# AI-Assisted Optimization of Software Energy Consumption

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Software Engineering
July 2025
MD Shah Noor Khan

UNIVERSITY OF TURKU
Department of Computing

MD Shah Noor Khan: AI-Assisted Optimization of Software Energy Consumption

Master of Science (Tech) Thesis, 92 p., 17 app. p.
Software Engineering
July 2025

---

The escalating energy footprint of software systems demands urgent intervention, particularly in widely deployed platforms like WordPress, which powers 43.5% of global websites. This thesis develops an AI-assisted methodology to optimize energy consumption in WordPress's PHP backend, emphasizing rigorous empirical validation. A controlled lab environment (Odroid H3+ rig with PowerGoblin instrumentation) enabled 100 Hz sample-rate energy profiling of computational inefficiencies, while a Random Forest classifier prioritized optimization targets based on execution time, energy, and power metrics. Heuristic-guided refactoring generated context-specific optimizations, such as replacing nested loops with hash-map logic and implementing transient caching. Experimental results demonstrate 99.44% energy reduction for algorithmic bottlenecks (e.g. loop patterns) and statistically significant improvements in memory-intensive operations. The methodology validates AI-driven prioritization as a robust framework for identifying and mitigating energy hotspots in PHP-based systems, advancing sustainable software engineering practices.

Keywords: energy efficiency, PHP optimization, WordPress, AI-assisted refactoring, sustainable software, empirical study.

# Contents

# 1 Introduction

Information and Communication Technology (ICT) infrastructures consumed approximately 4% of global electricity in 2020—a share that is projected to continue rising through 2030 as data centers, 5G networks, and other digital services rapidly expand [1], [2]. Volatility in energy markets and increasing concerns over the environmental impact of critical raw material extraction and resource use further compound the urgency to curtail ICT's environmental footprint [3], [4]. In response, regulatory frameworks such as the EU's Corporate Sustainability Reporting Directive (CSRD) and the International Sustainability Standards Board (ISSB) mandate comprehensive emissions reporting, including those attributable to ICT operations [5], [6]. These developments have catalyzed a surge of innovation in green and sustainable software engineering, with a focus on designing applications and systems that minimize energy consumption, optimize resource utilization across hardware and networks, and enable transparent measurement and management of software-related emissions [7], [8].

The concept of sustainable software is by no means new. Technical debt and economic sustainability have long been recognized challenges in software engineering, while environmental sustainability has garnered significant attention over the past decade [9], [10]. Foundational studies in the early 2000s examined ICT's environmental impacts, and subsequent work in the 2010s proposed methods for crafting more sustainable applications [10], [11]. Nevertheless, software energy demand continues

to escalate, underscoring the need for systematic optimization and integration of sustainability practices throughout the software development lifecycle [9], [10]. Historically, performance tuning has been viewed as prohibitively complex or costly compared to simply upgrading hardware—a phenomenon encapsulated by Wirth's Law, which observes that software bloat often offsets gains in hardware speed [12].

## 1.1 Challenges in Sustainable Software Engineering

Despite growing awareness, the development of energy-efficient software is still limited by the absence of standardized interfaces, metrics, and tools for sustainability assessment. Many practitioners report uncertainty about how to measure and reduce software energy consumption, as much of the existing research remains focused on theoretical frameworks rather than providing actionable solutions for day-to-day development [9]. These gaps present significant challenges to embedding sustainability into established development workflows. To address these issues, this thesis introduces these challenges by introducing practical, lightweight instrumentation techniques and by integrating energy-per-transaction and carbon-intensity metrics directly into standard build processes, ensuring that energy efficiency becomes a visible and manageable criterion alongside functionality and performance.

## 1.2 Objectives of the Study

WordPress, built primarily on PHP, is the world's most widely used content management system—powering 43.5% of all websites as of June 2025 [13]. Given its ubiquity and the central role of PHP in its architecture, even modest improvements in code efficiency can have a significant aggregate impact on global web energy consumption. Crucially, the PHP backend serves as an integral component and persistent source of energy consumption across all WordPress usage scenarios – including mobile ac-

cess, desktop browsing, and API interactions – making its optimization universally relevant. The primary objective of this work is to establish an empirically grounded, AI-assisted methodology for identifying and optimizing energy-intensive PHP code segments within WordPress environments. Specific goals include capturing reliable performance and energy metrics under controlled experimental conditions, employing machine learning to prioritize inefficiencies, generating heuristic-driven refactoring recommendations, and validating improvements through statistical analysis.

## 1.3 Research Questions

This thesis is guided by the following research questions, which collectively shape its methodology and evaluation:

- **RQ1:** Which backend PHP functions in WordPress contribute most significantly to excessive energy consumption?
- **RQ2:** Through what experimental setup can execution time, processor energy, and average processor power be captured and evaluated with fidelity?
- **RQ3:** Can a Random Forest classifier, trained on combined performance and energy features, reliably prioritize inefficient code segments?
- **RQ4:** What categories of refactoring recommendations can be derived automatically via AI-driven heuristics?
- **RQ5:** To what extent are these AI-assisted optimization techniques generalizable to other WordPress workloads or similar PHP-based systems?

## 1.4 Rationale for Research Question Selection

The five research questions were chosen to reflect a complete optimization pipeline: from identifying "hot" code segments (RQ1), through instrumenting and measuring energy (RQ2), to applying AI for prioritization (RQ3), generating actionable recommendations (RQ4), and assessing broader applicability (RQ5). This progression ensures that each stage—profiling, classification, recommendation, and validation—is systematically addressed, yielding both a methodological framework and practical guidance for sustainable PHP development.

## 1.5 Systematic Literature Review Methodology

A systematic literature review (SLR) framed by Evidence-Based Software Engineering (EBSE) principles underpins the design of the four optimization techniques explored in this thesis [14]. Conducted solely by the author to maintain methodological consistency, this SLR acknowledges the increased risk of bias inherent in single-researcher reviews. While rigorous protocols and transparent reporting were followed, dual-reviewer screening is recognized as the gold standard for minimizing bias [15]. The detailed findings of this systematic literature review are presented in Chapter 2. The review covered publications from 2000 through June 2025 to ensure both foundational and current perspectives, and it provided an unbiased synthesis of advances in green IT, energy-aware algorithms, and sustainable computing. Search terms—organized by concept in Table 1.1—combined descriptors of sustainable software engineering, energy-efficiency framing, profiling and measurement tools, optimization techniques, AI-driven prioritization, sustainable computing context, and development workflows.

The Boolean fragments that were systematically applied across five core repositories (IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, Scopus)

Table 1.1: Boolean-query components by concept

| **Concept** | **Search Terms** |
|---|---|
| Platform | WordPress OR "PHP CMS" |
| Energy-Efficiency Framing | "Energy efficiency" OR "Energy consumption" OR "Power usage" OR "Carbon intensity" OR "Green ICT" |
| Profiling and Measurement | "Query Monitor" OR "Intel Power Gadget" OR "SmartPower 3" OR "PowerGoblin" OR "Energy profiling" |
| Optimization Techniques | "Loop optimization" OR "Memory buffering" OR "Object reuse" OR "SQL tuning" OR "Green coding" |
| AI-Driven Prioritization | "Random Forest" OR "Machine learning" OR "Heuristic recommendation" OR "AI-assisted optimization" |
| Sustainable Computing Context | "Sustainable software engineering" OR "Green software practices" OR "Software sustainability metrics" |
| Development Workflow | PHP OR Plugin OR Backend OR "Continuous Integration" OR "CI/CD" |
| Excluded Topics | AND NOT Mobile OR iOS OR Android OR Embedded OR IoT OR Cryptocurrency OR "Building energy" |

