Ops

While initiated almost 25 years ago through games like EverQuest, Counter Strike, and World of Warcraft, since 2015 a significant number of games have now migrated to a "games- as-a-service" model with continuous online features, updates, and live events being sold as add-ons to core gameplay.

These GaaS deployments follow similar sustainability challenges to other SaaS products where optimisation of the server-side architecture has been the focus to date. This means using scalable cloud infrastructure in "green" datacentres that can spin servers up and down vs. peak usage, developing efficient code to handle thousands of players per server with minimal overhead, and deploying servers closer to players (edge computing) to reduce data transfer loads and response time. Additionally, scheduling compute-intensive background tasks (like analytics or AI model training for games) for times of day when renewable energy is abundant is an emerging practice in line with carbon-aware software principles.

Media Production and Streaming Services

Film/TV production, animation, and video streaming are known to be hugely wasteful sectors from many different perspectives but see the implementation of digital technologies as we will discuss here as solutions to various issues in the production process. Here we will focus on the software and compute side of modern production processes.

Virtual Production in Film/TV

Virtual production involves the use of LED wall backdrops and real-time 3D engines (e.g. Unreal Engine) to create sets and environments and vs. traditional staging. It has rapidly gained traction in high-end productions because it offers creative flexibility, as well as cutting carbon emissions by up to 50% compared to traditional filming methods [288].

The savings come primarily from reducing the need for physical set construction, on-location shoots, and the associated travel of crew and transport of equipment. Instead of flying a crew to multiple global locations (with diesel generators, trucks, etc.) for example, a production can film locally with dynamic digital backgrounds. Similarly AI editing tools might in future allow a director to virtually "reshoot" a scene or fix footage without expensive re-shoots, thus avoiding yet further travel and logistics.

Major studios like Disney and Netflix are investing in virtual production stages not only for efficiency but also as a step toward their net- zero targets. Challenges here include careful consideration of the energy usage of large volumes of LED panels in LED walls and energy expensive rendering hardware.

As such producers are optimizing render engine settings, depth of point-clouds, reusing virtual sets, and leveraging off-peak or renewable energy for rendering workloads but there is a need for precise carbon accounting of virtual production (power, hardware manufacturing) and support needed to assist creatives in using these technologies in energy efficient ways.

AI and Machine Learning in Content Creation

AI is increasingly used in media and game content creation, from AI-assisted animation and VFX to generative music or dialogue, but it's a double-edged sword. While AI can streamline workflows (e.g. automating labour-intensive tasks like tweening animation frames or enhancing video upscaling, saving time and avoiding some resource usage, training and running large AI models for these tasks is an energy and water (for cooling) intensive process.

Media companies are trying to mitigate training impacts by using pre-trained models, leveraging specialised efficient and renewably powered AI infrastructure from cloud providers, and improving algorithmic efficiency in their software and production workflows. This can mean that for short videos today, AI content generation can have a (sometime significantly) reduced carbon footprint vs. traditional production methods, but as video and model complexity rises the impact of training and inference rapidly begins match and may even outweigh the climate benefit of this approach.

Rendering and Animation Efficiency

Animation studios and VFX houses operate large render farms, historically consuming huge amounts of power 24/7 to meet deadlines. For this reason current research and development focuses on rendering optimization and cloud rendering using green energy.

Studios like DreamWorks and Pixar have invested in optimizing their renderers, through better algorithms that, for example converge to a noise-free image faster (fewer sample computations), and scheduling non-urgent render jobs for times when renewable energy supply is high or when cloud computing prices (often tied to energy cost) are low.

Some production pipelines now use real-time game engines for final animation output, skipping energy-intensive offline rendering for certain content styles. Additionally, studios are exploring intelligent power management in render farms: turning off idling servers when not in use, and using modern CPU/GPU features that reduce power at low utilization.

All major animation companies track power usage in their datacentres and many purchase carbon offsets or direct renewable power for them. While the fundamental physics of rendering (high compute loads) hasn't changed, the efficiency per frame rendered is improving in line with software and hardware advancement. Another common strategy is to take advantage of cloud hyperscalers efficient facilities (very low PUE and high renewable mix) rather than operating older in-house render farms.

Streaming Services and Content Delivery Networks (CDNs)

Video streaming the way the modern world consumes its media boyed by services like Netflix,

YouTube, and Twitch, as well as on demand TV services provided by traditional broadcasters. Efficiency efforts instreaming media typically fall into one of two camps reducing the bitrate and data needed for streaming, or improving the efficiency of the streaming infrastructure (from CDN networks through to user devices).

Considering bitrate and data, the development of advanced video codecs like AV1 (as used by Netflix and YouTube) mean that streaming companies can deliver video quality at 30-50% lower bitrates than older alternatives like HEVC[289], significantly cutting the data load per stream. Lower data transfer directly saves energy in network transmission and reduces load on datacentre caches.

Newer codecs can be more computationally heavy to encode/decode, but increasingly have hardware acceleration support in modern (last 5 years) machines, and studies indicate that the energy savings from reduced data transfer outweigh the slight increase in CPU/GPU work on modern devices .

Netflix's engineering reports note that optimizing encoding recipes and deploying efficient codecs at scale is a priority for both quality and sustainability, as seen in gaming, their adaptive streaming algorithms aim to minimize bits delivered without disrupting user experience, reducing energy use across the delivery chain.

A common way of mitigating load, reducing latency, buffering and improving user experience in media delivery operations has been to use Content Delivery Networks (CDNs) which place video and other resources on content endpoints (servers) geographically closer to the user. This means that rather than every user pulling video from a distant datacentre (with multiple redundant hops), they obtain a local copy of the content faster and more directly. Today this process has the added value of reducing transit energy.

Modern CDNs use high-performance servers that can serve more streams per watt. They also turn off or idle caches during low demand. Technologies like multicast ABR (adaptive bitrate) are state-of-art for live streaming, sending one stream to many viewers simultaneously, smoothing out peak loads and avoiding the need to operated extra servers for popular live events [290].

Traditional streaming infrastructure had to be provisioned for peak load and ran continuously. Today, cloud-based streaming stacks championed but AWS, Google and others can scale dynamically, if viewership dips at 3 AM for example, cloud servers can scale down, saving energy, then scale up at peak evening hours. This elasticity, combined with running on energy-efficient cloud datacentres, means reduces energy waste.

However, even today, network equipment tends to draw power nearly constantly, whether fully utilized or not. And research focusses on designing smarter networking gear that can enter low- power states when traffic is light.

Despite these advances however, streaming media remains an energy expensive operation and while all the major streaming platforms Netflix, Google/YouTube, Amazon/Prime Video, etc. have corporate sustainability pledges consumption drives demand and carbon neutrality has to be managed through renewables energy and carbon offsets.

Furthermore, user devices are now recognized as the largest energy component of streaming's footprint [291] with modern Smart TVs drawing 100W+ and running for hours, meaning the industry is moving towards better TV energy standards and features like adaptive brightness detection to dim

screens and save power as well as providing specific energy saving modes. Some streaming apps also provide settings to cap resolution or frame rate to save data and thereby energy.

Limitations and Gaps

As with many other sectors the current state-of the-art reflects organisations addressing "low-hanging fruit", tackling obvious inefficiencies like rendering invisible frames or unoptimized standby modes, but many deeper or systemic inefficiencies remain. There is a need for more robust tools, cultural shifts in user and development priorities, and possibly structural changes (in business models or regulation) to overcome these limitations. Some specific shortcomings and unsolved challenges include:

Limitation/Gap	Description	Example/Implication	Application To GreenCode
Lack of Energy Profiling Tools	Developers have limited tools and standardized metrics to measure or optimize software energy usage, making it difficult to pinpoint inefficiencies[292].	For instance, unlike performance and memory profilers, energy profilers are rare, a game or streaming app team may not easily see how code changes impact power draw, leading to undetected waste in power consumption.	GreenCode can add value by developing user-friendly tools that integrate energy analysis into everyday developer environments, making energy impacts visible and actionable during coding and testing.
Legacy Software Inefficiencies	Many media and gaming systems rely on legacy code, content or engines not designed with energy efficiency in mind, resulting in wasteful resource usage. Reworking or patching old titles for efficiency is rarely prioritized (unless there is a business case like a remaster), leaving a long tail of popular software that continues running inefficiently.	Older game engines or streaming workflows may ignore modern low-power hardware features, causing higher energy draw. An outdated video processing module might use heavy CPU loops instead of efficient GPU acceleration, needlessly burning energy. Legacy game engines and code may have inefficient loops or assume unlimited resources, but they persist in current games or backward-compatibility modes.	GreenCode could analyse legacy codebases using AI and recommend and automate precise code-level refactoring to improve energy efficiency, helping companies modernize products without full rewrites.
Legacy Hardware Inefficiencies	Older streaming devices or set-top boxes might not support new efficient codecs, causing services to fall back to less efficient delivery for those.	This is counter to the general benefits of extending device longevity through optimisation, and means improvements may only apply to new releases or hardware, while significant energy drain continues from older systems still in use.	
User Devices	In media streaming, the energy use of user devices (TVs, consoles, phones) can be a big factor.	The software industry can optimize code and streams, but if a TV's backlight is a power hog, the overall footprint	GreenCode may be able to support beneficial forced obsolescence of

	However, manufacturers of TVs or monitors have historically prioritized display quality over energy, and while there are power regulations (e.g. EU energy labels for TVs), new features like 4K HDR and 120Hz screens can negate gains.	remains high. This is a cross-industry gap requiring collaboration between software, hardware, and standards bodies to ensure that as we roll out advanced media experiences, the devices are not unnecessarily inefficient.	inefficient hardware by providing factual data to regulators and suppliers of set-top-boxes and similar equipment.
Opaque Cloud Energy Usage	In cloud-based operations, there is often a lack of transparency into actual energy consumed by workloads, users can't easily measure power use of cloud resources they rely on[293]. Similarly, streaming companies relying on third-party ISPs and CDNs may not have full control over or data explaining their energy profiles.	A streaming service running on a cloud platform cannot see per-stream server energy consumption, making it hard to optimize or report sustainability. This "cloud opacity" means broadcasters and game providers must trust coarse estimates, hindering precise efficiency improvements. While initiatives like SCI and SGA's frameworks aim to improve accounting (e.g. measuring per-user or per-hour emissions), much of the industry still lacks granular data. Better instrumentation and data sharing from cloud providers is needed as are organizational interventions such as forming partnerships to jointly improve efficiency across the value chain.	GreenCode could support transparency by offering APIs and integrations that help developers extract energy consumption data from cloud platforms, and better predict performance before deploying to the cloud, enabling better decisions on architecture and deployment.
Skills and Awareness Gaps	Despite efforts by the GSF and SGA, software teams often lack specialized knowledge or training in energy-efficient design, so efficiency best practices are not applied by default.	A game development team might focus solely on performance and graphics, unaware of coding patterns that could cut power usage. This gap in expertise means opportunities for energy savings (like algorithmic optimizations or efficient memory use) are frequently missed during development. Developers need to be formally exposed to green coding techniques and need clear advice, metrics or requirements on efficiency to make it a first-class requirement in software development vs. an afterthought.	GreenCode could add value through targeted training, examples, and documentation that demystify energy-efficient practices, empowering developers to build greener software even without prior expertise.
Performance Over Efficiency Focus	The industry's emphasis on high performance and quality (UHD video, high FPS gaming) demanded by hardcore gamers can overshadow efficiency goals, reducing incentives to optimize for lower power. When games introduce "eco" modes or	When Call of Duty added a default Eco Mode in menus, between 7 and 20% of players (depending on platform) opted to turn it off to regain full effects [294]. Both the industry and gamers need to take responsible action with respect to expectations for entertainment. Developers may	GreenCode's tools could provide quantifiable energy-performance trade-offs, helping teams make informed decisions about when and how to prioritize energy savings without sacrificing user experience.

	slightly lower fidelity to save energy, a subset of users may disable these.	in future need to apply energy saving solutions without given the user options to disable them.	
Rising Demand Outpacing Efficiency	Increases in usage and content demands often offset efficiency gains, as services expand (more streams, higher resolutions, AI features), total energy consumption keeps climbing.	Streaming video in 4K and ubiquitous online gaming are driving up overall energy use, eroding gains from more efficient codecs or hardware. This rebound effect means that even though networks and data centres are more efficient per unit of work, the overall streaming and gaming energy footprint can still grow rapidly.	GreenCode could add value by modelling rebound effects and helping developers design software defaults and features that manage resource use proactively, ensuring efficiency gains keep pace with demand growth.
Commercial Pressure	TODO		
Rebound Effects	Games are often resourced constrained and the sustainability benefits from optimisation may be redirected into enabling more advanced features.	For instance, new console generations include energy-saving tech but then add power-hungry features that eat up those savings[295]. A cloud gaming provider might run servers at full throttle for responsiveness, prioritizing user experience over energy use, thus foregoing potential power reductions.	GreenCode will shine a light on the impact of these decisions and could help foster a more carbon responsible approach to resource usage.
Regulation	Although regulations like CSRD are coming online, many organizations are not fully prepared to comply. In gaming especially, figuring out how to attribute the energy use of players globally to a product is complex.	The SGA's work shows an intent to standardize this[Error! Bookmark not defined.], but until it matures, companies may struggle to accurately report or might undercount certain aspects. We don't yet have a "performance per watt" requirement for software, and having this might support deeper change through policy.	

GreenCode: State of the Art Review

Summary and Future Opportunities

Energy-efficient computing in media, gaming, and streaming is advancing through targeted software and hardware optimizations, ranging from power-aware console updates and idle-state game rendering tweaks to more efficient video codecs, cloud infrastructures, and virtual production.

However, significant opportunities remain: AI-powered tools like GreenCode can automate code-level energy optimizations; cross-platform design can minimize waste; and smarter scheduling can align heavy workloads with renewable energy supply.

While tools and standards are emerging, widespread adoption is still limited. Future gains lie in embedding energy-awareness across the software development lifecycle, retrofitting legacy software systems, and enabling longer device lifespans, ensuring that sustainability becomes a core design and operational priority across the digital entertainment ecosystem.

Opportunities

The Opportunities table below outlines where the sector can move from isolated optimisations to repeatable, scalable impact across the digital entertainment lifecycle. It highlights opportunities to apply AI-driven optimisation to identify and remediate energy waste in code and pipelines, adopt carbon-aware scheduling for non-urgent workloads (rendering, builds, patch distribution), and improve cross-platform design so applications exploit each device's most efficient hardware paths. It also emphasises high-impact routes that align strongly with GreenCode's strengths: retrofitting legacy games and media workflows, leveraging edge/content optimisation and accelerators, extending device lifetimes through software adaptation, and supporting emerging standards/benchmarking efforts that can make energy efficiency measurable, comparable, and incentivised across the industry.

Opportunity	Description	Example/Implication	Application To GreenCode
AI-Driven Optimization	Leverage AI to automatically analyse and optimize software and systems for lower energy consumption. AI can uncover complex patterns and adjustments beyond manual tuning.	Google's DeepMind AI famously cut data centre cooling energy by up to 40% by dynamically optimizing settings[296]. Similarly, AI-driven tools could auto-tune game engines or streaming codecs in real time, finding the best performance-per-watt settings and reducing power draw without human intervention.	GreenCode can deliver AI-powered systems that automatically analyse source code, detect inefficiencies, and propose or apply changes, enabling developers to improve energy performance with minimal manual effort.