are listed in Table 1.1. To capture emergent industry standards and policy frameworks, targeted ad-hoc searches supplemented these with grey literature from European Commission portals, standards bodies (ETSI), industry consortia (Green Software Foundation), preprint archives (arXiv), and technical blogs (TechTarget). Database selection encompassed these technical bibliographies as well as authoritative portals such as the Green Software Foundation and arXiv preprints. Search terms combined descriptors of sustainable software engineering and energy metrics. The inclusion criteria mandated that sources provide empirical evidence, established frameworks, or validated models addressing software energy efficiency. Study evaluation employed a three-tiered screening protocol: (1) Title-level assessment eliminated manifestly irrelevant works (e.g. hardware-centric optimizations or non-ICT domains); (2) Abstract review verified preliminary alignment with software energy

efficiency objectives; and (3) Full-text analysis rigorously validated methodological quality and adherence to inclusion standards. Non-peer-reviewed and non-technical publications were systematically excluded, except for authoritative documents from recognized governmental or standards bodies (e.g. European Commission directives, IEA reports, ETSI standards), which underwent immediate full-text appraisal due to their foundational policy relevance. Quality assessment prioritized methodological robustness, empirical validity, and temporal relevance, with benchmark studies spanning Nurmivaara's systematic review of green software practices [16], the comprehensive industry-academic analysis of sustainable development strategies [17], Cruz et al.'s research agenda for sustainable AI systems [18], and the empirical evaluation of green computing implementations [19]. This SLR revealed prevailing energy-optimization patterns—loop refactoring, memory buffering, and object pooling—that directly informed the heuristic recipes and test-case implementations developed in later chapters.

## 1.6 Delimitations

This study is strictly limited to the energy efficiency of WordPress's PHP backend code. Any aspects that lie outside this technical focus have been intentionally omitted. Specifically, front-end optimizations (CSS/JavaScript), plugin or theme configuration beyond our three test cases, and domains such as mobile or embedded systems—where power constraints and execution environments differ fundamentally from server-side PHP—are not examined.

## 1.7 Thesis Organization

The remainder of this thesis is arranged as follows. Chapter 2 reviews fundamental concepts and existing energy-optimization strategies for PHP applications, provid-

ing a systematic literature review of the field. Chapter 3 explores various AI-driven energy optimization techniques and their foundational principles in software systems, leading to the selection of the core methodology for this thesis. Chapter 4 details the comprehensive experimental methodology, including the design of the dual-environment profiling setup, data processing and feature engineering, the AI classification and heuristic optimization framework, and the rigorous validation protocols employed. Chapter 5 presents the empirical results from the experimental validation, offering a detailed analysis of the observed energy reductions, performance improvements, and statistical findings. Finally, Chapter 6 concludes the thesis by summarizing the research, addressing the research questions, outlining the study's contributions, discussing its limitations, and proposing directions for future work.

## 1.8 Declaration of Generative AI Use

This thesis employed AI-based tools exclusively for auxiliary tasks under University of Turku's guidelines. The specific tools, their applications, and verification protocols ensuring research integrity are detailed below.

ChatGPT (gpt-4-turbo, OpenAI) and Scopus AI (Literature Review Assistant) were utilized. Both tools were selected for their specific capabilities in academic writing support and literature processing respectively, with strict adherence to ethical guidelines.

ChatGPT served three primary functions: (1) sentence polishing and clarity enhancement across all chapters; (2) paraphrasing and text restructuring in Sections 1.1–1.7 and Chapters 2–3; and (3) clarifying methodological explanations in Sections 4.1–4.2. Additionally, it standardized section formatting document-wide. Scopus AI was exclusively applied to literature synthesis and reorganization of Systematic Literature Review findings in Section 2.4.

Three safeguards were implemented for verification protocols: cross-validation of all AI-generated content against primary sources; manual verification of terminological consistency; and strict prohibition against processing experimental data or confidential materials.

The fundamental research design, experimental methodology, data analysis, and scientific conclusions remain exclusively human-generated. AI tools were strictly limited to the auxiliary writing support tasks specified above.

# 2 Foundations of Green ICT and Sustainable Software

Green Information and Communication Technology (Green ICT) is an interdisciplinary domain focusing on reducing the environmental impact of Information and Communication Technology (ICT) systems. It addresses issues like energy consumption, e-waste—the fastest-growing waste category globally and carbon emissions, promoting sustainable technological growth [20]. As digital infrastructure expands globally, Green ICT becomes critical in achieving environmentally responsible development. This section elaborates on the definition, importance, strategies, and applications of Green ICT and its role in the broader sustainability agenda.

The urgency of Green ICT is underscored by the ICT sector's significant environmental footprint. According to a systematic literature review, the ICT sector is responsible for between 1.8% and 3.9% of global greenhouse gas emissions, with projections suggesting emissions could double by 2030 without intervention [21]. This growth is driven by rising energy demands from data centers, AI-driven applications, and the proliferation of connected devices [22], [23]. The European Union has responded with initiatives like the European Green Deal and ecodesign regulations to standardize energy efficiency and lifecycle assessments [20], [24].

## 2.1 Conceptual Foundations of Green ICT

The rapid expansion of digital technologies has brought both unprecedented opportunities and new environmental challenges. As society becomes increasingly reliant on information and communication technology (ICT), there is a growing recognition of the need to address its ecological impact. Green ICT has emerged as a multidisciplinary field dedicated to reducing the environmental footprint of digital infrastructure and services. This section introduces the foundational concepts, motivations, and strategies that underpin Green ICT, setting the stage for a deeper exploration of its definitions, imperatives, technical approaches, and evolving trends.

### 2.1.1 Defining Green ICT

Green ICT encompasses practices, policies, and technologies aimed at minimizing the environmental footprint of ICT systems [25]. It includes energy-efficient hardware, sustainable software development, eco-friendly data centers, and responsible e-waste management. The concept extends beyond operational efficiency, encompassing the entire lifecycle of ICT products—from design to disposal [25]. Green ICT is a young and pioneering field that focuses on minimizing the negative impact of ICT on the environment and optimizing its positive impact through various activities and strategies.

Green ICT refers to the environmentally conscious design, production, and use of ICT systems to reduce their negative ecological impact. This includes energy-efficient hardware, renewable energy integration, and software solutions optimized for minimal resource consumption [26]. Green ICT encompasses practices aimed at minimizing the environmental footprint of ICT systems, focusing on energy efficiency and sustainability throughout the entire lifecycle of ICT products—from raw material extraction and manufacturing to disposal and recycling [22].

### 2.1.2 Environmental and Economic Imperatives

The ICT sector significantly contributes to global energy consumption and carbon emissions. Studies project that the ICT industry's share of global electricity usage could rise from 7% in 2020 to 13% by 2030. This increase is driven by the rapid expansion of cloud computing, AI, and IoT devices [27], [28]. Without proactive measures, ICT could become a major obstacle to achieving global sustainability targets, particularly given the sector's growing reliance on energy-intensive infrastructure like data centers and 5G networks [27], [29]. Sustainable e-waste management practices, aligned with the EU's Circular Economy Action Plan, ensure materials are reused or recycled, minimizing ecological harm [29].

Reasons for prioritizing Green ICT include environmental responsibility, economic efficiency, regulatory compliance, and corporate social responsibility. It aims to align technological growth with sustainable development by minimizing the environmental footprint of information and communication technologies.

One of the core objectives of Green ICT is to reduce greenhouse gas emissions and minimize electronic waste, the fastest-growing waste category globally [29]. Energy-efficient technologies, such as optimized server operations and modular device design, can mitigate the environmental impact of data centers and networks, which currently account for 15% of ICT's electricity use and are projected to grow [27], [28].

Adopting Green ICT practices reduces operational costs through energy savings. For example, the EN 305 200 Series standards for energy-efficient ICT networks enable organizations to lower energy bills by up to 30% through dynamic workload management and hardware optimization [29]. Streamlining data center operations with AI-driven cooling systems has been shown to cut energy use by 40%, enhancing financial stability [27].

Governments and international bodies, such as the EU, are enforcing policies like the European Green Deal and ecodesign regulations (e.g. Regulation 2019/424)

to standardize energy efficiency in ICT infrastructure [29]. Compliance with these frameworks not only avoids penalties but also aligns businesses with global climate neutrality goals, such as reducing ICT-related emissions by 15–20% by 2030 [29].