Carbon-Aware Scheduling	Schedule and locate computational workloads based on renewable energy availability or low-carbon intensity of power sources. Non-urgent tasks can run when and where green power is plentiful, minimizing carbon footprint.	A video processing job could be queued for midday when solar power peaks, or shifted to a data centre in a region with surplus wind energy. Similarly other jobs such as rendering or patch compilation and delivery could be scheduled in a similar way. Cloud providers are beginning to offer carbon-footprint scheduling options, aligning rendering or batch updates with periods of cleaner power to cut emissions.	GreenCode can provide value by developing software modules that connect applications with real-time carbon intensity data, enabling smart scheduling of tasks to minimize emissions while maintaining performance goals.
Efficient Cross-Platform Design	Design software and engines to maximize energy efficiency across different devices and platforms by using native optimizations. This ensures that applications run lean on each hardware type without redundant overhead.	A streaming app might utilize hardware-accelerated video decoding on every platform (mobile, PC, smart TV) to reduce CPU work, or a game engine could adapt rendering techniques per device (consoles vs. phones) to avoid unnecessary computations. Since there is *no one-size-fits-all* for energy saving across devices, flexible, platform-aware design can significantly lower overall power use.	GreenCode can offer analysis tools and guidelines that identify platform-specific optimization opportunities, helping developers tailor apps for energy efficiency on diverse devices and avoid one-size-fits-all waste.
Legacy Code Optimisation in Games	Provide an AI system that analyses a game's source code or a mobile app and suggests or applies optimizations (e.g. eliminating busy-wait loops, using more efficient data structures) that cut CPU/GPU cycles, etc.	These could be game-changing benefits for games companies dealing with both new and legacy software. Improving both user experience, time to market and commercial outcomes.	GreenCode could make legacy code assessment part of the standard development pipeline, much like linters or security scanners, thereby baking sustainability into the software development life cycle (SDLC). This would directly address today's tooling gap and empower developers to act on energy inefficiencies that are currently hidden or become obscured over time.
Edge Computing & Content Optimization	Use edge computing and smarter content delivery strategies to reduce energy waste in networks and central servers. Processing data closer to users (or using peer-to-peer	The Quanteec peer-to-peer streaming solution reduces upstream cache server usage by 70-75% for live video[297], which means fewer data centre resources are needed during peak streams. By caching content at the network edge or on user devices, streaming platforms and game services can decrease backbone network	GreenCode can help developers design edge-aware applications by providing patterns and templates for distributed computing, enabling greener architectures that balance performance with reduced energy use in networks and datacentres.

	techniques) cuts down on long-distance data transport and server load.	energy and scale down centralized infrastructure, saving power.	
Specialized Hardware Acceleration	Offload heavy media and gaming tasks to specialized low-power hardware (GPUs, ASICs, FPGAs) and optimize software to use them. These accelerators perform specific tasks much more efficiently than general-purpose CPUs.	For instance, encoding or decoding video with a dedicated media accelerator uses a fraction of the energy of a CPU doing the same work. AI inferencing in games or streaming can run on neural accelerators that deliver higher performance per watt. Embracing such hardware, and writing software to utilize it, can dramatically cut energy per operation (e.g. console games using GPU for physics calculations instead of taxing the CPU).	GreenCode could map common workloads to energy-efficient hardware solutions and guide developers on how to refactor code to leverage accelerators, reducing power use without sacrificing capabilities.
Extending device lifetimes	Extending the useful life of gaming and media devices, reduces the need for new hardware production (which is energy-intensive and has a carbon cost). The industry could explore modular upgrades (like console mid-generation refreshes that are plug-in modules) or cloud assist modes for older devices (offloading heavy tasks to cloud for a device that can't handle them alone).	Designing games and apps that scale gracefully to older hardware, or offering "lite" versions, avoids forcing constant upgrades. While hardware-focused, these ideas rely on software to make them seamless. If a 2017 smart TV can still run the latest streaming app efficiently in 2025 because the software auto-adjusts to its capability, that keeps it out of the landfill longer.	The current gap in legacy support can be flipped into an opportunity: invest in optimizing and updating the most popular older software and games with efficiency patches. This "Green Retrofits" concept could become a trend, for example, an update that re-compresses textures in an old game or adds frame rate caps in menus, delivering immediate energy savings to thousands of ongoing players.
Green Software Standards & Tooling	Develop and adopt standards, frameworks, and tools that integrate energy efficiency into the software development lifecycle. This includes benchmarking software energy use and providing transparency to drive competition on efficiency.	Initiatives like the *Software Carbon Efficiency Rating (SCER)* propose consumer-friendly scores for software energy/carbon impact, analogous to EnergyStar for appliances. In practice, a media streaming service could be labelled for efficiency, motivating developers to optimize code and infrastructure. Additionally, new IDE plugins and CI tools are emerging that flag energy-heavy code or suggest greener alternatives, embedding energy awareness into everyday development and	GreenCode can contribute value by working to define measurable software sustainability standards and providing integrated tools that benchmark software energy use, helping teams target and verify energy improvements.

		pushing the industry toward more sustainable software.	
Collaboration on standards	The gaming and media industries could establish common benchmarks for energy efficiency, e.g., frames-per-joule metrics or "energy per scene rendered" benchmarks. An analogy is how mobile phone makers competed on battery life benchmarks or how Green Web efforts look at a carbon cost per user.	If game engines or streaming services are benchmarked and ranked by efficiency, it could spur competition in a positive way. Regulators might also step in: for example, extending ecodesign directives to include software efficiency or requiring app stores to list whether an app/game has passed any green certification. Such moves could accelerate innovation, as companies would have clear targets to meet (much like fuel economy standards did for cars).	Enabling the detailed measurement and certification of software systems GreenCode could help establish this as a real undertaking in market, particularly if this can be achieved in collaboration with bodies like the GSF or SGA.

GreenCode: State of the Art Review

SotA of Energy-efficient Computing for Public Sector

Contributing Partner(s): IrRADIARE, Digital Tactics

This section explores how IT/software energy efficiency is currently addressed within the public sector, its growing importance, and how green software and ICT sustainability can improve the status quo.

Background

The public sector, from local government agencies to national bodies, is increasingly reliant on complex Information and Communication Technology (ICT) systems to deliver services to citizens. This includes everything from managing citizen databases and public services to handling massive amounts of data in datacentres.

The energy consumption of these systems has a significant environmental and economic impact, with the ICT sector's electricity consumption potentially rising to 20% of the world's total by 2025 without a focus on efficiency. On the other hand, a significantly increasing number of energy efficiency and smart city projects are being developed by cities and government bodies, leading to an increase in the number of devices, computing power and, thus, energy. This creates a paradox regarding energy efficiency and the adequation of digital resources to achieve it. Additionally, the increase in green energy production is changing the production curve, and creating distinct curves for energy production, energy price and related GHG emissions, leading to an additional consideration factor regarding demand scheduling.

In this context, energy-efficient computing or emission-efficient computing is no longer a niche concern but a strategic necessity. By shaping energy use, public sector organizations can reduce operational costs, align with national and international sustainability goals, and set a leading example for the private sector.

Energy and emissions efficient computing is a multi-faceted approach that involves optimizing hardware, software, and infrastructure. While hardware improvements have been significant, the focus is now shifting to the software layer, which can be a primary source of energy waste regardless of the underlying hardware.

This is particularly important for the public sector because:

- Decarbonization at Scale: Governments, particularly regional and local, are some of the world's largest consumers of technology. Optimizing software on a national scale can lead to significant reductions in the overall carbon footprint of a country's digital infrastructure.

- Circular Economy: A focus on software efficiency supports the public sector's move towards a circular economy for IT, extending the useful life of existing hardware by making the software that runs on it more efficient.

- Policy Compliance: With increasingly high targets for GHG emission reductions, software optimisation is critical for government bodies to meet new, stricter energy consumption targets.

- Broader electrisation and digitalisation: the leaning towards electricity usage in devices, mobility and the broader availability of monitoring and controlling allows for the shaping of the demand curve to match energy, emission or cost goals.

Current State of the Art

The current state of the art in energy-efficient computing for the public sector is characterized by a multi-layered approach that combines policy, procurement, and technical solutions.

Policy and Regulation

Governments are establishing frameworks and policies to drive sustainability. For example, in Portugal there is the National Energy and Climate Plan (PNEC 2030) that is the main energy and climate policy instrument. It outlines targets and measures for the decade 2021-2030, including a 55% reduction in greenhouse gas emissions compared to 2005 levels and a target of 51% renewable energy in gross final energy consumption by 2030. The plan also focuses on energy efficiency, energy security, and the development of renewable energy sources like solar and wind.

Green Public Procurement (GPP)

Public bodies are leveraging their significant purchasing power to drive change. GPP policies mandate the procurement of goods and services with reduced environmental impact throughout their life cycle. Alongside procurement, public sector bodies are increasingly operationalising sustainability through service design and delivery standards.

Portugal

In Portugal, ECO360 - National Strategy for Green Public Procurement 2030 is the central policy for GPP, approved in February 2023. Its main objectives are to enhance the use of GPP as a tool for a sustainable transition, promote resource efficiency, stimulate the circular economy, and encourage eco-innovation in the market.

United Kingdom

In the UK, cross-government work has produced the Greener Service Principles (v1.0, March 2025) as practical guidance for designing, building, and operating lower-impact digital services. [298]

This is being reinforced by formal service design doctrine: the UK government has publicly described adding sustainability as a design principle for government services (April 2025), signalling a shift from “best-effort” green intentions to repeatable expectations embedded in delivery practice[299].

In practice, this pushes teams toward concrete software choices that reduce demand (lighter pages, fewer dependencies, lower data transfer, simpler journeys) and makes sustainability reviewable alongside accessibility, security, and reliability.

The Netherlands

In the Netherlands other responsible procurement techniques have been introduced from a guidance and execution perspective with organisations like the Buyers Group (ICT) establishing tools that help procurement professionals engage with sustainable IT and software suppliers without getting bogged down in technicalities.

Spain

Spain provides a clear national-policy anchor for green procurement through the "Plan de Contratación Pública Ecológica 2018–2025", which promotes the systematic inclusion of environmental criteria in public purchasing and positions procurement as an implementation instrument for wider sustainability objectives. This creates a consistent policy signal that can be applied across major ICT and digital service procurements rather than relying on isolated "green IT" initiatives.[300]

Germany

Germany illustrates a procurement-ready route based on third-party criteria and verification. The Blue Angel ecolabel includes award criteria for "Resource and Energy-Efficient Software Products" (DE-UZ 215) and has also defined criteria for energy-efficient data centre operation (DE-UZ 161). These instruments provide a practical mechanism for public buyers to specify efficiency requirements in tenders and reference an established assessment scheme rather than inventing bespoke criteria for each procurement.

Denmark

Denmark offers an example of translating EU-level green procurement criteria into actionable tender guidance. The Danish Agency for Digital Government examined green requirements for tenders covering datacentres, server rooms, and cloud services, concluded that the EU's Green Public Procurement criteria are the most appropriate to use, and reports that these criteria were tested in selected public procurement processes in 2022–2023. The resulting requirements have been incorporated into public tender guidelines provided via SKI, supporting repeatable adoption by public sector buyers[301].

Rest of the World

Outside Europe, public-sector "green IT" practice is typically implemented through procurement policies, ecolabel/standards-based tender requirements, and supplier disclosure obligations, with the strongest maturity today in hardware, data centres, cloud services, and organisational carbon reporting. Examples include Canada's federal green procurement policy and supplier disclosure standard, the United States' use of EPEAT/ENERGY STAR-anchored requirements for electronics, India's growing guidance and institutional mechanisms around sustainable procurement, China's use of environmental labelling and government procurement lists, and similar procurement frameworks across Japan, South Korea, Australia, and Singapore.

Green software is likely to become a more explicit consideration in these jurisdictions for two reasons: (1) as public services become increasingly digital and cloud-hosted, procurement and audit regimes are expanding from "buy efficient equipment" to "operate efficient services," and (2) supplier disclosure and net-zero commitments create pressure to demonstrate reductions in operational emissions that cannot be achieved through infrastructure choices alone. In the near term, this will most plausibly emerge as software-adjacent requirements e.g., reporting operational emissions for digital services, workload efficiency expectations for hosted services, carbon-aware operations for batch processing, and evidence of energy/performance regression controls, before it

matures into explicit tender criteria for software efficiency (such as measurable energy-per-transaction or efficiency baselines under representative workloads).

Carbon-aware Computing and Demand Shaping

Beyond efficiency-in-place, the state of the art is also moving toward carbon-aware workload shifting, where non-interactive workloads (batch analytics, backups, ETL, ML training) are scheduled to run when electricity is cleaner and/or cheaper. The Green Software Foundation's Carbon Aware SDK[302] exemplifies this tooling direction: it supports integrating carbon-intensity signals into software so systems can choose when and where to run to reduce emissions without necessarily changing functionality.

For public sector estates, often dominated by periodic workloads, reporting cycles, and overnight batch jobs, this creates a practical lever that complements GreenCode's code-level optimisation story (especially when code changes alone cannot be deployed quickly due to assurance and change-control).

Limitations and Gaps

The Limitations and Gaps table below summarises why public-sector digital estates often struggle to achieve measurable energy and carbon reductions through software change alone. It highlights structural constraints such as large legacy portfolios, fragmented ownership across departments and suppliers, and limited ability to obtain consistent cross-layer telemetry (application <> infrastructure <> facilities <> grid factors), particularly in outsourced and hybrid environments. The table also captures non-technical barriers that are especially pronounced in the public sector: procurement and compliance cycles that prioritise cost, risk, and availability; weak accountability mechanisms for operational efficiency; limited specialist capacity; and the absence of standardised, comparable methods for defining measurement boundaries, baselines, and "green regression" expectations across services.

Limitation/Gap	Description	Example/Implication	Application to GreenCode
Lack of Software Efficiency Standards	There are no universally agreed-upon metrics or standards for what constitutes "green software". This makes it difficult for public sector bodies to compare the energy efficiency of different applications and to set clear procurement criteria.	A government agency may not have a clear benchmark to evaluate if a new software solution is more energy-efficient than its existing one.	GreenCode can provide a certification framework to establish a clear and verifiable standard for software energy efficiency, filling a critical industry gap and enabling GPP.
Legacy Systems and Technical Debt	Many public sector organizations still rely on older software and hardware systems that are not designed for energy efficiency. The cost and complexity of modernizing these systems can be a major barrier.	A city government might be running critical services on a mainframe from the 1990s, where any change is a high-risk and costly endeavour.	GreenCode's AI-driven optimization can modernize legacy codebases, enhancing their efficiency and extending their useful life without a full-scale, costly replacement.

High Upfront Costs of Hardware	While energy-efficient hardware and retrofitting existing systems can lead to long-term savings, the initial capital investment can be prohibitive for budget-constrained public entities.	A public hospital might find it difficult to justify a large investment in a new, energy-efficient data centre, even if it saves money over a decade.	GreenCode provides a software-first approach to efficiency, offering a high-impact, lower-cost alternative that does not require significant hardware upgrades.
The "AI Paradox"	While AI is a powerful tool for optimizing efficiency, the computational power required to train and run large AI models is itself a massive consumer of energy.	Training a single large language model can consume a significant amount of energy, creating a paradox where the solution itself contributes to the problem.	GreenCode's core mission is to use AI to make computing more efficient on a broader level, effectively using AI to solve a problem that AI itself contributes to.

Summary and Future Opportunities

Energy-efficient computing is a cornerstone of the public sector's move toward sustainability, driven by both cost savings and regulatory pressure. While progress has been made through policy, procurement, and data centre optimization, the greatest remaining opportunity lies in the realm of software.

The GreenCode project is uniquely positioned to address the most critical shortcomings of the current state of the art by:

- Creating a "Green Software Certification": Providing the industry with a clear and credible standard for evaluating and certifying software's energy efficiency.
- Leveraging AI for Continuous Optimization: Using AI and machine learning to continuously adapt and improve software as technologies and workloads evolve.
- Bridging the Policy-to-Practice Gap: Offering a tangible, software-based solution that enables public sector bodies to meet their sustainability commitments and GPP goals without requiring prohibitively high hardware investments.

By focusing on these areas, GreenCode can provide its partners with a competitive advantage and make a lasting contribution to the decarbonization of computing at scale within the public sector.

Opportunities

The Opportunities table below outlines where the public sector is uniquely positioned to accelerate adoption of sustainable software engineering at scale. It highlights the leverage of procurement and policy to create clear demand signals (e.g., measurable sustainability requirements, evidence expectations, and reporting alignment), and the potential to embed sustainability into service management and delivery pipelines via standardised measurement modes, CI/CD gates, and governance-ready reporting artefacts. It also emphasises opportunities to prioritise high-impact interventions—legacy rationalisation, shared platform optimisation, and workload placement, supported by transparent benchmarking and assurance approaches that suppliers can meet consistently. In this way, sustainability can become a managed operational objective across public digital services rather than a best-effort aspiration.