For businesses, Green ICT is a tangible way to demonstrate corporate social responsibility by aligning operations with the principles of sustainability and social well-being. Green ICT allows companies to demonstrate CSR through renewable energy adoption and eco-friendly product design. The EU's focus on "green data services" and lifecycle assessments (e.g. ITU-T L.1440) fosters trust among stakeholders by linking sustainability to brand reputation [29]. Consumers increasingly favor businesses that align with the UN's Sustainable Development Goals (SDGs), making Green ICT a competitive advantage [29].

### 2.1.3 Core Technical Strategies

Green ICT strategies are designed to address the growing environmental impact of the ICT sector through technological and operational innovation. One of the most impactful strategies involves the development of energy-efficient hardware that minimizes power usage without compromising performance.

Modern processors, servers, and electronic devices are increasingly being engineered with lower power requirements to reduce total energy consumption. For instance, advances in chip manufacturing—such as extreme ultraviolet (EUV) lithography and 3D transistor architectures—allow processors to achieve high performance with significantly less energy use [30], [31]. Similarly, hardware components that implement dynamic voltage and frequency scaling (DVFS) enable systems to adjust their energy usage based on real-time workload demands, in alignment with standards such as the EN 305 200 Series [32]. These developments not only conserve electricity but also lower thermal output, extending device longevity and reducing cooling requirements. Such innovations are central to minimizing the environmental

footprint of ICT infrastructure.

Alongside hardware advances, software efficiency plays a crucial role in the overall energy profile of computing systems. Green ICT emphasizes reducing computational complexity and promoting energy-aware design in software development. Efficient code implementation, including the use of streamlined algorithms and resource-optimized logic, enables applications to complete tasks more rapidly and with reduced energy consumption [16]. For example, database optimization through indexing or elimination of redundant computations can significantly reduce server load and power usage. These practices ensure that software performance is not only fast and scalable but also consistent with sustainability principles.

Improving the environmental performance of data centers is another critical priority within Green ICT initiatives. As data centers consume an estimated 1.5% of global electricity [33], transforming these facilities into energy-efficient operations is essential. Key strategies include transitioning to renewable energy sources such as solar and wind power, often supported by policy frameworks like the EU's Climate-Neutral Data Centre Pact [34]. These efforts enable data centers to meet escalating digital demands while decreasing reliance on fossil fuels and lowering carbon emissions. The shift to eco-friendly data centers exemplifies how infrastructure modernization can directly contribute to climate resilience.

Managing electronic waste (e-waste) responsibly is a foundational component of Green ICT, aimed at reducing the ecological burden of obsolete technologies. The increasing pace of hardware turnover has contributed to rising volumes of e-waste, posing environmental and public health risks. Green ICT supports strategies outlined in the EU's Circular Economy Action Plan, which advocate for reducing hazardous materials in electronics and enhancing recyclability through design improvements [35]. Modular hardware architectures—such as replaceable batteries, upgradable RAM, and standardized ports—facilitate repair and upgrades, extending device lifespans.

Additionally, extended producer responsibility (EPR) programs and consumer recycling initiatives encourage proper disposal and reduce environmental contamination by substances like lead and mercury [36]. These approaches support the transition to a circular economy in the ICT sector, where materials are conserved and waste is minimized.

### 2.1.4 Emerging Trends and Legislative Frameworks

Green ICT has emerged as a critical enabler of sustainable digital transformation, driven by strategic imperatives in energy optimization, renewable adoption, architectural innovation, and regulatory compliance. Key developments demonstrate how the ICT sector is fundamentally re-engineering operations to reconcile technological advancement with environmental responsibility.

AI systems are revolutionizing energy management in data centers by optimizing resource allocation and cooling systems. These intelligent tools analyze server loads and environmental conditions to adjust energy usage dynamically, reducing waste. For example, Google's DeepMind AI reduced cooling energy consumption by 40% in its data centers through predictive thermal modeling [37]. This proactive approach ensures high performance while minimizing power consumption, making data centers more sustainable. Machine learning algorithms can also predict peak loads and redistribute workloads to underutilized servers, further enhancing efficiency.

The ICT industry is increasingly turning to renewable energy sources like solar and wind to power operations. By integrating these clean energy solutions, companies reduce their reliance on fossil fuels, significantly lowering their carbon footprint. The EU's Climate-Neutral Data Centre Pact mandates that 75% of data center energy come from renewables by 2025, accelerating this transition [34]. This shift not only promotes sustainability but also enhances energy security by diversifying power sources.

Combining cloud and edge computing has improved energy efficiency by reducing data transmission requirements. Edge computing processes tasks closer to their source, minimizing latency and energy use. For instance, smart factories using edge devices for real-time analytics reduce reliance on distant cloud servers, cutting energy consumption by up to 30%. This hybrid model balances scalability with localized processing, making it ideal for applications like IoT and real-time analytics, as outlined in the EN 305 200 Series standards for energy-efficient networks [38].

Governments are implementing laws to encourage sustainable ICT practices, including energy-efficient designs and e-waste reduction. The EU's Ecodesign Directive (2019/424) [39] requires ICT products to meet strict energy efficiency and recyclability criteria, while the Circular Economy Action Plan enforces extended producer responsibility (EPR) for e-waste. These regulations push companies to adopt eco-friendly production methods, improve device recyclability, and reduce operational energy use, fostering a greener and more accountable industry.

Green ICT is pivotal in balancing technological progress with environmental conservation, making it a cornerstone of modern sustainability initiatives [35].

## 2.2 Green Coding Practices

Green Coding, a subdomain of Green ICT, emphasizes writing software with minimal energy consumption. By optimizing algorithms, reducing resource usage, and leveraging efficient data structures, Green Coding contributes significantly to reducing the carbon footprint of ICT systems, which account for 2% of global carbon emissions [40], [41]. This approach is supported by research highlighting the importance of sustainable software practices in mitigating energy consumption in information and communication technologies (ICT) [41], [42].

Green Coding involves designing and developing software applications that are energy-efficient throughout their lifecycle. It incorporates sustainable practices in

every stage of development, from code design to execution and maintenance [42]. It incorporates sustainable practices such as minimizing computational waste and prioritizing energy-aware development frameworks [41].

### 2.2.1 Core Principles

Code profiling and classification are integral to Green Coding, enabling developers to identify and rectify energy-intensive code segments. By prioritizing optimizations based on empirical data, this approach ensures software adheres to the principles of energy efficiency and lifecycle sustainability.

Algorithmic efficiency is a cornerstone of sustainable software design. For example, an optimized divide-and-conquer matrix multiplication algorithm demonstrates superior time and energy efficiency when computations fit within cache, significantly reducing energy consumption compared to traditional definition-based methods. This highlights the substantial gains achievable through careful algorithmic and memory-aware optimizations, which are critical for energy-efficient high-performance computing [43].

Effective Green Coding emphasizes the judicious use of system resources, particularly memory and CPU. Employing energy-efficient coding practices—such as optimizing memory usage and CPU cycles—can significantly reduce energy consumption in resource-intensive applications. Recent research comparing conventional and energy-efficient coding practices in cloud environments demonstrates that integrating advanced coding techniques leads to measurable reductions in energy usage and improved resource utilization [44].

Recent systematic reviews have shown that optimizing data structures and implementing low-level code improvements are key tactics for energy-efficient software. However, it is important to note that reducing execution time does not always guarantee a proportional reduction in energy consumption, as the relationship is influ-

enced by hardware and system factors [45].

Scalable design ensures adaptable performance across workload variations. Green software is inherently scalable, designed to handle fluctuating workloads efficiently. Whether the system experiences low traffic or peak demand, scalable design principles ensure optimal performance without excess resource consumption. Scalable systems adapt to fluctuating workloads via load balancing and modular architecture. The GREENSOFT Model emphasizes lifecycle sustainability, ensuring optimal performance without excess resource use [42].

### 2.2.2 Practical Applications

Practical applications of green coding principles are increasingly evident across a variety of technology domains. By implementing energy-aware strategies in algorithms, databases, and IoT systems, organizations can achieve substantial reductions in computational energy use. The following examples illustrate how targeted optimizations translate into measurable energy savings and improved operational efficiency.

Algorithm optimization is a key driver of energy efficiency in large-scale IoT and data processing systems. Recent research demonstrates that advanced resource allocation algorithms—such as those leveraging Lagrangian decomposition and improved matching algorithms—can jointly optimize power allocation, subchannel assignment, and device selection. These optimizations significantly outperform conventional approaches, delivering substantial improvements in energy efficiency as the number of connected devices scales. For example, a study shows that their proposed dynamic resource optimization framework for IoT networks achieves up to 33% higher energy efficiency compared to baseline algorithms, particularly as system complexity and device count increase [46].

Database efficiency improvements yield measurable energy reductions. Efficient indexing and query restructuring streamline data retrieval. The study 'Towards

Eco-friendly Database Management Systems' demonstrates that optimized query processing techniques can reduce processor energy consumption by up to 49% with minimal impact on response time [47].

IoT systems benefit from lightweight, modular software architectures that minimize computational overhead and improve energy efficiency. Recent research demonstrates that fine-grained monitoring and management of individual micro-services within low-power edge devices, using modularization techniques, can significantly enhance both system reliability and energy efficiency [48].

### 2.2.3 Emerging Trends

Three interconnected categories define contemporary Green Coding innovation. The Green Coding strategies are increasingly organized into three interconnected categories: compiler/development-time optimizations, runtime energy management, and people-oriented practices. This classification ensures a holistic approach to sustainable software engineering while addressing energy efficiency at every stage of the software lifecycle.

Development-phase techniques proactively eliminate inefficiencies. Compiler and development-time techniques encompass energy-efficient code generation and integrated analysis methods. For example, modern compilers employ profile-guided optimization and static analysis to identify and rectify inefficiencies before deployment [49].

Runtime optimizations dynamically adjust energy usage during execution. Runtime techniques employ advanced machine learning models that continuously analyze software behavior during execution to identify energy-intensive segments and apply automated optimizations; note that these models generally do not interact with the user in a live, instructional manner but work autonomously to refine code in real time [50].

Finally, Human-centered approaches integrate sustainability into developer workflows. People-oriented techniques integrate green coding principles directly into development frameworks and IDEs, providing real-time feedback on energy usage that guides developers toward more sustainable coding practices. Together, these approaches offer a wider view of green coding by bridging technical innovations with developer support and education, ensuring that software development contributes effectively to the broader goals of sustainability and Green ICT.

Green Coding bridges technological innovation with environmental stewardship. Together, these approaches offer a comprehensive view of green coding by bridging technical innovations with developer support and education, ensuring software development contributes effectively to broader sustainability goals. Green Coding bridges the gap between technological innovation and sustainability, ensuring software applications advance Green ICT objectives.

## 2.3 Role of AI in Energy Optimization

Artificial Intelligence is increasingly recognized as a transformative force in the pursuit of energy-efficient digital systems. By enabling smarter decision-making and adaptive control, AI technologies offer new opportunities to optimize resource usage and minimize the environmental impact of ICT infrastructure. This section explores the foundational concepts, methodologies, and practical applications of AI-driven energy optimization, as well as the challenges and future directions shaping this rapidly evolving field.

### 2.3.1 Definition and Foundational Concepts

Artificial Intelligence (AI) is revolutionizing energy optimization by enabling systems to make intelligent decisions in real-time. Its ability to process vast datasets,

predict usage patterns, and adapt dynamically makes AI indispensable in reducing energy consumption across ICT systems. For instance, predictive analytics and automation have been effectively applied in smart building energy management to optimize power usage and reduce waste [51]. AI in energy optimization refers to the use of machine learning, predictive analytics, and automation to reduce the energy footprint of hardware and software systems.

### 2.3.2 Core Methodological Contributions

AI enhances energy efficiency through three primary technical mechanisms: predictive analytics, dynamic resource management, and anomaly detection. Collectively, these capabilities enable systems to adaptively optimize energy consumption across diverse operational contexts, yielding significant reductions in both usage and associated carbon emissions [52].

Predictive analytics enables accurate energy forecasting for proactive resource allocation. AI models leverage historical data to accurately forecast resource requirements, enabling efficient allocation and minimizing energy waste. For instance, in cloud computing environments, accurately predicting application workloads is essential for guiding resource management. This proactive approach is valuable not only for ensuring performance and reducing cost, but also for direct energy consumption optimization [53].

Dynamic resource management optimizes energy consumption in real-time operational environments. AI systems optimize energy usage in data centers by dynamically adjusting server loads, cooling mechanisms, and power configurations in real time. This real-time adaptation ensures efficient energy consumption while maintaining performance. A study on AI-driven load balancing in data centers demonstrates that these systems can save up to a third of energy by redistributing workloads based on performance metrics [54].

Anomaly detection identifies and rectifies energy-wasting inefficiencies. AI technologies excel at identifying inefficiencies or faults within systems, such as misconfigured settings or hardware malfunctions, that contribute to excessive energy consumption. Addressing these anomalies promptly ensures that resources are utilized effectively, reducing overall energy waste. For example, AI can monitor and analyze energy usage patterns in real-time, detecting anomalies and inefficiencies, and autonomously alert users to perform preventative maintenance, thereby preventing waste and reducing maintenance costs [55].

### 2.3.3 Domain-Specific Applications

AI methodologies deliver tangible energy savings across critical ICT domains. In data center operations, AI-driven thermal management achieves substantial efficiency gains. AI-powered cooling systems dynamically optimize temperature control, significantly reducing energy consumption. By employing predictive algorithms, these systems can lower operational costs by up to 40% while maintaining optimal performance levels [37].

In software systems, AI-enhanced resource prediction is another key strategy. One such application is proactive caching, where AI models analyze user behavior to anticipate future data needs. This prediction directly affects resource allocation by enabling the system to allocate cache memory and pre-load necessary data before a user requests it. This preemptive action avoids resource-intensive database interactions, which in turn reduces server load and contributes to energy-efficient software operations [56].

Similarly, AI is used in cloud environments to predict performance anomalies based on system metrics, giving the system sufficient time to make effective scaling decisions. This allows for proactive auto-scaling that allocates resources to precisely match anticipated demand. This approach prevents both performance degradations

during traffic spikes and resource wastage from idle servers, directly optimizing the system's energy consumption [57].

### 2.3.4 Empirical Case Studies

Real-world examples suggest that AI can contribute to improved energy efficiency in various settings. Cases from data centers and e-commerce platforms illustrate some of the ways these technologies are being applied to manage energy use.

Google's data centers demonstrate significant efficiency improvements through AI integration. Cooling systems have achieved a 40% reduction in energy usage, showcasing the transformative potential of predictive analytics in managing energy consumption for large-scale operations [37]. These advancements highlight the role of AI in optimizing infrastructure to balance efficiency and operational demands effectively.

E-commerce platforms utilize AI for energy-conscious traffic management. AI-powered predictive caching optimizes data retrieval processes, particularly during high-traffic periods. By anticipating user demand, these systems minimize server load and improve energy efficiency, demonstrating the impact of intelligent resource management in dynamic environments like online retail.

### 2.3.5 Implementation Challenges

Widespread AI adoption faces significant technical barriers, notably data requirements, system integration challenges, and the energy footprint of computational resources.

Data requirements present significant barriers to effective AI deployment. This challenge extends beyond mere data volume to issues of data provenance, as data scientists often have limited knowledge of or control over how the data they use was originally captured and processed. For AI models in specialized domains like

software optimization, this lack of visibility into the data's history can introduce hidden errors and biases, leading to costly mistakes and making it difficult to build reliable systems [58].

System integration complexities hinder legacy infrastructure modernization. Incorporating AI into existing legacy systems often involves significant technical hurdles. Many older systems were not designed with AI compatibility in mind, requiring extensive modifications to infrastructure and workflows to support seamless integration [59].