Opportunity Area	Description	Expected Impact on Energy Efficiency	Alignment with GreenCode

Energy efficiency as a procurement requirement	Embedding explicit, software-level energy and sustainability criteria into public sector procurement frameworks, tenders, and digital service standards.	Drives market-wide adoption of energy-efficient software practices through purchasing power.	Strongly aligned with GreenCode's goal of operationalising sustainability in decision-making processes.
Standardised, software-centric energy measurement	Adoption of consistent methodologies for measuring and reporting software energy consumption across public sector systems, services, and suppliers.	Enables comparability across projects and supports evidence-based optimisation.	Directly supports GreenCode's emphasis on measurement, traceability, and evidence.
Lifecycle-aware sustainability assessment	Evaluating energy and environmental impacts across the full lifecycle of public sector software systems, including development, deployment, operation, and decommissioning.	Prevents long-term energy inefficiencies caused by incremental system growth and technical debt.	Closely aligned with GreenCode's lifecycle-oriented sustainability model.
Modernisation of legacy public sector systems	Systematic energy-aware refactoring, consolidation, and modernisation of legacy applications and infrastructures common in public administrations.	Reduces energy waste arising from outdated architectures and inefficient software patterns.	Supports GreenCode's focus on legacy code analysis and sustainable evolution.
Cross-layer observability and accountability	Linking software architecture, runtime behaviour, and infrastructure energy metrics to improve transparency and accountability in public sector IT operations.	Enables identification of energy hotspots and prioritisation of optimisation efforts.	Aligns with GreenCode's integrated architecture-runtime-measurement approach.
Policy-aligned sustainability assessment frameworks	Aligning software energy assessment with existing public sector governance, audit, and reporting frameworks (e.g. digital service standards, sustainability reporting obligations).	Facilitates institutional adoption without introducing parallel, disconnected processes.	Positions GreenCode as a bridge between engineering practice and policy compliance.
Energy-aware cloud and hybrid deployment strategies	Optimising workload placement, scheduling, and scaling decisions across cloud and on-premise environments based on energy and carbon considerations.	Reduces energy and carbon intensity of public digital services at scale.	Supports GreenCode's system-of-systems perspective across deployment environments.
Evidence-based certification and transparency mechanisms	Supporting credible sustainability claims through verifiable evidence, enabling public trust and regulatory confidence in digital services.	Encourages continuous optimisation rather than one-off reporting exercises.	Complements GreenCode's role as an evidence generator for certification and reporting.

GreenCode: State of the Art Review

SotA of Energy-efficient Computing for Mainframe Applications

Contributing Partner(s): VBT

Background

Mainframe systems have been the backbone of critical systems for many large enterprises, especially in industries like finance, insurance, and healthcare. They are designed for stability, scalability, and reliability. Despite their robust capabilities, many mainframe systems are based on legacy code that was not designed with energy efficiency in mind. As the demand for sustainable computing increases, optimizing the energy consumption of these systems has become a priority.

Energy optimization of mainframe legacy code is a particularly challenging task because it involves updating and modernizing systems without disrupting critical operations. The volume of legacy code is huge, and the number of mainframe developers who can optimize the code is insufficient.

Generative AI (GenAI) and Natural Language Processing (NLP) provide promising approaches to optimize the energy efficiency of these legacy systems. By using AI models to refactor, analyze, and optimize legacy code, GreenCode aims to help organizations achieve better performance while reducing energy consumption.

Current State of the Art

Challenges of Mainframe Legacy Systems

Although technological improvements on the hardware and system software consider reducing energy usage, the presence of legacy software on mainframe systems leads to following issues:

- There are millions of mainframe software codes from decades ago.
- Tight Coupling with Hardware: Mainframe systems are often closely tied to specific hardware architectures. Optimizing code for energy efficiency requires understanding how software interacts with hardware.
- Lack of Documentation: Legacy codebases often lack proper documentation, making it difficult for modern AI tools to fully understand the context of the code and suggest improvements.
- Skill Shortage: There is a limited pool of developers who are proficient in mainframe programming languages like COBOL, making it harder to refactor legacy code manually.
- Critical Nature of Applications: Mainframe applications often run mission-critical workloads. Making changes without thorough testing can lead to system instability or downtime and replacing them with more modern systems can be costly and risky.
- Legacy software limits agility, making it difficult for organizations to respond quickly to market changes and regulatory demands.
- Ultimately, it negatively impacts the institution's competitive advantage.

Importance of mainframe systems in IT domain

Mainframe systems, traditionally built on platforms like IBM's z/OS, handle vast volumes of data processing. While they are incredibly efficient in terms of throughput and transaction processing,

they were not designed with energy efficiency as a primary consideration. In many cases, energy consumption is directly linked to the complexity of the code being executed, the hardware usage patterns, and the algorithms in place.

Mainframe systems are widely used in sectors with high online transaction volumes. According to a 2021 report by IBM [303] :

- 45 of the top 50 banks,
- 4 of the top 5 airlines,
- 7 of the top 10 global retailers,
- 67 of the Fortune 100 companies

leverage the mainframe as their core platform.

IBM's report shows that mainframes still handle almost 70% of the world's production IT workloads.

Key Concepts and Technologies

Energy Optimization Techniques for Mainframe Systems

In the context of mainframe systems, energy optimization techniques include:
Algorithmic Efficiency: Mainframe applications often rely on large-scale data processing. Optimizing the algorithms used for sorting, searching, and transaction processing can drastically reduce energy consumption. For instance, switching from inefficient sorting algorithms to more efficient ones can result in significant energy savings.

- Memory Optimization: Mainframe systems often use large amounts of memory. Efficient memory management, such as reducing memory leaks, minimizing memory usage in critical paths, and optimizing cache utilization, can improve energy efficiency.
- Parallelism and Concurrency: Modernizing legacy systems to leverage multi-threading and parallel processing capabilities can help distribute workloads across multiple processors or cores, reducing processing time and energy consumption.
- Efficient I/O Operations: Many mainframe systems perform intensive I/O operations. Optimizing how data is read from and written to storage can significantly reduce energy overhead, especially when dealing with large data volumes.
- Reducing Redundant Operations: Legacy systems may contain redundant computations or inefficient loops that waste CPU cycles. Identifying and eliminating these redundancies can lead to a noticeable reduction in energy consumption.

Generative AI (GenAI) for Mainframe Code Optimization

Generative AI models are particularly useful in the context of legacy code optimization. These models can be trained to understand the structure and behavior of legacy code and generate suggestions for improving its energy efficiency. GenAI can be leveraged for:

- Code Refactoring: GenAI models can automate the process of refactoring inefficient algorithms and code structures. For example, they can suggest optimized versions of legacy sorting algorithms or reduce the complexity of memory-intensive operations.

- Automated Code Generation: AI models can generate energy-efficient code based on modern best practices, replacing legacy code that was written without energy optimization in mind.

- Energy-Aware Code Suggestions: AI models can suggest energy-efficient alternatives for existing code, such as replacing resource-heavy functions with more optimized ones.

- Cross-Language Translation: GenAI can help translate older mainframe languages (e.g., COBOL) to more modern, energy-efficient programming languages, improving maintainability and performance.

Natural Language Processing (NLP) for Code Understanding and Refactoring

NLP tools are essential for understanding the semantics of mainframe code and automating its optimization:

- Code Summarization: NLP techniques can generate high-level descriptions of legacy code, helping developers quickly understand the logic and identify inefficient areas that could be optimized for energy efficiency.

- Semantic Code Analysis: NLP models can analyze the code's logical structure, helping to identify sections that may be inefficient or redundant and suggesting improvements.

- Code Transformation: NLP can assist in translating old mainframe code (e.g., COBOL) into more modern and energy-efficient equivalents. This transformation can also involve converting procedural code into more modular and parallelizable forms.

- Automated Documentation: NLP can automatically generate documentation for legacy code, which makes it easier to identify areas that could benefit from optimization.

Framework for Energy Optimization of Mainframe Legacy Code Using GenAI and NLP

To leverage GenAI and NLP for energy optimization in mainframe legacy code, we propose a multi-step framework that includes the following stages:

Step 1: Energy Profiling

Before any optimization efforts can take place, it is important to understand where energy inefficiencies are occurring. Profiling tools designed for mainframe systems (e.g., IBM's Energy Management Suite) can provide detailed insights into energy usage patterns. These tools track CPU utilization, memory consumption, disk I/O, and network activity, providing valuable data to guide the optimization process.

Step 2: Code Understanding with NLP

NLP tools can assist in automating the analysis of the legacy code:

- Code Summarization: NLP tools can provide high-level summaries of the code's functionality, making it easier for developers to identify areas for energy optimization.
- Semantic Code Analysis: NLP models can analyze the relationships between different parts of the code, identifying redundant or inefficient operations that contribute to unnecessary energy consumption.
- Automated Refactoring Suggestions: NLP-powered tools can automatically suggest changes to improve code structure and reduce energy usage.

Step 3: Code Refactoring with GenAI

Once energy inefficiencies have been identified, GenAI models can be used to refactor the legacy code:

- Algorithmic Refactoring: GenAI models can suggest more efficient algorithms for data processing tasks.
- Memory Management Refactoring: AI models can suggest changes to optimize memory usage, reducing overhead caused by memory leaks and inefficient allocations.
- Concurrency Refactoring: GenAI can refactor code to take advantage of modern multi-threading or parallel processing capabilities, distributing workloads across available processors.

Step 4: Validation and Testing

Once the code has been refactored, it is important to validate the changes. AI models can assist by automatically testing the refactored code for correctness, performance, and energy efficiency. This step ensures that the refactor does not negatively impact the functionality of the system.

Step 5: Continuous Monitoring and Optimization

Mainframe systems require ongoing maintenance. Using AI and NLP for continuous monitoring of energy consumption can help identify further optimization opportunities over time. This step ensures that the energy efficiency of the mainframe system continues to improve as new workloads are added.

Existing Solution Providers

In the mainframe sector, there are several products/platforms aimed at application modernization: IBM watsonx Code Assistant, AWS BluAge, AWS Q Developer, Google Mainframe Modernization Solutions (G4), Google Duet AI for Dev, GitHub Copilot, and TSRI.

Apart from IBM watsonx Code Assistant, the main goal of these tools is application modernization; they aim to convert legacy mainframe code into more modern Java or C++ languages and move the new application to the cloud, without a decarbonization goal.

Tools	Focus	Core Function	Target Use Case
IBM watsonx Code Assistant for Z	Built specifically for mainframe modernization	Uses AI to assist in Cobol code refactoring, including	Mainframe Cobol optimization

	and optimization within IBM Z ecosystem.	code simplification, performance improvement, memory and CPU efficiency.	
AWS BluAge	Automated mainframe application refactoring	Automatically translates legacy code into modern microservices-ready Java or cloud-native code. Re-platform applications to run on AWS cloud not on IBM Z.	Large legacy applications in Cobol, RPG, etc. Organizations looking to fully replace legacy systems with modern Java-based applications. Replace Cobol on AWS.
AWS Q Developer	AI-assisted software development and modernization	Assists developers in understanding, refactoring and testing code. Can help generate unit tests, explain code structure, etc. Integrated with tools like AWS CodeWhisperer and IDE plugins.	Developers working on code analysis, manual or semi-automated modernization, or incremental refactoring. More granular and interactive support compared to full automation.
Google G4 / Dual Run	Cloud-first re-platforming	Uses automated code conversion template to convert to languages like Java that can be deployed in Google cloud.	For organizations whose end goal is cloud re-platforming.
Google Duet AI for Dev	Google's equivalent to GitHub Copilot	Productivity assistant that can help developers with coding, data analysis, and collaboration.	Useful for modern languages not mainframe environments.
GitHub Copilot	AI assistant by GitHub+OpenAI	Can help with basic Cobol syntax suggestions. Directly integrates with IDEs. Can adapt to individual coding styles.	Individual developers and enterprises working on code analysis, modernization, or incremental refactoring.
TSRI	Specializes in automated code transformation, including Cobol	Migration from Cobol to Java, C++, etc.	For organizations whose end goal is cloud re-platforming.

GreenCode: State of the Art Review

GreenCode/Industry Feature comparison

Features/ Tools	GreenCode	IBM watsonx Code Assistant for Z	AWS BluAge	AWS Q Developer	Google G4 / Dual Run	Google Duet AI for Dev	GitHub Copilot	TSRI
Energy Optimization	☑ Yes	☑ Yes	✗ No	✗ No	✗ No	✗ No	✗ No	⚠ Indirect
Energy Profiling	⚠ Via integration	⚠ Via integration	✗ No	✗ No	✗ No	✗ No	✗ No	✗ No
Platform flexibility for Generated/Optimized Code	☑ Yes, Mainframe or appropriate platform	☑ Yes, Mainframe or appropriate platform	✗ No, moves to cloud environment	✗ No, moves to cloud environment	✗ No, moves to cloud environment	✗ No, moves to cloud environment	✗ No, moves to cloud environment	✗ No, moves to cloud environment
COBOL Support	☑ Yes	☑ Strong, produces efficient code in Cobol or Java	☑ Converts line by line to Java	⚠ Limited, helps in code understanding	☑ Converts line by line to Java	✗ No	⚠ Basic, code understanding and translating into modern languages	☑ Converts line by line to Java, C++
PL/I Support	☑ Yes	⚠ Soon	☑ Converts line by line to Java	✗ No	☑ Converts line by line to Java	✗ No	✗ No	☑ Converts line by line to Java, C++
Application Discovery	☑ Yes	⚠ Via Wazi	☑ Strong (for migration)	⚠ Some (Java migration focused)	⚠ Some	✗ No	✗ No	☑ Yes
Refactoring technique	☑ GenAI	☑ GenAI	⚠ Rule-based	⚠ GenAI assistance on understanding	⚠ Rule-based	✗ No legacy language support	⚠ Basic, GenAI	☑ Hybrid (GenAI + Rule-based)
Code Understanding technique	☑ NLP	☑ GenAI + static and flow analysis	☑ Static analysis, Model Driven Architecture, reverse engineering	☑ Basic, GenAI based explanations	⚠ Runtime analysis	✗ Pure GenAI for modern languages only	☑ NLP	☑ Semantic graphs + full program analysis
Validation & Testing	☑ Yes	☑ Basic	☑ Yes	✗ No	☑ Cloud	✗ No	✗ No	☑ Cloud
On Premise LLM	☑ Yes	☑ Yes	✗ No	⚠ Planned	✗ No	✗ No	✗ No	✗ No
Mainframe specific LLM	☑ Yes	☑ Yes	✗ No	⚠ Partial	✗ No	✗ No	✗ No	✗ No

GreenCode: State of the Art Review

Solutions at object code level, at run-time (without source code optimization)

IBM, the main product provider in the mainframe sector, offers various solutions at different levels (without source code optimization) (These solutions can always be used as complementary to any of the above-mentioned tools.):

- IBM Automatic Binary Optimizer (ABO), enhances Cobol applications to be more CPU-efficient without compiling (might yield energy savings), based on object code. It doesn't provide documentation on source code.

- Compiler-level optimizations via IBM Enterprise COBOL for z/OS compiler. The latest compiler version includes advanced optimization flags and energy-efficient compile options, uses newer hardware instructions that are more power-efficient, leading to a reduction in CPU time for the same job/task.

- z/OS Performance Tools (RMF, SMF, Omegamon) to measure energy/CPU consumption. These tools do not optimize code directly but provide critical performance and energy telemetry needed to make optimization decisions.

As seen in the table above, only IBM watsonx Code Assistant has features similar to those of GreenCode in the "mainframe legacy code conversion" domain. However, GreenCode goes further by evaluating energy consumption and infrastructure energy assessments along with the functional comparison of old and new code in the mainframe platform. GreenCode will differ also in semantic business understanding with smarter business domain-aware modelling. GreenCode will offer a valuable alternative to IBM mainframe users.

GreenCode not only performs corrective actions on code bases through analysis but also automates the decarbonization of software systems using cutting-edge GenAI. Additionally, GreenCode will provide a holistic, transparent, and well-controlled approach to improving software quality and delivering other valuable benefits. It will offer this as a complete decarbonization suite across many system environments and programming languages.

Limitations and Gaps

Improving energy efficiency in mainframe estates remains challenging despite their mature operational discipline. Constraints such as opaque or vendor-specific telemetry, limited access to fine-grained instrumentation for software-level attribution, and the difficulty of comparing energy efficiency across mixed estates where mainframes coexist with distributed platforms are key issues.

There are also structural barriers to optimisation: large, business-critical legacy codebases with scarce skills, high change risk and long release cycles, and modernisation approaches (including AI-assisted translation) that can introduce new technical debt, unpredictable performance characteristics, or loss of platform-specific efficiencies.

Finally, there is a lack of widely adopted, mainframe-relevant sustainability benchmarks and assurance pathways that can link measured impact to governance and procurement expectations.

Limitation/Gap	Description	Example/Implication	Application To GreenCode
Energy Profiling is Missing or Not Integrated	Tools don't include energy profiling, carbon footprint estimation or energy modelling. No inside into which code regions consume the most energy, I/O vs CPU vs memory vs wait cycles.	IBM WatsonX Code Assistant for Z gives CPU consumption of the legacy code and the optimized code. Other energy and performance related data is collected on HMC and SMF/RMF but remains unintegrated.	Integrate energy and performance profilers (mainframe HMC and SMF/RMF records) and build models to estimate energy per function/module.
GenAI and Rule-based Refactoring Tools Lack Energy/Performance-Aware Guidance	Neither GenAI (e.g., IBM WatsonX, Copilot) nor rule-based refactoring tools are trained for low-energy patterns.	They focus on functionality. They may suggest more readable but less efficient code (e.g., replacing nested loops with map/filter pipelines without knowing energy trade-off).	Train/refine LLMs using corpora annotated with energy/performance benchmarks or energy/performance cost models per instruction pattern.
No Feedback Loop Between Runtime Energy Data and Static Code	Tools don't use runtime energy usage to inform further static refactoring.	Example: If a specific VSAM (IBM proprietary data system) access pattern causes I/O spikes, tools don't suggest buffered reads or index adjustments	Introduce a closed-loop system: Runtime → Profiling → Static Code → GenAI/Rules → Runtime.
Lack of Deep Business Logic Understanding	Tools often translate code literally but miss the domain intent or embedded business rules.	Example: A Cobol payroll system's logic is translated to Java but loses HR-specific rules or regulation context. This leads to functional divergence or broken compliance.	Integrate domain-specific language models or knowledge graphs to infer business rules before refactoring.
Weak Data and Dependency Flow Analysis		They don't resolve cross-file data flows, especially with includes, copybooks or shared memory. They miss I/O paths, VSAM/DB2 schemas or JCL-to-program linkages.	Add bidirectional static + dynamic analysis with mainframe-specific I/O modeling.

Summary and Future Opportunities

Mainframe applications sit in a distinctive position in the sustainability landscape: they often run highly consolidated, operationally mature workloads, yet the path from platform-level efficiency to software-level optimisation evidence is still weak.

The main blockers are not a lack of operational control, but limited fine-grained attribution and comparability in hybrid estates, combined with the high risk and inertia of business-critical legacy codebases and scarce specialist skills. Modernisation and AI-assisted translation can help address maintainability and talent constraints, but can also introduce new performance uncertainty or technical debt unless changes are tightly validated.

For GreenCode, the opportunity is to pair mainframe discipline (change control, capacity planning, workload management) with sustainability-aware measurement, attribution and regression

validation, enabling optimisations and modernisation choices that are measurable, comparable, and governable across the wider enterprise environment.

Opportunities

Mainframe contexts offer distinctive leverage for sustainability improvements when approached with evidence and appropriate guardrails. Opportunities exist to combine existing operational strengths (capacity planning, workload management, disciplined change control) with energy-aware measurement and attribution that can connect application behaviour to platform signals. High-impact modernisation and optimisation pathways, selective refactoring, automated remediation of inefficiencies, and AI-assisted analysis/translation with regression validation, can reduce waste while preserving the reliability and performance characteristics mainframes are valued for. The opportunity to develop mainframe-specific benchmarks, reporting artefacts, and assurance approaches so efficiency improvements are measurable, comparable, and actionable across hybrid enterprise estates is important.

Opportunity	Description	Example/Implication	Application To GreenCode
Energy/Performance Profiling	Energy/Performance-aware profiling is crucial point for the refactoring that has the prime target as energy-efficiency.	The KPIs for energy and performance evaluation of application should be checked after each optimization step. Without effectively measuring the energy hotspots the refactorization cannot reach its optimization target.	Mainframe systems have proprietary architectures with closed monitoring APIs. GreenCode will evaluate the energy and performance data collected by the IBM Mainframe hardware and monitoring/management software (Support Elements, HMC, RMF/SMF) and push this data to the code optimization pipeline. The energy/performance data will be used to detect and re-optimize code with high energy smells.
Refactoring with LLMs Trained Using Energy-aware Benchmarks and Cost Models	Be the first GenAI-based refactoring tool focused on energy efficiency, with profiling, hotspots, and low-energy pattern libraries.		GreenCode aims to develop fine-grained energy maps at function/module level. To do this, it will train/refine LLMs using corpora annotated with energy benchmarks or energy cost models per instruction pattern.
Semantic Business Understanding	Tools often translate code literally but miss the domain intent or embedded business rules.	Example: A Cobol payroll system's logic is translated to Java but loses HR-specific rules or regulation context. This leads to functional divergence or broken compliance.	GreenCode will integrate smarter domain-aware language models or knowledge graphs, potentially fine-tuned on HR, finance, banking, logistics, insurance, etc. domains to infer business rules before refactoring.

GreenCode: State of the Art Review

Conclusion and Synthesis

Overall Conclusion

This state-of-the-art review shows that software sustainability has moved from a niche engineering concern to an emerging mainstream requirement, driven by rising energy costs, policy pressure, and ESG reporting expectations. As noted throughout the thematic chapters, the field has developed a growing set of concepts, measurement tools, and best-practice guidance, but the capabilities remain fragmented and unevenly operationalised across software types and deployment contexts.

A consistent theme across domains is the gap between knowing and doing: organisations may be able to estimate or report energy/carbon footprints, but often lack practical, repeatable pathways to reduce them in production software without compromising performance, reliability, or delivery speed. This gap is especially visible where systems are complex (distributed and multi-tenant cloud services), constrained (embedded and heterogeneous platforms), or difficult to instrument (legacy and proprietary environments such as mainframes).

At the same time, AI has accelerated software engineering tooling, but much of the current generation of GenAI and rule-based refactoring capability is trained for correctness and developer productivity rather than energy/performance-aware optimisation, and rarely supports closed-loop feedback from runtime evidence back into code change recommendations. The result is that current tools can modernise or refactor code, but do not consistently optimise for energy efficiency or provide traceable evidence that links code changes to verified energy reductions.

Standards and certification initiatives, particularly SCI and related schemes, are maturing and provide an increasingly important foundation for comparable reporting and procurement-driven incentives. However, they currently emphasise calculation and disclosure more than automated, evidence-backed optimisation workflows. This creates an opportunity for complementary tooling that turns “measurement and reporting” into continuous improvement, enabling certification to evolve from passive disclosure towards validated engineering action.

Against this backdrop, GreenCode is positioned to integrate the developments surfaced in other projects, standards bodies, and tool ecosystems into a single, modular approach that links (i) software quality/performance engineering, (ii) measurement and attribution, (iii) AI-assisted optimisation, and (iv) standards-aligned reporting and assurance. Its emphasis on legacy modernisation and optimisation addresses a major practical reality: significant emissions and costs are locked into existing systems, and measurable improvements can be realised rapidly if evidence and optimisation are integrated into the maintenance lifecycle.

GreenCode: State of the Art Review

Consolidated Gap and Opportunity Analysis

This analysis consolidates the findings of the previous sections as a set of high-level gaps and opportunities for the GreenCode project to deliver value.

Gap	Current state	What's missing	GreenCode opportunity
Measurement is available, but not consistently automated, repeatable, and comparable	Many approaches exist (on-chip sensors, external measurement, provider telemetry), but results vary with tooling, access constraints, and workload design.	Automated measurement pipelines and repeatable protocols that work across environments and generate consistent evidence suitable for engineering decisions.	Provide an end-to-end evidence pipeline (capture → store → analyse → compare) that can be applied consistently and scaled through automation.
Limited attribution from energy consumption to actionable software artefacts	Energy evidence is often system-level; attribution to modules/functions is hard, especially in multi-process, distributed, or legacy systems.	Practical, usable attribution methods that map energy/performance hotspots to code regions and architectural components.	Build software-centric energy maps and traceability that connect runtime measurements to code and architectural representations.
Weak "closed loop" between runtime behaviour and refactoring/optimisation tools	Many tools analyse code statically or refactor for readability/maintainability, but do not use runtime energy data to drive iterative optimisation.	A closed-loop workflow: Runtime evidence → profiling → refactoring guidance → validation → re-measurement.	Make closed-loop optimisation a core capability across target domains.
GenAI tooling is not energy/performance-aware by default	GenAI and refactoring tools are generally not trained on energy-efficient patterns and may recommend changes that are functionally correct but energy-suboptimal.	Training/benchmarking approaches that include energy and performance cost signals, and that encode low-energy patterns and trade-offs.	Develop/refine models and pattern libraries grounded in benchmarks/cost models and validated against measured outcomes.
Traceability for audit, certification, and procurement remains limited	Even where energy is measured, results are not consistently produced in forms suitable for assurance, certification, or comparable procurement claims.	Evidence bundles with clear boundaries, assumptions, and reproducible measurement contexts.	Generate traceable, auditable evidence artefacts aligned to SCI and certification workflows, reducing risk of superficial "green badging".
Multi-tenant and shared infrastructure allocation rules are inconsistent	In cloud/platform environments, per-service energy/emissions depends heavily on allocation choices (idle sharing, contention, baselines), producing inconsistent results.	Explicit allocation rules, confidence grading, and transparency about assumptions.	Provide defined allocation models and confidence levels that remain compatible with standards-aligned reporting.
Lifecycle effects are under-addressed (software	Many studies and tools focus on point-in-time improvements and do not adequately address	Lifecycle-aware tracking of energy/performance changes across versions and	Treat sustainability as a continuous property and support

evolution, configuration drift, firmware changes)	long-lived software evolution and regression in efficiency.	operational contexts.	ongoing monitoring and regression detection.
Domain constraints prevent "one-size-fits-all" tooling	Mainframes have proprietary monitoring channels; embedded systems are heterogeneous; web/SaaS is elastic; gaming/media has performance/latency constraints.	A modular approach that allows domain-specific adapters and optimisation strategies without fragmenting the overall evidence model.	Deliver a core platform plus sector-specific modules, maintaining a common evidence and optimisation lifecycle across domains.
Adoption barriers remain organisational, not just technical	Even where tactics exist, adoption can be limited by perceived trade-offs, costs, and resistance; sustainability may be waived under performance/SLA pressure.	Tooling that integrates with existing SDLC workflows and supports explicit trade-off decisions with evidence.	Embed sustainability into day-to-day engineering, with decision records, patterns, and evidence-based exceptions.

GreenCode: State of the Art Review

High Level Requirements

The following high-level requirements are derived from the gaps identified above and will be combined with those identified in the use-case definition and preliminary requirements work.

1. **Evidence-first measurement automation**
   Provide automated capture of energy/performance evidence across runs, with repeatable protocols and explicit system boundaries.

2. **Multi-source measurement adapters**
   Support heterogeneous measurement sources (on-chip, external/wall-plug, platform telemetry, provider telemetry), with clear metadata on precision, assumptions, and confidence.

3. **Attribution and traceability to software artefacts**
   Map measured behaviour to code regions, modules, and architectural components to make evidence actionable for engineering teams.

4. **Closed-loop optimisation workflow**
   Implement a continuous improvement loop: instrument → profile → recommend change → validate → re-measure → track regressions.

5. **Energy/performance-aware refactoring and pattern guidance**
   Provide rule-based and AI-assisted recommendations that explicitly consider energy/performance trade-offs, supported by low-energy pattern libraries and measurable validation.

6. **Standards-aligned reporting and export**
   Produce outputs aligned with SCI / ISO 21031 expectations and compatible with certification and procurement evidence needs, without inventing ad-hoc metrics.

7. **Multi-tenant allocation and confidence grading**
   Provide explicit allocation rules and confidence levels for shared infrastructure scenarios, ensuring transparency and comparability.

8. **Lifecycle-aware monitoring and regression detection**
   Track sustainability performance across versions, configuration changes, and operational evolution, supporting continuous monitoring rather than one-off audits.

9. **Modular, extensible architecture for sector adaptation**
   Enable domain-specific adapters (e.g., proprietary mainframe monitoring APIs, embedded heterogeneity) while maintaining a common evidence model.

10. **Workflow integration and adoption support**
    Integrate into existing engineering and DevOps toolchains, and support decision records and trade-off management where sustainability conflicts with performance/functional constraints.

GreenCode: State of the Art Review

New Business Models

The following new business models may emerge as a result of the GreenCode project

Optimisation-as-a-Service (OaaS) for legacy and complex systems
A managed service that profiles systems, identifies hotspots, recommends validated refactoring, and tracks improvements over time—prioritising measurable reductions in TCO and technical debt alongside sustainability outcomes.

Certification-ready evidence automation ("Assurance-as-a-Service")
A platform that generates traceable evidence bundles aligned to SCI and certification schemes, helping organisations move from disclosure to validated improvement while reducing greenwashing risk.

Continuous sustainability monitoring integrated into DevOps toolchains
Subscription tooling that embeds runtime measurement and regression detection into CI/CD (and operational monitoring), supporting continuous improvement expectations (e.g., annual remeasurement requirements in certification contexts).

Sector modules and "domain adapter marketplace"
A core platform plus paid sector-specific packs (mainframe adapters; embedded evaluation kits; cloud multi-tenant allocation modules), enabling broad applicability without losing domain depth.

Benchmarking and evaluation lab services
A revenue model around running controlled, reproducible evaluations for vendors and large enterprises—especially where measurement access is constrained or specialised (embedded boards, proprietary stacks).

Energy-aware refactoring models and pattern libraries as a data product
Curated, benchmark-annotated corpora and reusable low-energy pattern libraries that can be licensed to tool vendors or used internally to train/refine models for energy-aware code changes.

Infrastructure sustainability tooling aligned to post-2026 efficiency targets
Where relevant to partners and market demand, tooling that supports data-centre design/operations decisions to remain within stringent efficiency targets (as noted in the drafting prompts), provided this remains linked to software-driven evidence and does not duplicate mature hardware-only standards.