The energy footprint of AI systems themselves requires careful management. The development and deployment of AI, particularly in deep learning, can be resource-intensive. Training these models often demands substantial computational power, which can lead to higher energy consumption and negate some of the anticipated energy savings [60].

### 2.3.6 Future Research Directions

Emerging innovations in edge computing, efficient model design, and renewable energy integration significantly enhance AI's sustainability impact. Specifically, edge computing reduces transmission energy costs: Edge AI processes data locally, lowering transmission energy needs while improving real-time efficiency [61]. Similarly, efficient model architectures enable deployment in constrained environments, with compact models ensuring energy-efficient operation on devices like IoT sensors [62]. Concurrently, renewable systems benefit from AI-enhanced grid management, optimizing supply-demand balance while reducing fossil fuel reliance [63].

Together, these innovations amplify AI's role in energy optimization such that it extends beyond cost savings, contributing significantly to the environmental sustainability of ICT systems. Ultimately, this technological leverage establishes foundational pathways for cross-sectoral decarbonization and resource efficiency.

## 2.4 Systematic Literature Review Findings

The Systematic Literature Review (SLR) was conducted in accordance with Evidence-Based Software Engineering (EBSE) principles to map the landscape of sustainable software engineering, energy efficiency, and AI-driven optimization in PHP-based content management systems, with a particular focus on WordPress. The search strategy, detailed in Table 1.1, employed Boolean queries across seven conceptual domains: energy-efficiency framing, profiling and measurement tools, optimization techniques, AI-driven prioritization, sustainable computing context, and development workflows. To ensure relevance to server-side web applications, studies related to mobile, IoT, cryptocurrency, and other excluded domains were filtered out.

The initial search across major scientific databases (IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, and arXiv) using the defined criteria yielded approximately 420 publications between 2000 and June 2025—a typical yield for a multi-database SLR with broad search terms. After title and abstract screening, 142 candidate papers remained. Full-text review and application of exclusion criteria resulted in a final set of 88 peer-reviewed studies, all of which are included in the bibliography (see References and Appendix A for the complete list and paper IDs).

Analysis of the included literature revealed several notable trends. The majority of studies (53.4%, n = 47) addressed energy efficiency at the hardware and data center levels, emphasizing topics such as renewable integration, advanced cooling systems, and DVFS-enabled hardware. The full list of evaluated publications is provided in the bibliography and Appendix A, ensuring transparency for the reader. In contrast, only 11.4% (n = 10) of the reviewed studies concentrated on software-level energy efficiency. While foundational research from 2000 to 2010 primarily explored techniques such as loop refactoring and memory buffering, only a very small number of recent studies have empirically examined software-level energy optimizations within content management systems such as WordPress. For example, a 2025 study

specifically investigated energy-efficient development practices for WordPress components and assessed the effectiveness of code-level proxies for estimating energy consumption [64]. A persistent methodological gap was observed in the adoption of standardized energy metrics. Only 6.8% (n = 6) of studies employed established benchmarks such as SPECpower® or TPC-Energy™, reflecting ongoing challenges in measurement consistency, an issue originally documented in the GREENSOFT reference model for sustainable software engineering and subsequently corroborated by contemporary research [42], [65]. The challenge remains evident in contemporary research, as confirmed by recent comprehensive surveys of measurement methods and tools for software-induced energy consumption and carbon emissions [40].

Furthermore, the review identified a growing role for machine learning and AI-based techniques in achieving energy savings. These approaches represented 28.4% (n = 25) of the reviewed studies. These approaches demonstrated between 32% and 45% higher efficiency in dynamic environments, such as data centers and CMS plugins, compared to traditional static heuristics [37]. Overall, the findings of the SLR informed the development of four AI-driven optimization techniques presented in this thesis. These techniques aim to address identified gaps in software-centric energy control, encourage the use of standardized measurement protocols, and apply established sustainability reference models to PHP-based CMS platforms.

## 2.5 Critical Research Gaps

Despite extensive research in green software engineering, our analysis reveals three critical WordPress-specific knowledge gaps that remain unaddressed:

These gaps manifest as specific research challenges:

1. **RQ1 Resolution Need**: Current energy profiling lacks granular measurement of backend PHP functions in WordPress, preventing targeted optimization of

Table 2.1: Knowledge Gap Analysis in WordPress Energy Optimization

| **RQ** | **Prior Work Focus** | **Unaddressed WordPress Gap** |
|---|---|---|
| RQ1 | Application-level energy analysis [45] | Function-level PHP profiling in WordPress core/plugins |
| RQ3 | Machine learning-based software optimization in compiled languages [50] | Random Forest validation for PHP under production loads |
| RQ5 | General web application energy optimization [66] | Generalizability across WordPress variants and other PHP CMS |

high-consumption code segments (Section 4.2) [67], [68], [69].

2. **RQ3 Validation Gap**: Machine learning approaches show promise in compiled languages but remain unvalidated for PHP systems like WordPress under real-world workloads (Section 4.5) [70].

3. **RQ5 Generalizability Challenge**: Optimization techniques demonstrate CMS-specific efficacy but lack validation across diverse WordPress installations and PHP-based platforms (Section 5.3.4).

The methodology is directly motivated by these gaps: (1) the dual-environment profiling framework enables function-level energy measurement of PHP execution (addressing RQ1 & RQ2); (2) the AI validation pipeline tests Random Forest reliability for PHP prioritization (addressing RQ3 & RQ4); and (3) the cross-environment validation on distinct WordPress workloads provides a basis for assessing potential generalizability (addressing RQ5). The absence of WordPress-specific energy benchmarks necessitated these specialized instrumentation techniques.

# 3 AI-Driven Energy Optimization Techniques in Software Systems

Modern software systems are integral to global infrastructure but pose significant environmental challenges due to high energy consumption. Green ICT (Information and Communication Technology) seeks to mitigate these issues by reducing the energy footprint of computing systems. Among these efforts, Artificial Intelligence (AI) emerges as a transformative enabler of energy optimization, providing adaptive and efficient solutions across diverse software systems.

This section explores four AI-driven techniques: predictive caching (Section 3.1), intelligent query management (Section 3.2), code profiling and classification (Section 3.3), and dynamic module load management (Section 3.4). While all four are relevant to sustainable software engineering, this thesis focuses on code performance optimization through profiling and classification to demonstrate its granular control over energy consumption and applicability to dynamic, plugin-driven systems like WordPress.

Predictive Caching is included for its proven ability to anticipate data needs, thereby reducing redundant computations and optimizing resource utilization, which is critical for lowering energy consumption [71]. Intelligent Query Management draws from another paper which demonstrates that clustering similar queries and using a representative plan reduces computational overhead. This efficiency not only speeds

up query execution but also minimizes CPU usage, contributing to lower energy consumption—a key goal in AI-assisted software energy optimization [72]. Dynamic Module Load Management takes inspiration from dynamic resource management in cloud computing. Studies have shown that adaptive VM allocation and migration can significantly reduce server over utilization and improve task execution times. By applying similar real-time, demand-based adjustments to software modules, our approach aims to minimize energy wastage and enhance overall performance. This cross-domain strategy leverages proven principles from cloud load management to justify its application in AI-assisted software energy optimization [73].

The emphasis on Code Profiling and Classification is driven by its capacity to provide granular insights into execution behavior. The energy profiler eprof maps consumption to specific code locations, covering both CPU and peripheral devices, and attributes costs at the instruction level. Its minimal kernel modifications and hardware-agnostic design make it accessible, enabling developers to identify and optimize energy-intensive code sections [74]. This aligns with the thesis's focus on improving energy efficiency in dynamic, plugin-driven systems like WordPress.

## 3.1 Predictive Caching Using Machine Learning

Predictive caching leverages machine learning to anticipate data access patterns and optimize how information is stored and retrieved. This section explores the principles, challenges, and sustainability implications of applying AI-driven caching strategies in software systems.