References

[1] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Florence, Italy, Jul. 2019, pp. 3645–3650, doi: 10.18653/v1/P19-1355.
[2] L. A. Barroso and U. Hölzle, “The case for energy-proportional computing,” *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007, doi: 10.1109/MC.2007.443.
[3] B. Priya, E. S. Pilli, and R. C. Joshi, “A Survey on Energy and Power Consumption Models for Greener Cloud,” *2013 3rd IEEE Int. Adv. Comput. Conf. (IACC)*, pp. 76–82, 2013, doi: 10.1109/iadcc.2013.6514198.
[4] A. Bergen, U. Stege, R. Desmarais, and S. Ganti, “Towards software-adaptive green computing based on server power consumption,” *Proc. 3rd Int. Work. Green Sustain. Softw.*, pp. 9–16, 2014, doi: 10.1145/2593743.2593745.
[5] A. Kipp, T. Jiang, J. Liu, M. Fugini, M. Vitali, B. Pernici, and I. Salomie, “Applying green metrics to optimise the energy consumption footprint of IT service centres,” *Int. J. Space-Based Situated Comput.*, vol. 2, no. 3, pp. 158–174, 2012, doi: 10.1504/IJSSC.2012.048897.
[6] A. Kipp, T. Jiang, M. Fugini, and I. Salomie, “Layered Green Performance Indicators,” *Futur. Gener. Comput. Syst.*, vol. 28, no. 2, pp. 478–489, 2012, doi: 10.1016/j.future.2011.05.005.
[7] A. Uchechukwu, K. Li, and Y. Shen, “Energy consumption in cloud computing data centers,” Int. J. Cloud Comput. Services Sci. (IJ-CLOSER), vol. 3, no. 3, pp. 156–173, Jun. 2014.
[8] C. Jiang, Y. Qiu, H. Gao, T. Fan, K. Li, and J. Wan, “An Edge Computing Platform for Intelligent Operational Monitoring in Internet Data Centers,” *IEEE Access*, vol. 7, pp. 133375–133387, 2019, doi: 10.1109/access.2019.2939614.
[9] Q. Chen, P. Grosso, K. van der Veldt, C. de Laat, R. Hofman, and H. Bal, “Profiling energy consumption of VMs for green cloud computing,” *2011 IEEE Ninth Int. Conf. Dependable, Auton. Secur. Comput.*, pp. 768–775, 2011, doi: 10.1109/dasc.2011.131.
[10] A. Noureddine, R. Rouvoy, and L. Seinturier, “A review of energy measurement approaches,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 47, no. 3, pp. 42–49, 2013, doi: 10.1145/2553070.2553077.
[11] A. Guldner *et al.*, “Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green Software Measurement Model (GSMM),” *Futur. Gener. Comput. Syst.*, vol. 155, pp. 402–418, 2024, doi: 10.1016/j.future.2024.01.033.
[12] D. Armstrong, K. Djemame, and R. Kavanagh, “Towards energy aware cloud computing application construction,” *Journal of Cloud Computing*, vol. 6, no. 1, p. 14, 2017, doi: 10.1186/s13677-017-0083-2.
[13] F. Almeida, J. Arteaga, V. Blanco, and A. Cabrera, “Energy Measurement Tools for Ultrascale Computing: A Survey,” *Supercomput. Front. Innov.*, vol. 2, no. 2, pp. 64–76, 2015, doi: 10.14529/jsfi150204.
[14] Hewlett Packard Enterprise, HPE Integrated Lights-Out (iLO) – Secure Server Management from Anywhere. [Online]. Available: https://www.hpe.com/de/de/hpe-integrated-lights-out-ilo.html [Accessed: 17 Jul. 2025]
[15] Dell Technologies, Dell OpenManage iDRAC – Secure and Remote Server Management. [Online]. Available: https://www.dell.com/de-de/lp/dt/open-manage-idrac [Accessed: 17 Jul. 2025]
[16] Drraghavendra, “A Holistic Framework for LLM-Driven Software Engineering Tasks: Leveraging GCP and Contact Center AI,” Stackademic (Medium), Sep. 23, 2024. [Online]. Available: https://blog.stackademic.com/a-holistic-framework-for-llm-driven-software-engineering-tasks-leveraging-gcp-and-contact-center-44c6a84a12e3
[17] N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, “RAPL in action: Experiences in using RAPL for power measurements,” ACM Trans. Model. Perform. Eval. Comput. Syst., vol. 3, no. 2, pp. 1–26, 2018, doi: 10.1145/3177754.
[18] V. M. Weaver *et al.*, “Measuring Energy and Power with PAPI,” *2012 41st Int. Conf. Parallel Process. Work.*, vol. 1, pp. 262–268, 2012, doi: 10.1109/icppw.2012.39.
[19] N. K. Chatradhi, “amd_energy.” [Online]. Available: https://github.com/amd/amd_energy [Accessed: Jan. 07, 2025.]
[20] Z. Yang, K. Adamek, and W. Armour, “Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC ’24,” *SC24: Int. Conf. High Perform. Comput., Netw., Storage Anal.*, vol. 00, pp. 1–17, 2024, doi: 10.1109/sc41406.2024.00028.

[21] NVIDIA Corporation, NVIDIA Management Library (NVML) – C-based API for Monitoring and Managing Data Center GPUs. [Online]. Available: https://developer.nvidia.com/management-library-nvml [Accessed: 17 Jul. 2025]
[22] J. Aslan, K. Mayers, J. G. Koomey, and C. France, "Electricity Intensity of Internet Data Transmission: Untangling the Estimates," *J. Ind. Ecol.*, vol. 22, no. 4, pp. 785–798, 2018, doi: 10.1111/jiec.12630.
[23] D. Mytton, D. Lundén, and J. Malmodin, "Network energy use not directly proportional to data volume: The power model approach for more reliable network energy consumption calculations," *J. Ind. Ecol.*, vol. 28, no. 4, pp. 966–980, 2024, doi: 10.1111/jiec.13512.
[24] C. Blakeney, "GPS-UP: A Better Metric For Comparing Software Energy Efficiency," Green Software Foundation, Dec. 14, 2021. [Online]. Available: https://greensoftware.foundation/articles/gps-up-a-better-metric-for-comparing-software-energy-efficiency/
[25] Bharany, S. et al., "A Systematic Survey on Energy-Efficient Techniques in Sustainable Cloud Computing," Sustainability, vol. 14, no. 10, Art. no. 6256, May 2022, doi: 10.3390/su14106256.
[26] "AI Code Generation: Boost Developer Productivity & Efficiency," getambassador.io. [Online]. Available: https://www.getambassador.io/blog/ai-code-generation-boost-developer-productivity. [Accessed: Apr. 15, 2025].
[27] Pouyeh Banijamali, Iffat Fatima, Patricia Lago, and Ilja Heitlager, "Unveiling key performance indicators for the energy efficiency of cloud data storage," in Proceedings of the 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pp. 222–229, 2024.
[28] Iffat Fatima, Markus Funke, and Patricia Lago, "Providing guidance to software practitioners: A framework for creating KPIs," IEEE Software, 2024.
[29] "SPEC Power and Performance Methodology," Standard Performance Evaluation Corporation, [Online]. Available: http://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf. [Accessed: 24-Aug-2025].
[30] Javier Mancebo, Hector O. Arriaga, Félix García, Ma Ángeles Moraga, Ignacio García-Rodríguez de Guzmán, and Coral Calero. *EET: a device to support the measurement of software consumption*. International Workshop on Green And Sustainable Software, 2018
[31] Mancebo, Javier, et al. "FEETINGS: Framework for Energy Efficiency Testing to Improve Environmental Goal of the Software." Sustainable Computing: Informatics and Systems 30 (2021): 100558. https://reader.elsevier.com/reader/sd/pii/S2210537921000494
[32] "HXi Series™ HX850i High-Performance ATX Power Supply — 850 Watt 80 Plus® Platinum Certified PSU (UK Plug)," Corsair, [Online]. Available: https://www.corsair.com/uk/en/p/psu/cp-9020073-uk/hxi-series-hx850i-high-performance-atx-power-supply-850-watt-80-plus-platinum-certified-psu-uk-plug-cp-9020073-uk. [Accessed: 24-Aug-2025].
[33] "Component-level energy monitoring for greener IT operations," Innovate UK Business Connect, [Online]. Available: https://iuk-business-connect.org.uk/opportunities/component-level-energy-monitoring-for-greener-it-operations/. [Accessed: 24-Aug-2025].
[34] See 15.
[35] "PLATYPUS: With Great Power comes Great Leakage," *PLATYPUS*, [Online]. Available: https://platypusattack.com/. [Accessed: 24-Aug-2025].
[36] Ahmad, I., et al., "Machine learning for energy estimation in software systems," *Sustainable Computing*, 2023.
[37] Sheng Li et al., "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," *MICRO*, 2009.
[38] N. L. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. doi: 10.1145/2024716.2024718.
[39] David Brooks, Vivek Tiwari, and Margaret Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *ISCA*, 2000.
[40] Sungpack Hong et al., "Green-Marl: A DSL for easy and efficient graph analysis," *ASPLOS*, 2012.
[41] Kerrison, S., and Eder, K., "Energy modelling of software for a hardware energy model," *ACM Transactions on Embedded Computing Systems*, 2015.

[42] Pallister, J., Kerrison, S., and Eder, K., “Data dependent energy modelling for worst case energy consumption analysis,” *ESWEEK*, 2015.
[43] K. N. Gregertsen and A. Skavhaug, “Implementation and usage of the new Ada 2012 execution time control features,” *Ada User Journal*, vol. 32, no. 4, pp. 265–272, Dec. 2011.
[44] Java Community Process, “JSR 1: Real-Time Specification for Java,” Java Specification Request, Final Version, Java Community Process, approved 15 Dec 1998, first release Jan 2002. [Online]. Available: https://jcp.org/en/jsr/detail?id=1 [Accessed: 24-Aug-2025].
[45] JDK Enhancement Proposal (JEP) 509: JFR CPU-Time Profiling (Experimental), OpenJDK, Release 25, created 4 Aug 2024; updated 31 Jul 2025. [Online]. Available: https://openjdk.org/jeps/509. [Accessed: 24-Aug-2025].
[46] Kepler Project, CNCF Sandbox, 2023.
[47] Cloud Carbon Footprint, Open Source Project, 2022.
[48] Google, “Carbon-Intelligent Computing,” 2021.
[49] Amazon and Google Cloud, “Regional Carbon Data APIs,” 2023.
[50] Intel Corporation, “Energy Analysis - Intel® VTune™ Profiler User Guide,” [Online]. Available: https://www.intel.com/content/www/us/en/docs/vtune-profiler/user-guide/2023-0/energy-analysis.html.
[51] Arm Ltd., “Arm Streamline User Guide (Version 7.2),” [Online]. Available: https://developer.arm.com/documentation/101816/0702/.
[52] DIMPACT, “The DIMPACT Methodology,” DIMPACT Publication, 21 Oct. 2022. [Online]. Available: https://dimpact.org/resource?resource=2.
[53] NVIDIA Corporation, “NVML API Reference,” [Online]. Available: https://docs.nvidia.com/deploy/nvml-api/nvml-api-reference.html.
[54] NVIDIA Corporation, “NVIDIA Data Center GPU Manager (DCGM) Documentation (latest),” [Online]. Available: https://docs.nvidia.com/datacenter/dcgm/latest/index.html.
[55] Advanced Micro Devices, Inc., “AMD System Management Interface (AMD SMI) documentation,” [Online]. Available: https://rocm.docs.amd.com/projects/amdsmi/en/latest/.
[56] Microsoft Corporation, “powrprof.h header (Power Profile functions) — Win32 apps,” [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/powrprof/.
[57] Cloud Carbon Footprint, “Cloud Carbon Footprint,” GitHub repository. [Online]. Available: https://github.com/cloud-carbon-footprint/cloud-carbon-footprint.
[58] Wang, F., et al. (2024). A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. *arXiv preprint arXiv:2411.03350*.
[59] Subramanian, S., et ak. (2025). Small language models (slms) can still pack a punch: A survey. *arXiv preprint arXiv:2501.05465*.
[60] Nguyen, C. V., et al (2024). A Survey of Small Language Models. *arXiv preprint arxiv: 2410.20011.*
[61] Q. Zhang, Z. Liu and S. Pan, "The Rise of Small Language Models," in IEEE Intelligent Systems, vol. 40, no. 1, pp. 30-37, Jan.-Feb. 2025, doi: 10.1109/MIS.2024.3517792.
[62] Hasan, M. M, et al. (2025). Assessing Small Language Models for Code Generation: An Empirical Study with Benchmarks. *arXiv preprint arXiv:2507.03160*.
[63] Souza, D., et al. (2025). Code generation with small language models: A deep evaluation on codeforces. *arXiv preprint arXiv:2504.07343*.
[64] Cho, J., et al. (2025). Self-Correcting Code Generation Using Small Language Models. *arXiv preprint arXiv:2505.23060*.
[65] Cong Li, et al. 2024. Understanding Code Changes Practically with Small-Scale Language Models. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24). Association for Computing Machinery, New York, NY, USA, 216–228. https://doi.org/10.1145/3691620.3694999
[66] Bappy, M. A. H., et al. (2025). Case Study: Fine-tuning Small Language Models for Accurate and Private CWE Detection in Python Code. *arXiv preprint arXiv:2504.16584*.
[67] Gheyi, R., et al. (2025). Evaluating the Effectiveness of Small Language Models in Detecting Refactoring Bugs. *arXiv preprint arXiv:2502.18454*.
[68] Durán, F. et al. (2024). Energy consumption of code small language models serving with runtime engines and execution providers. *arXiv preprint arXiv: 2412.15441*

[69] Shrinivasagouda Goudar, S. (2024). A Systematic Evaluation of Prompts for LLM-assisted Software Engineering Tasks. Master Thesis, Rheinland-Pfälzische Technische Universität Kaiserslautern (RPTU).
[70] Braberman, V. et al. (2024). Tasks people prompt: A taxonomy of LLM downstream tasks in software verification and falsification approaches. *arXiv preprint arXiv:2404.09384*.
[71] K. S. Cheung, M. Kaul, G. Jahangirova, M. R. Mousavi, and E. Zie, "Comparative Analysis of Carbon Footprint in Manual vs. LLM-Assisted Code Development," in *ResponsibleSE '25: Proceedings of the 1st International Workshop on Responsible Software Engineering* (co-located with FSE 2025), Trondheim, Norway, Jun. 2025, pp. 13–20, doi: 10.1145/3711919.3728678.
[72] Shazeer, N., et al. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ICLR 2017*. https://arxiv.org/abs/1701.06538

[73] Fedus, W., et al. (2022). Switch transformer: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*. https://arxiv.org/abs/2101.03961

[74] Jiang, A.Q., et al. (2024). Mixtral of experts. *Computation and Language*. https://arxiv.org/abs/2401.04088
[75] Cai, W., et al. (2024). A survey on mixture of experts in large language models. *arXiv preprint*. https://arxiv.org/abs/2407.06204
[76] Y. Zong, Y. Deng, and P. Nie, "Mix-of-Language-Experts Architecture for Multilingual Programming," *arXiv preprint* arXiv:2506.18923, Jun. 2025.
[77] Q. Wang, X. Han, J. Wang, L. Xing, Q. Hu, L. Zhang, C. Deng, and J. Feng, "MultiPL-MoE: Multi-Programming-Lingual Extension of Large Language Models through Hybrid Mixture-of-Experts," in *Findings of the Association for Computational Linguistics: EMNLP 2025*, Suzhou, China, Nov. 2025, pp. 12817–12828, doi: 10.18653/v1/2025.findings-emnlp.686.
[78] Y. Ding, J. Liu, Y. Wei, and L. Zhang, "XFT: Unlocking the Power of Code Instruction Tuning by Simply Merging Upcycled Mixture-of-Experts," in *Proc. 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024), Long Papers*, Bangkok, Thailand, Aug. 2024, pp. 12941–12955, doi: 10.18653/v1/2024.acl-long.699.
[79] F. Souza, T. Offermans, R. Barendse, G. Postma, and J. Jansen, "Contextual Mixture of Experts: Integrating Knowledge into Predictive Modeling," *arXiv preprint* arXiv:2211.00558, Nov. 2022.
[80] Liu, J., et al. (2024). A survey on inference optimization techniques for mixture of experts models. *arXiv preprint*. https://arxiv.org/abs/2412.14219
[81] Du, N., et al. (2022). GLaM: Efficient scaling of language models with mixture-of-experts. *International Conference on Machine Learning*. https://arxiv.org/abs/2112.06905
[82] H. Huang, N. Ardalani, A. Sun, K. Liu, H.-H. S. Lee, S. Bhosale, C.-J. Wu, and B. Lee, "Toward Efficient Inference for Mixture of Experts," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, 2024, doi: 10.52202/079017-2670.
[83] O. Balmau, A.-M. Kermarrec, R. Pires, and M. Vujasinovic, "Accelerating MoE Model Inference with Expert Sharding," in *Proc. 5th Workshop on Machine Learning and Systems (EuroMLSys '25)*, 2025, doi: 10.1145/3721146.3721940.
[84] Y. Li, P. Zheng, S. Chen, Z. Xu, Y. Lai, Y. Du, and Z. Wang, "Speculative MoE: Communication Efficient Parallel MoE Inference with Speculative Token and Expert Pre-scheduling," *arXiv:2503.04398*, 2025, doi: 10.48550/arXiv.2503.04398.
[8585] V. Stojković, Z. Zhang, L. Gan, S. Tiwari, E. Chen, and C. Wu, "DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency," in *Proc. IEEE Int. Symp. High-Performance Computer Architecture (HPCA)*, 2025, pp. 1348–1362, doi: 10.1109/HPCA61900.2025.00102
[86] . Fernandez, C. Na, V. Tiwari, Y. Bisk, S. Luccioni, and E. Strubell, "Energy Considerations of Large Language Model Inference and Efficiency Optimizations," in *Proc. 63rd Annual Meeting of the Association for Computational Linguistics (ACL), Long Papers*, 2025, pp. 32556–32569, doi: 10.18653/v1/2025.acl-long.1563
[87] S. Poddar, P. Koley, J. Misra, N. Ganguly, and S. Ghosh, "Towards Sustainable NLP: Insights from Benchmarking Inference Energy in Large Language Models," in *Proc. NAACL-HLT 2025, Long Papers*, 2025, pp. 12688–12704, doi: 10.18653/v1/2025.naacl-long.632
[88] Zhao, W., et al. (2021). Hierarchical routing mixture of experts. *2020 25th International Conference on Pattern Recognition (ICPR)*. https://ieeexplore.ieee.org/document/9412813/
[89] Mu, S., et al. (2025). A comprehensive survey of mixture-of-experts: Algorithms, theory, and applications. *arXiv preprint*. https://arxiv.org/abs/2503.07137

[90] G. Pinto, F. Castor, and Y. D. Liu, "Mining Questions About Software Energy Consumption," *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, ACM, pp. 22–31, 2014. doi: 10.1145/2597073.2597085.
[91] F. Alaswad and E. Poovammal, "Software quality prediction using machine learning", Materials Today: Proceedings Volume 62 Part 7, 2022, doi: 10.1016/j.matpr.2022.03.165
[92] S. Shafiq, A. Mashkoor, C. Mayr-Dorn and A. Egyed, "A Literature Review of Using Machine Learning in Software Development Life Cycle Stages", in IEEE Access, vol. 9, pp. 140896-140920, 2021, doi: 10.1109/ACCESS.2021.3119746
[93] F. L. Caram, B. R. D. O. Rodrigues, A. S. Campanelli and F. S. Parreiras, "Machine Learning Techniques for Code Smells Detection: A Systematic Mapping Study", in International Journal of Software Engineering and Knowledge Engineering, vol. 29, nr. 02, 2019, doi: 10.1142/S021819401950013X
[94] M. I. Azeem, F. Palomba, L. Shi, Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis", in Information and Software Technology, Volume 108, 2019, doi: 10.1016/j.infsof.2018.12.009
[95] H. Alsolai and M. Roper, "A Systematic Review of Feature Selection Techniques in Software Quality Prediction," 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates, 2019, pp. 1-5, doi: 10.1109/ICECTA48151.2019.8959566
[96] For example, as listed here: https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
[97] SonarSource S.A., "SonarQube," [Online]. Available: https://www.sonarqube.org/
[98] Snyk Ltd., "Snyk AI-powered Developer Security Platform," [Online]. Available: https://snyk.io/
[99] M. Martinez, M. Monperrus, and P. Baudry, "Automated Software Refactoring," *Journal of Systems and Software*, vol. 83, no. 1, pp. 3–17, Jan. 2010.
[100] OpenRewrite. "Large-scale automated source code refactoring," 2025. [Online]. Available: https://docs.openrewrite.org
[101] R. Gheyi, F. Ramalho, and A. Cavalcanti, "Evaluating the Effectiveness of Small Language Models in Detecting Refactoring Bugs," *arXiv preprint arXiv:2502.18454*, 2025.
[102] A. Nguyen-Duc et al., "Generative AI for Software Engineering: A Research Agenda," *arXiv preprint arXiv:2310.18648*, 2023.
[103] Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau, Germany
[104] Liu, Y. et al. (2025). LLMigrate: Transforming" Lazy" Large Language Models into Efficient Source Code Migrators. *arXiv preprint arXiv:2503.23791*.
[105] Hong, J. et al. (2025). Type-migrating C-to-Rust translation using a large language model. *Empirical Software Engineering*, *30*(1), 3.
[106] Eniser, H. F. et al. (2024). Towards translating real-world code with llms: A study of translating to rust. *arXiv preprint arXiv:2405.11514*.
[107] Pietrini, R. et al. (2024). Bridging Eras: Transforming Fortran legacies into Python with the power of large language models. In *2024 IEEE 3rd International Conference on Computing and Machine Intelligence (ICMI)* (pp. 1-5). IEEE.
[108] Chen, L. et al. (2024). Fortran2CPP: Automating Fortran-to-C++ Translation using LLMs via Multi-Turn Dialogue and Dual-Agent Integration. *arXiv preprint arXiv:2412.19770*.
[109] Fischer-Heselhaus, S. et al. (2023). AI vs. Dinosaurs–Automated Re-implementation of Legacy Mainframe Applications in Java by Combining Program Synthesis and GPT. In *International Conference on Mobile, Secure, and Programmable Networking* (pp. 205-221). Cham: Springer Nature Switzerland
[110] Kumar, A. et al. (2024). Automated validation of cobol to java transformation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (pp. 2415-2418).
[111] Kumar, J. et al. (2025). I Can't Share Code, but I need Translation--An Empirical Study on Code Translation through Federated LLM. *arXiv preprint arXiv:2501.05724*.
[112] Yuan, Z., et al. (2024). Transagent: An llm-based multi-agent system for code translation. *arXiv preprint arXiv:2409.19894*.
[113] *Cursor, "@ Mentions," Cursor Docs. [Online]. Available: https://cursor.com/docs/context/mentions*
[114] Microsoft, "Generate documentation using GitHub Copilot tools," *Microsoft Learn Module AZ-2007*, 2025. [Online]. Available: https://learn.microsoft.com/en-us/training/modules/generate-documentation-using-github-copilot-tools/

[115] doc-comments-ai, GitHub repository, maintained by Fynn Flügge, 2025. [Online]. Available: https://github.com/fynnfluegge/doc-comments-ai
[116] Luo, Q. et al. (2024). Repoagent: An llm-powered open-source framework for repository-level code documentation generation. *arXiv preprint arXiv:2402.16667*.
[117] Diggs, C. et al. (2024). Leveraging LLMs for legacy code modernization: Challenges and opportunities for LLM-generated documentation. *arXiv preprint arXiv:2411.14971*.
[118] Dvivedi, S. S. et al. (2024). A comparative analysis of large language models for code documentation generation. In *Proceedings of the 1st ACM international conference on AI-powered software* (pp. 65-73).
[119] Wang, J., et al. (2024). Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering*, *50*(4), 911-936.
[120] Zhang, Q., et al. (2025). Large Language Models for Unit Testing: A Systematic Literature Review. *arXiv preprint arXiv:2506.15227*.
[121] V. L. L. Lacoste, A. Ligozat, and T. J. Walsh, "CodeCarbon: Tracking Carbon Emissions of Machine Learning Models," *arXiv preprint arXiv:2104.10048*, 2021.
[122] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory Power Estimation and Capping," *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '10)*, pp. 189–194, 2010.
[123] D. Lo, L. Cherkasova, and A. Kumar, "PowerAPI: Energy-Aware Software Framework for Profiling and Optimizing Energy Consumption in Java Applications," *IEEE Int'l Conf. Green Computing and Communications (GreenCom)*, pp. 30–37, 2015
[124] M. Gottschalk, A. Bergel, and C. Brabrand, "GreenScaler: Energy-Aware Regression Testing," *Proceedings of the 44th International Conference on Software Engineering (ICSE '22)*, ACM/IEEE, pp. 1602–1614, 2022.
[125] S. Duan, et al. (2023). Leveraging Reinforcement Learning and Large Language Models for Code Optimization. *arXiv preprint arXiv:2312.05657*
[126] D. Huang, et al. (2024). Effi-Code: Unleashing Code Efficiency in Language Models *arXiv preprint arXiv:2410.10209*
[127] H. Peng, et al. (2024). Large Language Models for Energy-Efficient Code: Emerging Results and Future Directions. *arXiv preprint arXiv:2410.09241*
[128] T. Würthinger et al., "GraalVM: Run Programs Faster Anywhere," *Communications of the ACM*, vol. 62, no. 12, pp. 44–54, 2019.
[129] Microsoft, "Carbon Aware SDK," 2024. [Online]. Available: https://github.com/Green-Software-Foundation/carbon-aware-sdk
[130] EcoCode Project, "EcoCode: Detecting Energy Inefficient Patterns in Software Projects," Green Software Foundation, 2023. [Online]. Available: https://greensoftware.foundation
[131] H. Hivert et al., "Scaphandre: Energy Consumption Metering Agent for Virtualized Environments," *FOSDEM Conference*, 2021.
[132] D. A. Padua, K. Inoue, and K. Inoue, "EnergyVis: Visual Analytics for Energy Consumption of Software Systems," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 601–615, Feb. 2022.
[133] Anthony, L. F. W., et al., "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," *ICML Workshop on Tackling Climate Change with Machine Learning*, 2020.
[134] Henderson, P., et al., "Towards the systematic reporting of the energy and carbon footprints of machine learning," *Journal of Machine Learning Research*, 2020.
[135] Helmholtz AI, "Helmholtz AI," 2025. [Online]. Available: https://www.helmholtz.ai/
[136] S. Zhan et al., "MLPerf Power: Benchmarking the Energy Efficiency of Machine Learning Systems from µWatts to MWatts for Sustainable AI," arXiv preprint arXiv:2410.12032, Oct. 2024. [Online]. Available: https://arxiv.org/abs/2410.12032.
[137] Roy Schwartz, Jesse Dodge, Noah Anthony Smith, and Oren Etzioni, "Green AI," *Communications of the ACM*, volume 63, number 12, pages 54–63, 2020.
[138] Meghana Tedla, Shubham Kulkarni, and Karthik Vaidhyanathan, "Ecomls: A self-adaptation approach for architecting green ML-enabled systems," in Proceedings of the 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pp. 230–237, 2024.
[139] Vincenzo De Martino, Silverio Martínez-Fernández, and Fabio Palomba, "Do developers adopt green architectural tactics for ML-enabled systems? A mining software repository study," in Proceedings of the 2025

IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), pp. 135–139, 2025.
[140] Roberto Verdecchia, Patricia Lago, Christof Ebert, and Carol De Vries, "Green IT and green software," IEEE Software, vol. 38, no. 6, pp. 7–15, 2021.
[141] Mahsa Salmani, Armaghan Eshaghi, Enxiao Luan, and Sreenil Saha, "Photonic computing to accelerate data processing in wireless communications," Optics Express, vol. 29, no. 14, pp. 22299–22314, 2021.
[142] Patricia Lago, Roberto Verdecchia, Nelly Condori-Fernandez, Eko Rahmadian, Janina Sturm, Thijmen van Nijnanten, Rex Bosma, Christophe Debuysscher, and Paulo Ricardo, "Designing for sustainability: lessons learned from four industrial projects," in Advances and New Trends in Environmental Informatics: Digital Twins for Sustainability, pp. 3–18, Springer, 2020.
[143] Eames Trinh, Markus Funke, Patricia Lago, and Justus Bogner, "Sustainability integration of AI into the software development life cycle," in Proceedings of the 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pp. 199–206, 2024.
[144] Tom Cappendijk, Pepijn de Reus, and Ana Oprescu, "An exploration of prompting LLMs to generate energy-efficient code," in Proceedings of the 2025 IEEE/ACM 9th International Workshop on Green and Sustainable Software (GREENS), pp. 31–38, 2025.
[145] Dennis Junger, Max Westing, Christopher P Freitag, Achim Guldner, Konstantin Mittelbach, Kira Oberg¨oker, Sebastian Weber, Stefan Naumann, and Volker Wohlgemuth. Potentials of green coding–findings and recommendations for industry, education and science–extended paper. arXiv preprint arXiv:2402.18227, 2024.
[146] Rogardt Heldal, Ngoc-Thanh Nguyen, Ana Moreira, Patricia Lago, Leticia Duboc, Stefanie Betz, Vlad C. Coroamă, Birgit Penzenstadler, Jari Porras, Rafael Capilla, et al., "Sustainability competencies and skills in software engineering: An industry perspective," Journal of Systems and Software, vol. 211, p. 111978, 2024.
[147] Dominic Lammert, Stefanie Betz, Jari Porras, and Shola Oyedeji, "Sustainability in the software industry: a survey study on the perception, responsibility, and motivation of software practitioners," Digital ecosystems, blockchain evolution, and sustainable transformation (DEBEST), 2023.
[148] *See 121.*
[149] Embrace the future with carbon-aware computing: A path to sustainable cloud usage. (n.d.). *Argon & Co*. Retrieved July 21, 2025, from https://www.argonandco.com/en/news-insights/articles/embrace-the-future-with-carbon-aware-computing-a-path-to-sustainable-cloud-usage/
[150] Manimegalai, R., Sandhanam, S., Nandhini, A., & Pandia, P. (2024a, January 23). *Energy Efficient Coding Practices for Sustainable Software Development*. Proceedings of the First International Conference on Science, Engineering and Technology Practices for Sustainable Development, ICSETPSD 2023, 17th-18th November 2023, Coimbatore, Tamilnadu, India. https://eudl.eu/doi/10.4108/eai.17-11-2023.2342635
[151] Sofia, A. S., & Kumar, P. G. (2015). Implementation of energy efficient green computing in cloud computing. *International Journal of Enterprise Network Management*, *6*(3), 222–237. https://doi.org/10.1504/IJENM.2015.071135
[152] Kanso, H., Noureddine, A., & Exposito, E. (2024). A Review of Energy Aware Cyber-Physical Systems. *Cyber-Physical Systems*, *10*(1), 1–42. https://doi.org/10.1080/23335777.2022.2163298
[153] Linux Foundation, "Energy-Aware Scheduling (EAS)," 2021.
[154] Apple Developer Docs, "Energy Impact in Activity Monitor," 2023.
[155] A. Holczmann, S. Hsu, B. Johnson, and A. Hussain, "Green Software Patterns."[Online]. Available: https://patterns.greensoftware.foundation/ [Accessed: Oct. 09, 2024]
[156] P. A. Malla and S. Sheikh, "Analysis of QoS aware energy-efficient resource provisioning techniques in cloud computing," *Int. J. Commun. Syst.*, vol. 36, no. 1, 2023, doi: 10.1002/dac.5359.
[157] W. Yao, Z. Wang, Y. Hou, X. Zhu, X. Li, and Y. Xia, "An energy-efficient load balance strategy based on virtual machine consolidation in cloud environment," *Futur. Gener. Comput. Syst.*, vol. 146, pp. 222–233, 2023, doi: 10.1016/j.future.2023.04.014.
[158] S. Lambert, E. Caron, L. Lefevre, and R. Grivel, "Consolidation of virtual machines to reduce energy consumption of data centers by using ballooning, sharing and swapping mechanisms," *Futur. Gener. Comput. Syst.*, p. 107968, 2025, doi: 10.1016/j.future.2025.107968.
[159] J. Haddon, "Does the EU's approach to regulating data centres make sense?" *Capacity Media*, 10 Jun. 2024. [Online]. Available: https://www.capacitymedia.com/article/2dckl480ygjaprin8irk0/feature/does-the-eus-approach-to-regulating-data-centres-make-sense. [Accessed: 22.07.2025]