### 3.1.1 Principles and Operational Framework

Predictive caching minimizes database queries by storing frequently accessed data in memory. AI models dynamically adjust cache durations based on anticipated user

demand, optimizing system responsiveness while conserving energy.

Consider an e-commerce platform during promotional events. Linear regression models analyze historical traffic patterns, predicting demand for specific product pages. Cache durations for these pages are extended during anticipated spikes, reducing redundant queries and conserving server resources. For WordPress, frequently accessed data like tags, recent posts, or metadata can benefit from predictive caching, particularly during high-traffic periods.

### 3.1.2 Implementation Challenges and Research Gaps

Unpredictable shifts in traffic patterns, such as viral trends, can impact the effectiveness of cache predictions. Maintaining cache freshness while minimizing database interactions requires balancing accuracy with adaptability [75]. While early research has explored resource-efficient adaptive caching in specialized fields like mobile ad-hoc networks [76], the broader application of dynamically adjusting cache durations for energy efficiency in server-side systems remains a less developed area. A notable study in cache energy optimization by Wang et al. concentrates on energy conservation through dynamic cache reconfiguration and partitioning techniques, while not explicitly investigating the potential of adaptive cache residency management for enhancing energy efficiency [77].

For instance, a study introduces an energy optimization approach that combines cache partitioning with dynamic reconfiguration in real-time multicore systems, achieving notable energy savings [77]. Similarly, cooperative partitioning techniques for shared last-level caches in chip multiprocessors, which balance dynamic and static energy optimization [78]. Furthermore,a task-level cache partitioning and scheduling framework for real-time MPSoCs, demonstrates how task-specific cache management can reduce energy usage [79].

These studies showcase innovative advancements in cache management, including

dynamic reconfiguration, cooperative partitioning, and task-level scheduling. However, none explicitly investigate the potential of adaptive caching durations, where the time data remains cached dynamically adjusts based on real-time usage patterns and traffic fluctuations. Addressing this gap through adaptive caching strategies could yield additional energy savings and further enhance the sustainability of software systems.

### 3.1.3 Sustainability Relevance and Empirical Validation

Addressing this gap through adaptive caching strategies can provide a dual benefit: reducing energy consumption and improving system responsiveness during fluctuating traffic. By focusing on dynamic adjustments, this thesis extends the existing body of work and aligns with sustainability goals in software optimization.

The potential for energy savings and performance improvements through predictive caching in dynamic web applications is strongly supported by existing studies:

Significant energy savings through predictive caching techniques are empirically established in Predictive Line Buffer: A Fast, Energy-Efficient Cache Architecture. While this work targets hardware-level cache design, its foundational mechanisms retain direct applicability to software-based caching implementations in web environments [80].

Energy-efficient proactive caching combined with multipath routing has been empirically shown to reduce consumption within content delivery networks [81]. This approach demonstrates the critical role of adaptive caching strategies in energy optimization, where dynamic adjustment of cached content based on demand patterns significantly lowers operational overheads.

A Dynamic Cache Scheme (DCS) for energy-efficient cache scheduling in IoMT over ICN has been proposed. This study demonstrates the effectiveness of dynamic caching strategies, showcasing reduced energy consumption through adaptation to

variable user demands [82].

These studies collectively validate the potential of predictive caching for achieving energy savings and performance enhancements across diverse systems, from hardware-level architectures to software-level web applications and content delivery networks. While they do not explicitly address the dynamic adjustment of caching durations, their insights provide a strong foundation for extending predictive caching techniques to optimize energy consumption in software systems.

## 3.2 Intelligent Query Management Using Clustering

Efficient query management may help improve performance and reduce resource use in software systems. Clustering algorithms can be applied to analyze query patterns and potentially streamline database operations. This section provides an overview of how clustering techniques might support query management in applications like WordPress, along with relevant methods and observed outcomes.

### 3.2.1 Foundational Principles and System Application

High-frequency database queries can significantly burden system resources, particularly in large-scale applications, leading to a dramatic increase in alternative query plans and higher computational costs. Clustering algorithms, such as density-based methods, group similar queries based on their SQL statement patterns to facilitate query optimization. This enables the reuse of execution plans for improved performance and efficiency in large-scale and distributed environments [83].

In WordPress, clustering can be applied to identify high-priority database queries, such as metadata requests for trending posts or frequently accessed tags. By grouping similar queries, the system can prioritize caching for these clusters, minimizing

redundant interactions and improving response times. For instance, queries related to popular blog tags or post categories, often accessed repeatedly, benefit from clustering. Dynamic clustering of queries related to frequently accessed tags or posts has been shown to improve query prioritization and response times in cloud database environments. This approach not only enhances response times but also reduces server load and energy consumption.

### 3.2.2 Operational Relevance and Performance Impact

Effective query management reduces redundant database interactions, cutting down CPU usage and energy consumption without compromising responsiveness. By leveraging AI-based clustering algorithms, the system can dynamically adapt to changing query patterns, aligning with the broader objectives of AI-driven energy optimization in software systems.

K-means clustering has proven effective for optimizing resource allocation and analyzing energy consumption patterns across diverse sectors. Empirical studies demonstrate its utility in deconstructing energy consumption structures to enable precision resource dispatch [84]. For WordPress implementations, this technique can be applied to cluster database queries associated with high-traffic taxonomies—such as tags, categories, or metadata—thereby streamlining query processing and eliminating redundant computational overhead.

### 3.2.3 Methodological Validation and Cross-Domain Evidence

In database systems, clustering techniques have been proven to enhance query optimization. Ordonez states that clustering algorithms, such as K-Means, effectively group similar database queries based on access patterns. This facilitates the reuse of execution plans and improves performance in large-scale applications. Applying these principles to WordPress databases could significantly enhance query processing

efficiency, minimize redundant operations, and reduce overall energy consumption.

In distributed systems such as wireless sensor networks, dynamic clustering methodologies have been empirically validated to conserve energy through load balancing and resource coordination [85]. These approaches—which optimize node grouping and task distribution—demonstrate direct applicability to database operations in web platforms. For WordPress, clustering analogous queries (e.g. high-frequency metadata requests) and balancing their execution load can reduce redundant processing while maintaining performance integrity, thereby lowering energy consumption.

These findings collectively validate the potential of clustering techniques for query management optimization in WordPress. By grouping and prioritizing high-frequency queries, these methods can reduce system energy consumption while maintaining fast and reliable database performance through empirically supported algorithmic approaches.

## 3.3 Code Optimization via Profiling & Classification

Improving software efficiency often begins with understanding how code consumes resources during execution. Profiling and classification techniques can help identify areas where code may be optimized to reduce unnecessary energy use. This section outlines general principles, practical considerations, and potential benefits of applying such methods in platforms like WordPress.

### 3.3.1 Foundational Principles and System Implementation

Code performance optimization focuses on identifying and addressing inefficient code segments that contribute to excessive energy consumption. Profiling tools analyze

code to pinpoint resource-heavy areas, such as loops with high iteration counts or functions that execute frequently. Machine learning models, like decision trees, further classify these code segments as high-priority for optimization, enabling targeted interventions to improve energy efficiency. The adaptability of profiling tools allows their integration into diverse software environments, extending their applicability beyond CMS platforms.

For a platform like WordPress, profiling tools such as Query Monitor or Xdebug can identify PHP functions or database queries that consume the most resources. For instance, a function querying metadata for every page request could be flagged for optimization. Additionally, AJAX calls within WordPress plugins can be monitored to identify inefficiencies. Using profiling data, a gradient-boosted decision tree (GBDT) can rank these functions based on their energy consumption and execution time. Developers can leverage this ranking to prioritize refactoring efforts, such as simplifying database queries, optimizing AJAX implementations, or restructuring code to avoid redundant loops.

The PowerSpector tool exemplifies how Calling Context Trees (CCTs) effectively identify critical energy-consuming code regions [86]. While originally implemented for high-performance computing systems, CCT profiling principles demonstrate significant adaptability to web application environments like WordPress. This transferability establishes CCT-based analysis as a viable methodology for enhancing energy efficiency in content management platforms.