[160] Joint Research Centre, "The EU Code of Conduct for Data Centres – towards more innovative, sustainable and secure data centre facilities," *European Commission*, 5 Sep. 2023. [Online]. Available: https://joint-research-centre.ec.europa.eu/jrc-news-and-updates/eu-code-conduct-data-centres-towards-more-innovative-sustainable-and-secure-data-centre-facilities-2023-09-05_en. [Accessed: 22.07.2025]
[161] S. Frey, F. Fittkau, and W. Hasselbring, "Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud," *2013 35th Int. Conf. Softw. Eng. (ICSE)*, pp. 512–521, 2013, doi: 10.1109/icse.2013.6606597.
[162] F. Khan, "Optimized Deployment of Multi-cloud Applications via HTN Planning," 2023.
[163] Facebook Inc., Prophet – Forecasting at Scale. [Online]. Available: https://facebook.github.io/prophet/ [Accessed: 17 Jul. 2025]
[164] H. Kahil, S. Sharma, P. Välisuo, and M. Elmusrati, "Reinforcement learning for data center energy efficiency optimization: A systematic literature review and research roadmap," *Appl. Energy*, vol. 389, p. 125734, 2025, doi: 10.1016/j.apenergy.2025.125734.
[165] cite{morris2020infrastructure}
[166] Red Hat, Inc., Ansible Collaborative – A Gathering Space to Build Automation Skills and Success. [Online]. Available: https://www.redhat.com/en/ansible-collaborative [Accessed: 17 Jul. 2025]
[167] Progress Chef, Chef Software DevOps Automation Solutions. [Online]. Available: https://www.chef.io [Accessed: 17 Jul. 2025]

[168] Perforce Software, Puppet – Infrastructure Automation at Scale. [Online]. Available: https://www.puppet.com [Accessed: 17 Jul. 2025]
[169] HashiCorp, Terraform – Infrastructure as Code for Multi-Cloud Automation. [Online]. Available: https://developer.hashicorp.com/terraform [Accessed: 17 Jul. 2025]
[170] Amazon Web Services, Inc., AWS CloudFormation – Provision Infrastructure as Code. [Online]. Available: https://aws.amazon.com/de/cloudformation [Accessed: 17 Jul. 2025]
[171] Microsoft Corporation, Azure Resource Manager – Simplify Resource Management in Azure. [Online]. Available: https://azure.microsoft.com/de-de/get-started/azure-portal/resource-manager [Accessed: 17 Jul. 2025]
[172] Pulumi Corporation, Pulumi – Infrastructure as Code in Any Programming Language. [Online]. Available: https://www.pulumi.com [Accessed: 17 Jul. 2025]
[173] Amazon Web Services, Inc., AWS Cloud Development Kit (CDK) – Define Your Cloud Infrastructure Using Familiar Programming Languages. [Online]. Available: https://aws.amazon.com/de/cdk [Accessed: 17 Jul. 2025]
[174] HashiCorp, Cloud Development Kit for Terraform (CDKTF) – Define Infrastructure Using Familiar Programming Languages. [Online]. Available: https://developer.hashicorp.com/terraform/cdktf [Accessed: 17 Jul. 2025]
[175] D. Sokolowski, "Infrastructure as Code for Dynamic Deployments," in Proc. 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE '22), Singapore, Nov. 2022, pp. 1776–1780, doi: 10.1145/3540250.3558912.
[176] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba, "Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry," in Proc. 2019 IEEE Int. Conf. Software Maintenance and Evolution (ICSME), 2019, pp. 580–589, doi: 10.1109/ICSME.2019.00092.
[177] L. Harzenetter, U. Breitenbücher, T. Binz, and F. Leymann, "An Integrated Management System for Composed Applications Deployed by Different Deployment Automation Technologies," *SN Comput. Sci.*, vol. 4, no. 4, p. 370, 2023, doi: 10.1007/s42979-023-01810-4.
[178] F. Siebert, "Realtime garbage collection in the JamaicaVM 3.0," in Proc. 5th Int. Workshop Java Technol. Real-Time Embedded Syst. (JTRES '07), Vienna, Austria, Sep. 2007, pp. 94–103, doi: 10.1145/1288940.1288954.
[179] G. Contreras and M. Martonosi, "Techniques for Real-System Characterization of Java Virtual Machine Energy and Power Behavior," in Proc. 2006 IEEE Int. Symp. Workload Characterization (IISWC), San Jose, CA, USA, Oct. 2006, pp. 29–38, doi: 10.1109/IISWC.2006.302727.
[180] Green Coding Solutions, "Eco CI Energy Estimation," GitHub repository. [Online]. Available: https://github.com/green-coding-solutions/eco-ci-energy-estimation.
[181] R. Verdecchia, E. Cruciani, A. Bertolino, and B. Miranda, "Energy-Aware Software Testing," in Proc. IEEE/ACM 47th Int. Conf. Softw. Eng.: New Ideas and Emerging Results (ICSE-NIER), 2025.

[182] J. Pallister, S. J. Hollis, and J. Bennett, "Identifying Compiler Options to Minimize Energy Consumption for Embedded Platforms," *The Computer Journal*, vol. 58, no. 1, pp. 95–109, Jan. 2015, doi: 10.1093/comjnl/bxt129.
[183] F. Schuiki, F. Zaruba, T. Hoefler, and L. Benini, "Stream Semantic Registers: A Lightweight RISC-V ISA Extension Achieving Full Compute Utilization in Single-Issue Cores," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 212–227, Feb. 2021, doi: 10.1109/TC.2020.2987314.
[184] A. Tiwana and B. Konsynski, "Complementarities Between Organizational IT Architecture and Governance Structure," Inf. Syst. Res., vol. 21, no. 2, pp. 288–304, 2010, doi: 10.1287/isre.1080.0206.
[185] P. Hidas and Gartner, "Roadmap for Your Infrastructure - The Gartner Infrastructure Maturity Model.pdf,"
[186] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, and I. Hanschke, "Automation Processes for Enterprise Architecture Management," 2011 IEEE 15th Int. Enterp. Distrib. Object Comput. Conf. Work., pp. 340–349, 2011, doi: 10.1109/edocw.2011.19.
[187] T. Zaman, "A few weeks post Twitter-acquisition in 2022 we found 700 V100 gpus (pcie, lol) in the datacenter." Accessed: Oct. 17, 2024. [Online]. Available: https://x.com/tim_zaman/status/1815495006469365889?ref_src=twsrc%5Etfw
[188] M. Rezakhani, N. Sarrafzadeh-Ghadimi, R. Entezari-Maleki, L. Sousa, and A. Movaghar, "Energy-aware QoS-based dynamic virtual machine consolidation approach based on RL and ANN," Clust. Comput., vol. 27, no. 1, pp. 827–843, 2024, doi: 10.1007/s10586-023-03983-2.
[189] T. Binz, U. Breitenbucher, O. Kopp, and F. Leymann, "Automated Discovery and Maintenance of Enterprise Topology Graphs," 2013 IEEE 6th Int. Conf. Serv.-Oriented Comput. Appl., pp. 126–134, 2013, doi: 10.1109/soca.2013.29.
[190] T. Binz, C. Fehling, F. Leymann, A. Nowak, and D. Schumm, "Formalizing the Cloud through Enterprise Topology Graphs," 2012 IEEE Fifth Int. Conf. Cloud Comput., pp. 742–749, 2012, doi: 10.1109/cloud.2012.143.
[191] See 171.
[192] H. Holm, M. Buschle, R. Lagerström, and M. Ekstedt, "Automatic data collection for enterprise architecture models," Softw. Syst. Model., vol. 13, no. 2, pp. 825–841, 2014, doi: 10.1007/s10270-012-0252-1.
[193] See 168.
[194] qbilon GmbH, qbilon – Automated IT Landscape Discovery and Analysis. [Online]. Available: https://www.qbilon.io [Accessed: Jul. 17, 2025]
[195] Device42 (Freshworks Inc.), Device42 – Agentless Full-Stack IT Discovery & Dependency Mapping. [Online]. Available: https://www.device42.com [Accessed: 17 Jul. 2025]
[196] ServiceNow Inc., ServiceNow Discovery – IT Operations Management Product Page. [Online]. Available: https://www.servicenow.com/de/products/discovery.html [Accessed: 17 Jul. 2025]
[197] BMC Software Inc., BMC Helix Discovery – Automated IT Asset Discovery & Dependency Mapping, Jul. 10, 2025. [Online]. Available: https://www.bmc.com/it-solutions/bmc-helix-discovery.html [Accessed: 17 Jul. 2025]
[198] Faddom Ltd., Faddom – Agentless Application Dependency Mapping & Hybrid IT Discovery. [Online]. Available: https://faddom.com [Accessed: 17 Jul. 2025]
[199] Virima Inc., Virima – ITAM, ITSM & ITOM solutions with automated discovery, dependency mapping & ViVID™ visual impact display. [Online]. Available: https://virima.com [Accessed: 17 Jul. 2025]
[200] Lansweeper, Lansweeper – Technology Asset Intelligence Platform. [Online]. Available: https://www.lansweeper.com/de [Accessed: 17 Jul. 2025]
[201] Nagios Enterprises, Nagios – Open Source Monitoring and Network Management. [Online]. Available: https://www.nagios.org [Accessed: 17 Jul. 2025]
[202] Zabbix LLC, Zabbix – Enterprise-Class Open Source Network Monitoring Solution. [Online]. Available: https://www.zabbix.com [Accessed: 17 Jul. 2025]
[203] SolarWinds Corporation, SolarWinds – Observability, Database, and IT Service Management Solutions. [Online]. Available: https://www.solarwinds.com/de [Accessed: 17 Jul. 2025]
[204] Datadog, Inc., Datadog – Cloud Monitoring and Security Platform. [Online]. Available: https://www.datadoghq.com [Accessed: 17 Jul. 2025]
[205] Paessler AG, PRTG Network Monitor – All-in-one Network Monitoring Software. [Online]. Available: https://www.paessler.com/de/prtg [Accessed: 17 Jul. 2025]
[206] Icinga GmbH, Icinga – Open Source Monitoring for Complex IT Environments. [Online]. Available: https://icinga.com [Accessed: 17 Jul. 2025]

[207] Dynatrace, Inc., Dynatrace – AI-Powered Observability and Security Platform. [Online]. Available: https://www.dynatrace.com/de [Accessed: 17 Jul. 2025]
[208] Netdata, Inc., Netdata – Real-Time Infrastructure Monitoring. [Online]. Available: https://www.netdata.cloud [Accessed: 17 Jul. 2025]
[209] New Relic, Inc., New Relic – AI-Powered Full-Stack Observability & Security Platform. [Online]. Available: https://newrelic.com/de/ [Accessed: 17 Jul. 2025]
[210] Elastic N.V., Elastic Observability – Full-Stack Observability Built on an Open-Source Search AI Platform. [Online]. Available: https://www.elastic.co/observability [Accessed: 17 Jul. 2025]
[211] See 46
[212] See 123.
[213] Amazon Web Services, Amazon CloudWatch – Observability and Monitoring for AWS Cloud Resources, Service Level Agreement last updated Apr. 22, 2025. [Online]. Available: https://aws.amazon.com/de/cloudwatch [Accessed: 17 Jul. 2025]
[214] Microsoft Corporation, Azure Monitor – Modern Observability Tools for Cloud, Infrastructure, and Applications. [Online]. Available: https://azure.microsoft.com/de-de/products/monitor [Accessed: 17 Jul. 2025]
[215] Google LLC, Google Cloud Monitoring – Observability for Google Cloud and Hybrid Environments. [Online]. Available: https://cloud.google.com/monitoring?hl=de
[Accessed: 17 Jul. 2025]
[216] Alibaba Cloud, CloudMonitor – Real-Time Monitoring of Web Resources & Applications. [Online]. Available: https://www.alibabacloud.com/en/product/cloud-monitor?_p_lc=1 [Accessed: 17 Jul. 2025]
[217] C. Schumacher, "What's New In The Revised Blue Angel Criteria," *KDE Eco Blog*, Aug. 2024. [Online]. Available: https://eco.kde.org/blog/2024-08-26-revised-blue-angel-criteria/
[218] European Commission, *Corporate Sustainability Reporting Directive (CSRD)*, 2022.
[219] U.S. Securities and Exchange Commission, *Proposed Rule on Enhancement and Standardization of Climate-Related Disclosures*, 2022.
[220] C. Brinkmann, "Software Carbon Intensity: Reducing Software's CO_2 Footprint," *Tecnovy Insights*, Feb. 2025. [Online]. Available: https://tecnovy.com/en/software-carbon-intensity.
[221] ISO/IEC, "Information technology—Data centres—Key performance indicators—Part 1: Overview and general requirements," ISO/IEC 30134-1:2016, 2016. [Online]. Available: https://www.iso.org/standard/63450.html.
[222] Standard Performance Evaluation Corporation (SPEC), "The SERT® Suite—Design Document," Mar. 31, 2021. [Online]. Available: https://www.spec.org/sert2/SERT-designdocument-20210331.pdf.
[223] Reddi, V. et al., "MLPerf: An Industry Standard Benchmark for Machine Learning Performance," *IEEE Micro*, 2020 (extended for energy in 2023).
[224] ML CO2 Impact, "Machine Learning CO2 Impact Calculator," [Online]. Available: https://mlco2.github.io/impact/
[225] Hugging Face, "Displaying carbon emissions for your model," Hugging Face Hub Documentation. [Online]. Available: https://huggingface.co/docs/hub/en/model-cards-co2.
[226] S. Technologies (analysis), "GSF membership vs. global developer workforce," internal estimate, Aug. 2025 (based on SlashData global developer population report, 2025).
[227] Sustainable Games Alliance, "The games industry should lead the way on sustainability," *sustainablegamesalliance.org*, 2025. [Online]. Available: https://sustainablegamesalliance.org/
[228] Sustainable Games Alliance, "SGA Standard," *sustainablegamesalliance.org*, 2025. [Online]. Available: https://sustainablegamesalliance.org/standard/
[229] B. Lucks, "'Endboss' decarbonisation: How the Sustainable Games Alliance wants to reduce gaming's carbon footprint," *RESET.org*, May. 2025. [Online]. Available: https://en.reset.org/endboss-decarbonisation-how-the-sustainable-games-alliance-wants-to-reduce-gamings-carbon-footprint
[230] Blauer Engel, "Resource and Energy-Efficient Software Products (DE-UZ 215)," German Environment Agency, 2021. [Online]. Available: https://www.blauer-engel.de/en/productworld/software
[231] Labo Société Numérique, "A 'Responsible Digital' label for organizations," Jun. 2019, updated Sep. 2022. [Online]. Available: https://labo.societenumerique.gouv.fr/en/articles/a-digital-label-responsible-for-public-and-private-organizations/

[232] Pierre Fabre Group, "Pierre Fabre Laboratories Awarded Responsible Digital Label," Press Release, Feb. 2023. [Online]. Available: https://www.pierre-fabre.com/en/news/pierre-fabre-laboratories-awarded-responsible-digital-label
[233] Object Management Group, *Automated Source Code Resource Sustainability Measure (ASCRSM) 1.0*, OMG Standard, Apr. 2023. [Online]. Available: https://www.omg.org/spec/ASCRSM/1.0
[234] Green Code Lab, "Design4Green Challenge," 2015. [Online]. Available: https://www.design4green.org/
[235] Rajashekara, Kaushik and Koppera, Sharon (2024) 'Data and energy impacts of intelligent transportation—A review', MDPI - Publisher of Open Access Journals [Preprint]. Available at: Data and Energy Impacts of Intelligent Transportation—A Review .

[236] Wang, Xu *et al.* (2024) 'Moving Forward: A Review of Autonomous Driving Software and Hardware Systems', arXiv.org e-Print archive [Preprint]. Available at: Moving Forward: A Review of Autonomous Driving Software and... (Accessed: 28 May 2025).
[237] Katare, D. et al. (2023) 'A Survey on Approximate Edge AI for Energy Efficient Autonomous Driving Services'.
[238] Sudhakar, S., Sze, V. and Karaman, S. (2022) 'Data Centers on Wheels: Emissions From Computing Onboard Autonomous Vehicles', *IEEE Micro*, 43(1), pp. 29-39. Available at: https://doi.org/10.1109/MM.2022.3219803.