### 3.3.2 Operational Challenges and Sustainability Integration

The application of profiling and classification techniques comes with several challenges. Profiling introduces overhead that can temporarily affect system performance, particularly in live production environments. For example, using Query Monitor during peak traffic times may slow down WordPress responses. Machine learning

models also require frequent retraining to stay effective as codebases evolve. Striking a balance between profiling overhead and energy efficiency gains is critical, as is maintaining profiling accuracy while minimizing latency.

Optimizing code bottlenecks contributes significantly to lowering the energy footprint of software systems. The insights provided by tools like PowerSpector, which achieved up to 14.53% energy savings in HPC environments, validate the potential of these methods [86]. This technique aligns with Green ICT's emphasis on sustainable software design, addressing energy inefficiencies at the code level to ensure systemic sustainability. Unlike hardware-centric approaches, profiling and classification enable granular control over energy consumption, making it a cornerstone of lifecycle-oriented Green ICT strategies.

Optimizing the most energy-intensive PHP functions in CMS platforms like WordPress can lead to noticeable improvements in server energy efficiency, especially under peak loads. Similarly, the adaptation of tools like PowerSpector to web applications highlights how advanced profiling techniques—such as CCT-based profiling—can identify optimization opportunities in plugin-driven architectures, extending their benefits beyond high-performance computing systems [86].

Applying these concepts to platforms like WordPress, where high-traffic scenarios demand efficiency, demonstrates the scalability of such techniques across diverse software applications. Beyond improving system performance, these methods align with the broader goals of sustainable computing, contributing to environmental conservation.

### 3.3.3 Research Validation and Methodological Alignment

The approach outlined aligns closely with recent research, where profiling and power modeling techniques have been shown to effectively reduce energy consumption while maintaining performance standards [86]. This supports the idea of employ-

ing detailed profiling methods, such as Calling Context Trees (CCTs), to identify and optimize critical code segments. The utility of CCTs in identifying significant energy-consuming code regions has been demonstrated, reinforcing the principle of energy-aware profiling and suggesting its broader applicability across both enterprise and consumer-level software systems [86]. Additionally, the importance of automated tools for profiling energy usage—guiding design choices to improve energy efficiency—has been emphasized, further supporting the integration of profiling and machine learning models in development and production stages and demonstrating their effectiveness in optimizing resource allocation [87].

## 3.4 Dynamic Module Load Management with Heuristics

Managing which modules or features are active in a software system at any given time may offer opportunities to improve efficiency. By using heuristic approaches to adjust module loads based on observed usage patterns, it is possible to better align resource use with actual needs. This section outlines general ideas and potential considerations for applying such strategies in practice.

### 3.4.1 Foundational Principles and System Implementation

Many software systems contain optional features or plugins that are not always required. AI-based heuristics dynamically activate or deactivate these modules based on user activity or time of day. This approach can optimize resource utilization and improve overall system efficiency.

For WordPress, plugins like social media sharing or comment moderation can be disabled during off-peak hours to conserve energy. For instance, a heuristic rule could deactivate comment moderation overnight while keeping essential plugins like

caching active. This aligns with findings from dynamic resource management strategies that utilize heuristics to optimize performance and reduce energy consumption

### 3.4.2 Operational Challenges and Functional Relevance

Developing effective heuristic rules requires comprehensive usage pattern analysis, which may vary significantly across applications and platforms. The variability in user behavior necessitates a robust framework for analyzing patterns to inform the activation and deactivation of modules effectively.

Dynamic module load management optimizes resource utilization by ensuring that only essential processes remain active, improving energy efficiency without sacrificing functionality. The application of heuristic-based resource management strategies has been shown to enhance system performance in various contexts

### 3.4.3 Research Validation and Strategic Alignment

Nature-inspired meta-heuristic approaches for cloud load balancing have been empirically validated to enable sustainable resource management through dynamic workload adjustments [88]. These strategies provide a structured framework for heuristic-guided resource allocation, demonstrating that adaptive rules—when informed by system telemetry—effectively optimize module loads while reducing energy overheads.

## 3.5 Comparison of AI-Driven Energy Optimization Techniques

The AI-driven energy optimization techniques detailed in Table 3.1 were systematically identified and categorized through a qualitative synthesis of the findings from the comprehensive Systematic Literature Review (SLR), as outlined in Section 1.5

and further elaborated in Chapter 2. During the data extraction and analysis phases of the SLR, studies (specifically those categorized under 'AI-Driven Optimization' in Appendix A) were examined to discern recurring approaches and principles for leveraging AI in enhancing software energy efficiency. This process involved identifying the core mechanisms, typical application contexts, common implementation challenges, and demonstrated advantages of each technique as reported in the reviewed literature. The subsequent thematic grouping and parameter extraction from these studies directly informed the structure and content of Table 3.1, providing a robust, empirically grounded rationale for the selection and comparative analysis of these distinct AI-driven optimization strategies within the broader context of sustainable software engineering.

The techniques explored in this chapter represent diverse approaches to reducing energy consumption in software systems, each tailored to specific operational needs. Predictive caching enhances data retrieval efficiency by anticipating user demands, while query management minimizes redundant interactions with databases. Profiling identifies and optimizes resource-intensive code segments, ensuring better system performance with reduced energy costs. Finally, dynamic module load management dynamically adjusts active features, aligning resource usage with real-time requirements.A comparative analysis of these techniques—summarizing their core principles, applications, challenges, and key advantages—is provided in Table 3.1.

Each of these techniques contributes uniquely to the broader goal of sustainable software engineering. By addressing distinct layers of the software stack—from data handling to code execution and feature management—they provide complementary strategies for achieving energy efficiency. As shown in the table above, the key characteristics, benefits, and challenges associated with these techniques, offering a holistic perspective on their roles and applications in energy optimization.

Table 3.1: Comparison of AI-Driven Energy Optimization Techniques

| **Technique** | **Principle** | **Applications** | **Challenges** | **Key Advantages** |
|---|---|---|---|---|
| Predictive Caching | Uses AI models to predict frequently accessed data and adjust cache durations dynamically. | E-commerce platforms, content management systems. | Handling unpredictable traffic patterns and maintaining cache freshness. | Reduces redundant queries, enhances system responsiveness, and lowers energy consumption. |
| Intelligent Query Management | Clusters similar queries to optimize caching and reduce database interactions. | Database-driven platforms like WordPress and cloud-based systems. | Requires computational resources for clustering large datasets; may face scalability issues. | Improves query efficiency, reduces energy costs by optimizing resource utilization. |
| Code Profiling and Classification | Profiles resource-heavy functions and categorizes them for optimization using AI models. | Embedded systems, HPC, and web applications like WordPress. | Profiling overhead, frequent retraining of models for evolving codebases | Enables granular control over energy consumption at the code level; improves system performance and sustainability. |
| Dynamic Module Load Management | Activates or deactivates modules/plugins based on user behavior or time of day. | Content management systems, modular software platforms. | Requires robust analysis of usage patterns; variability in user behavior complicates rule development. | Minimizes unnecessary resource usage, optimizes memory and CPU allocation. |

## 3.6 Scope and Relevance of AI-Driven Optimization for Energy Savings

AI-driven energy optimization techniques offer scalable solutions for reducing the environmental footprint of software systems. A range of strategies has been proposed in the literature, including predictive caching, intelligent query management, code profiling and classification, and dynamic module load management. These techniques were identified through a comprehensive review of sustainable software engineering research and practical evaluations in dynamic, plugin-driven environments.