[239] Roussilhe, Gauthier, Ligozat, A.-L. and Quinton, S. (2023) 'A long road ahead: a review of the state of knowledge of the environmental effects of digitization', *Elsevier*, 62. Available at: https://doi.org/10.1016/j.cosust.2023.101296ï.

[240] Mukhanov, L., Nikolopoulos, D. S., & De Supinski, B. R. (2015, October). Alea: Fine-grain energy profiling with basic block sampling. In 2015 International Conference on Parallel Architecture and Compilation (PACT) (pp. 87-98). IEEE.

[241] Bunse, C., Höpfner, H., Roychoudhury, S., & Mansour, E. (2009, July). Choosing the best sorting algorithm for optimal energy consumption. In International Conference on Software and Data Technologies (Vol. 1, pp. 199-206). SCITEPRESS.

[242] H. Höpfner and C. Bunse, "Towards an Energy Aware DBMS: Energy Consumptions of Sorting and Join Algorithms," in *Proc. 21st Workshop Grundlagen von Datenbanken (GvD 2009)*, Rostock-Warnemünde, Germany, Jun. 2009, pp. 69–73, doi: 10.18453/rosdok_id00002192.
[243] Wysocki, W., Miciuła, I., & Plecka, P. (2025). Methods of Improving Software Energy Efficiency: A Systematic Literature Review and the Current State of Applied Methods in Practice. *Electronics*, *14*(7), 1331.
[244] *Data Centres and Data Transmission Networks*, International Energy Agency, last updated 11 July 2023. [Online]. Available: https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks
[245] J. C. T. Bieser, R. Hintemann, L. M. Hilty, and S. Beucker, "A review of assessments of the greenhouse gas footprint and abatement potential of information and communication technology," *Environmental Impact Assessment Review*, vol. 99, p. 107033, 2023, doi: 10.1016/j.eiar.2022.107033
[246] *Making software and data architectures more sustainable*, McKinsey & Company, published 2.7 years ago. [Online]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/tech-forward/making-software-and-data-architectures-more-sustainable
[247] See 22.
[248] A. Karabatic and M. Henriksson, "Comparison of energy usage and response time for web frameworks," M.S. thesis, School of Engineering, University of Borås, Sweden, 2022. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1769430/FULLTEXT01.pdf
[249] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva, "Energy efficiency across programming languages: How do energy, time, and memory relate?," in *Proc. 10th ACM SIGPLAN Int. Conf. Softw. Language Eng. (SLE)*, Vancouver, BC, Canada, Oct. 2017, pp. 256–267. [Online]. Available: https://dl.acm.org/doi/10.1145/3136014.3136031
[250] E. A. Santos, C. McLean, C. Solinas, and A. Hindle, "How does Docker affect energy consumption? Evaluating workloads in and out of Docker containers," *J. Softw. Syst.*, vol. 1, pp. 1–14, Jul. 2018. [Online]. Available: http://softwareprocess.ca/pubs/santos2018JSS-Docker-Energy.pdf

[251] M. Akour and M. Alenezi, "Reducing environmental impact with sustainable serverless computing," *Sustainability*, vol. 17, no. 7, art. 2999, Mar. 2025, doi: 10.3390/su17072999
[252] J. Stojkovic, N. Iliakopoulou, T. Xu, H. Franke, and J. Torrellas, "EcoFaaS: Rethinking the design of serverless environments for energy efficiency," in *Proc. 51st ACM/IEEE Ann. Int. Symp. Comput. Architecture (ISCA)*, Buenos Aires, Argentina, Jun. 29–Jul. 3, 2024, pp. 471–486, doi: 10.1109/ISCA59077.2024.00042.
[253] CO2.js, *The Green Web Foundation*, 2024. [Online]. Available: https://www.thegreenwebfoundation.org/co2-js/
[254] F. Irani, "CO2.js: An open library for digital carbon reporting," *Branch – Issue 4*, Summer 2022, Branch Magazine. [Online]. Available: https://branch.climateaction.tech/issues/issue-4/co2js/
[255] NTT DATA Corporation, "GSF Releases Software Carbon Intensity Specification v.1.0 for Scoring the Carbon Emissions of Software," *Press Release*, Dec. 23, 2022. [Online]. Available: https://www.nttdata.com/global/en/news/press-release/2022/december/gsf-releases-software-carbon-intensity-specification-v10-for-scoring-the-carbon-emissions
[256] T. Iwatsuka, D. Roux, and Y. Suenaga, "Upgrading to .NET 8: Inside the Carbon Aware SDK v1.4," *Green Software Foundation Blog*, Aug. 13, 2024. [Online]. Available: https://greensoftware.foundation/articles/upgrading-to-net-8-inside-the-carbon-aware-sdk-v1-4/
[257] A. Schmidt, G. Stock, R. Ohs, L. Gerhorst, B. Herzog, and T. Hönig, "carbond: An operating-system daemon for carbon awareness," *ACM SIGEnergy Energy Informatics Rev.*, vol. 4, no. 3, pp. 52–57, Sep. 2024, doi: 10.1145/3698365.3698374
[258] Sustainable Computing Systems, carbond, GitLab repository, 2024. [Online]. Available: https://gitlab.com/sustainable-computing-systems/carbond
[259] S. Werner, M. C. Borges, K. Wolf, and S. Tai, "A comprehensive experimentation framework for energy-efficient design of cloud-native applications," *arXiv preprint arXiv:2503.08641v1*, Mar. 11, 2025. [Online]. Available: https://arxiv.org/html/2503.08641v1
[260] G. Fitzmaurice, "Legacy IT infrastructure accounts for more than a third of enterprise power consumption, and it's creating a sustainability nightmare for IT leaders," *IT Pro*, Feb. 16, 2024. [Online]. Available: https://www.itpro.com/business/digital-transformation/legacy-it-infrastructure-accounts-for-more-than-a-third-of-enterprise-power-consumption-and-its-creating-a-sustainability-nightmare-for-it-leaders
[261] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about software energy consumption?," *IEEE Softw.*, vol. 33, no. 3, pp. 83–89, May–Jun. 2016, doi: 10.1109/MS.2016.66. [Online]. Available: https://ieeexplore.ieee.org/document/7155416/
[262] See 228.
[263] C. E. Freitag, M. Bernes-Lee, K. Widdicks, B. Knowles, G. S. Blair, and A. Friday, "The real and transformative impact of ICT: A critique of estimates, trends and regulations," *Patterns*, vol. 2, no. 9, Sep. 2021, doi: 10.1016/j.patter.2021.100353
[264] *Corporate Sustainability Reporting*, European Commission (Capital Markets Union & Financial Markets – Company Reporting), updated Feb. 26, 2025. [Online]. Available: https://finance.ec.europa.eu/capital-markets-union-and-financial-markets/company-reporting-and-auditing/company-reporting/corporate-sustainability-reporting_en
[265] *Green claims*, European Commission – Environment (Circular Economy), last updated Mar. 2023. [Online]. Available: https://environment.ec.europa.eu/topics/circular-economy/green-claims_en
[266] C. Groza, A. Dumitru-Cristian, M. Marcu, and R. Bogdan, "A developer-oriented framework for assessing power consumption in mobile applications: Android energy smells case study," *Sensors*, vol. 24, no. 19, art. 6469, Oct. 7, 2024, doi: 10.3390/s24196469
[267] *Media & Entertainment Video Games Sector*, U.S. Int'l Trade Administration, 2024. [Online]. Available: https://www.trade.gov/media-entertainment-video-games-sector
[268] C. Pérez, J. Verón, F. Pérez, M. Á. Moraga, C. Calero, and C. Cetina, "A comparative analysis of energy consumption between the widespread Unreal and Unity video game engines," *arXiv preprint arXiv:2402.06346*, 2024. [Online]. Available: https://arxiv.org/abs/2402.06346
[269] E. Mills, N. Bourassa, L. Rainer *et al.*, "Toward greener gaming: Estimating national energy use and energy efficiency potential," *Comput. Game J.*, vol. 8, pp. 157–178, 2019, doi: 10.1007/s40869-019-00084-2.

[270] "The State of Streaming Sustainability 2024," *Streaming Media*, published ~1.3 years ago. [Online]. Available: https://www.streamingmedia.com/Articles/Editorial/Featured-Articles/The-State-of-Streaming-Sustainability-2024-163113.aspx
[271] N. Rivero, "A third of humanity plays video games. This is what it means for the planet. To prevent greenhouse emissions and save gamers money on their utility bills, developers are looking for ways to make video games more energy efficient," *The Washington Post*, Dec. 12, 2024. [Online]. Available: https://www.washingtonpost.com/climate-solutions/2024/12/12/video-game-energy-efficiency-climate/
[272] *The SGA Standard*, Sustainable Games Alliance, 2024. [Online]. Available: https://sustainablegamesalliance.org/standard/
[273] *Corporate Sustainability Reporting*, European Commission, Finance — Capital Markets Union, updated 26 February 2025. [Online]. Available: https://finance.ec.europa.eu/capital-markets-union-and-financial-markets/company-reporting-and-auditing/company-reporting/corporate-sustainability-reporting_en
[274] P. van de Kamp, "CSRD and Green IT explained: How IT departments drive compliance," *Software Improvement Group*, Nov. 11, 2024. [Online]. Available: https://www.softwareimprovementgroup.com/csrd-and-green-it-explained/
[275] *See 247.*
[276] U.S. Securities and Exchange Commission, "SEC Adopts Rules to Enhance and Standardize Climate-Related Disclosures for Investors," *Press Release No. 2024-31*, Mar. 6, 2024. [Online]. Available: https://www.sec.gov/newsroom/press-releases/2024-31
[277] United Nations Environment Programme, Playing for the Planet 2023, UNEP, 2023. [Online]. Available: https://wedocs.unep.org/bitstream/handle/20.500.11822/45229/playing_for_the_planet_2023.pdf?sequence=3&isAllowed=y
[278] B. Hauglie, "Xbox is now the first carbon aware console, update rolling out to everyone soon," *Xbox Wire*, Jan. 11, 2023. [Online]. Available: https://news.xbox.com/en-us/2023/01/11/xbox-carbon-aware-console-sustainability/
[279] K. Mayers, "PlayStation 5 and its energy efficiency," *Sony Interactive Entertainment Blog*, Oct. 28, 2021. [Online]. Available: https://sonyinteractive.com/en/news/blog/playstation-5-and-its-energy-efficiency/
[280] T. Patterson, "A progress update on our carbon reduction goals at Xbox," *Xbox Wire*, Sept. 23, 2024. [Online]. Available: https://news.xbox.com/en-us/2024/09/23/xbox-carbon-reduction-sustainability-update-2024/
[281] Energy Efficiency of Games Consoles, European Commission — Energy-efficient Products Voluntary Agreement, version 1.0, Apr. 22, 2015. [Online]. Available: https://energy-efficient-products.ec.europa.eu/product-list/games-consoles_en
[282] Microsoft, "Halo Infinite case study," *Xbox Developer Sustainability Toolkit Case Studies*, Mar. 12, 2024. [Online]. Available: https://learn.microsoft.com/en-us/gaming/sustainability/case-studies/case-studies-halo
[283] Microsoft, "Fortnite case study," *Xbox Developer Sustainability Toolkit Case Studies*, 2024. [Online]. Available: https://learn.microsoft.com/en-us/gaming/sustainability/case-studies/case-studies-fortnite
[284] Microsoft, "The Elder Scrolls Online case study: new environmental sustainability features," *Xbox Developer Sustainability Toolkit Case Studies*, June 2024. [Online]. Available: https://learn.microsoft.com/en-us/gaming/sustainability/case-studies/case-studies-elder-scrolls-online/
[285] Microsoft, "Xbox is the first console platform to release dedicated energy consumption and carbon emissions measurement tools designed for (and with) game creators," *Microsoft Game Dev Blog*, Mar. 22, 2023. [Online]. Available: https://developer.microsoft.com/en-us/games/articles/2023/03/gdc-2023-xbox-sustainability-toolkit-for-game-creators/
[286] See 251.
[287] See 252.
[288] D. Keeney, *Virtual Production's Role in Carbon Reduction and Net Zero Production in the Screen Industries*, Future Observatory Cultural Policy Fellowship Report, Studio Ulster at Ulster University, Nov. 30, 2023. [Online]. Available: https://futureobservatory.org/files/dcmsreports/futureobservatory_culturalpolicyreport_studioulster.pdf
[289] J. Vincent, "AV1 is supposed to make streaming better, so why isn't everyone using it?," *The Verge*, Apr. 3, 2025. [Online]. Available: https://www.theverge.com/tech/635020/av1-streaming-netflix-youtube-google-adoption

[290] E. Levrel, "World Energy Saving Day 2024: Reducing the Footprint of Video Streaming," *Broadpeak Blog*, Oct. 21, 2024. [Online]. Available: https://broadpeak.tv/blog/world-energy-saving-day-in-streaming/
[291] Carbon Trust, "Carbon impact of video streaming," White Paper, 2021. [Online]. Available: https://ctprodstorageaccountp.blob.core.windows.net/prod-drupal-files/documents/resource/public/Carbon-impact-of-video-streaming.pdf
[292] S. Georgiou, *Energy and Run-Time Performance Practices in Software Engineering*, Ph.D. dissertation, Dept. of Management Science and Technology, Athens Univ. of Economics and Business, Athens, Greece, 2021. [Online]. Available : https://stefanos1316.github.io/my_curriculum_vitae/phd_thesis.pdf
[293] T. Eilam, "Towards Transparent and Trustworthy Cloud Carbon Accounting," keynote presentation, Middleware '21, IBM T. J. Watson Research Center, New York, USA, 2021. [Online]. Available: https://middleware-conf.github.io/2021/pdf/middleware21keynotes-final3.pdf
[294] Microsoft, "Call of Duty: Modern Warfare II and Call of Duty: Warzone case study," *Xbox Developer Sustainability Toolkit Case Studies*, Mar. 12, 2024. [Online]. Available: https://learn.microsoft.com/en-us/gaming/sustainability/case-studies/case-studies-cod
[295] C. Asher, "Playing dangerously: The environmental impact of video gaming consoles," *Mongabay*, Oct. 25, 2022. [Online]. Available: https://news.mongabay.com/2022/10/playing-dangerously-the-environmental-impact-of-video-gaming-consoles/ news.mongabay.com+9
[296] R. Evans and J. Gao, "DeepMind AI reduces Google data centre cooling bill by 40%," *DeepMind Blog*, Jul. 20, 2016. [Online]. Available: https://deepmind.google/discover/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-by-40/
[297] F. Polarczyk, "Driving Energy Efficiency in Media Streaming: Insights from Industry Leaders," *Accedo Blog*, Sept. 6, 2024. [Online]. Available: https://www.accedo.tv/insights-and-news/driving-energy-efficiency-in-media-streaming-insights-from-industry-leaders/
[298] Greener Service Principles, "Greener Service Principles," *Sustainable by design*, ver. 1.0, updated Mar. 14, 2025. [Online]. Available: https://greenerservices.github.io/.
[299] R. Cottrell, H. Arpalikli, and M. Allers, "Adding sustainability to the Government Design Principles," *Services in government* (GOV.UK blog), Apr. 2, 2025. [Online]. Available: https://services.blog.gov.uk/2025/04/02/adding-sustainability-to-the-government-design-principles/
[300] Ministerio para la Transición Ecológica y el Reto Demográfico (MITECO), "Contratación Pública Ecológica," *MITECO – Planes, estrategias y hojas de ruta*. [Online]. Available: https://www.miteco.gob.es/es/ministerio/planes-estrategias/plan-de-contratacion-publica-ecologica.html
[301] Danish Agency for Digital Government, "Digital Green Transition." [Online]. Available: https://en.digst.dk/digital-transformation/digital-green-transition/
[302] See 121.
[303] IBM(2021). Application Modernization on the Mainframe. https://www.ibm.com/downloads/cas/7BJPNGND?_ga=2.72317459.1696084635.1710142763-2067957453.1707311480