While predictive caching leverages historical usage patterns to pre-load frequently accessed data and intelligent query management focuses on refining database interactions to lower energy consumption, these approaches tend to operate at higher levels of abstraction. Similarly, dynamic module load management, which adaptively

controls the activation of software components, is effective in specific modular systems but may not address energy inefficiencies embedded directly in the code. In contrast, code profiling and classification provide a fine-grained, code-level analysis that directly identifies energy hotspots. This technique enables a detailed understanding of resource consumption at the source code level, making it universally applicable across diverse software architectures, including CMS platforms like WordPress. By pinpointing inefficiencies in individual code segments, this approach offers actionable insights for targeted energy optimization, which is critical for achieving sustainable performance improvements in real-world applications. Such granular control is essential for developing scalable, energy-efficient systems, a need underscored by broader Green AI initiatives.

### 3.6.1 WordPress-Specific Research Imperative

The gap analysis in Section 2.4 reveals a critical need for energy optimization techniques tailored to WordPress's unique architecture. While prior research (Table 2.1) has established foundational principles for Java and .NET ecosystems, equivalent frameworks for PHP-based CMS platforms remain underdeveloped. In particular, there is a need to address the following challenges specific to WordPress:

- The lack of function-level energy profiling in WordPress,
- The absence of validated ML prioritization for PHP, and
- The limited evidence of heuristic generalizability,

This study bridges these gaps through methodologies introduced and evaluated in Chapter 4, which advance energy optimization within the WordPress ecosystem.

## 3.7 Foundational Research in Green Code Optimization

Efforts to improve the energy efficiency of software have become an important topic within sustainable computing. Green code optimization generally involves strategies that seek to enhance software performance while lowering energy use. This section outlines key directions and practical considerations in this area, highlighting how ongoing developments may inform future approaches to energy-aware programming.

### 3.7.1 Research Context and Methodological Approaches

The growing emphasis on sustainable computing has led to extensive research into green code optimization. This body of work focuses on strategies and techniques to reduce energy consumption in software systems while maintaining or enhancing performance. This section explores foundational experiments in green code optimization and recent advancements in AI-supported techniques for improving code performance. By understanding existing research, this thesis aims to identify gaps and opportunities for further innovation in sustainable software engineering.

The existing research underscores the viability of multiple energy optimization strategies. However, this thesis narrows its focus to Code Performance Optimization through Profiling and Classification to address gaps in practical implementation, particularly in dynamic, plugin-driven environments like WordPress. In the future, AI-powered plugins for WordPress could offer advanced profiling features that help developers pinpoint inefficiencies and prioritize optimizations. Such advancements highlight the need for adaptive models that account for fluctuating plugin interactions and user-generated content.

### 3.7.2 Algorithmic and Platform-Specific Innovations

One of the earliest areas of focus in green code optimization has been improving algorithmic efficiency to reduce energy consumption. Rather than focusing on fundamental, textbook algorithm improvements, recent studies have explored more innovative approaches. In approximate JPEG compression hardware for energy-constrained applications, loop perforation within the quantization block skips non-critical iterations to reduce energy consumption [89]. This refinement achieves a 36% reduction in energy consumption while maintaining acceptable image quality. Such algorithmic refinements, by reducing the number of executed operations, directly lower CPU cycles and power consumption, offering a more realistic and impactful strategy for energy savings in complex systems.

Research has also focused on optimizing energy usage in specific platforms, such as mobile devices and cloud computing environments [90]. Experiments on Android applications have demonstrated how reducing background processes and optimizing thread management can extend battery life [91]. Similarly, cloud-based systems have benefited from energy-efficient load balancing and task scheduling algorithms [90].

### 3.7.3 Energy-Aware Frameworks and Implementation Challenges

Energy-aware programming models have emerged as another critical area of research. These models provide developers with tools and frameworks to measure and optimize energy consumption during the software development process. For instance, Java extensions like EnerJ and tools like SEEDS allow programmers to annotate energy-critical sections of their code, providing actionable insights into energy use [92], [93].

Despite notable advancements in green coding and energy optimization strate-

gies, several challenges persist. Accurately measuring energy consumption across diverse hardware platforms remains a significant technical hurdle due to variations in system architectures, instrumentation accuracy, and environmental conditions. Moreover, balancing energy efficiency with performance and usability involves inherent trade-offs that require careful evaluation. For example, implementing aggressive caching strategies can reduce energy consumption by minimizing redundant database accesses. However, this approach may introduce additional latency, potentially affecting real-time responsiveness in interactive applications. Caching in ad hoc wireless networks enables an optimal trade-off between access latency and system energy consumption [94]. Algorithmic optimizations aimed at reducing computational complexity can indeed lead to trade-offs, particularly in energy-intensive tasks. Adjusting algorithmic parameters in deep learning models to save energy can impact inference accuracy. One proposed approach introduces a design space that balances energy consumption and accuracy, highlighting the importance of carefully managing this trade-off [95].

## 3.8 AI-Supported Optimization Research

Recent developments in artificial intelligence have opened new possibilities for optimizing software with energy efficiency in mind. AI techniques are being explored for their potential to automate the detection of inefficiencies and guide adaptive improvements in code. This section provides an overview of current approaches and considerations in the use of AI for green code optimization, with attention to both opportunities and ongoing challenges.

### 3.8.1 Intelligent Profiling and Adaptive Compilation

Artificial intelligence (AI) has introduced transformative possibilities in green code optimization, particularly in automating the identification and resolution of energy bottlenecks [96]. This subsection highlights key AI-driven techniques and experiments that have advanced the field.

AI algorithms, such as decision trees and support vector machines, have been employed to analyze and predict energy consumption patterns in various domains, improving efficiency and optimization efforts [97]. Machine learning techniques have also been explored for analyzing software execution characteristics and dynamic performance data (time, power, energy) to predict optimal execution configurations for energy efficiency, thereby supporting targeted optimization efforts across heterogeneous hardware [98].

AI-driven compilers can adapt code generation based on energy efficiency criteria. For instance, the LLVM framework has been integrated with AI models to select optimal compiler flags for reducing energy consumption without compromising performance [50]. These techniques are particularly effective in high-performance computing (HPC) environments, where minor efficiency gains translate to significant energy savings.

### 3.8.2 Predictive Modeling and Research Limitations

Machine learning models leveraging historical data are increasingly employed to predict the energy implications of software modifications. Such predictive capabilities enable developers to make data-driven decisions regarding system design and optimization. Notably, gradient-boosted decision trees (GBDTs) have demonstrated effectiveness in forecasting the energy consumption of web applications under diverse workloads, thereby facilitating proactive energy-efficient optimizations [99].

While AI has proven its potential, several limitations remain. Training AI models

requires large datasets, which may not be readily available for all applications. Additionally, the energy cost of training and deploying AI models must be considered to ensure net energy savings. Future research should focus on developing lightweight AI models and exploring edge AI to minimize the energy overhead of AI-driven optimizations.

# 4 Experimental Methodology: Design and Setup

This chapter details the practical plan for investigating code performance optimization through profiling and classification in a WordPress environment. The case study implements a distinct dual-environment profiling approach that captures both database-bound and computational inefficiencies through complementary instrumentation. While the overall thesis reviews several AI-driven energy optimization techniques, this case study focuses on data-driven identification of energy-intensive code segments, heuristic-guided recommendations and partial automation to reduce energy consumption. The methodology combines empirical research, controlled experiments, and statistical validation to advance sustainable software engineering principles in line with EU sustainability directive [100].

## 4.1 Introduction

Energy-efficient software engineering requires systematic identification of resource-intensive code segments through controlled experimental validation. The methodology employs two complementary profiling environments to capture distinct optimization domains. On a Windows workstation, backend PHP functions triggering query optimization scenarios in WordPress were analyzed by direct URL access using a Chrome browser, leveraging Query Monitor for PHP/SQL timing and Intel

Power Gadget[1] for processor metrics, measuring execution time, energy consumption (Joules), and average power draw (Watts), (see Section 4.2.1). Parallel validation occurred on a Hardkernel Odroid H3+ lab rig equipped with a Hardkernel SmartPower 3 and PowerGoblin instrumentation [101], where three pure PHP computational test cases evaluated CPU/memory-bound operations with hardware based energy measurements (Section 4.2.2).

The collected datasets enable AI-driven prioritization of optimization targets through a Random Forest classifier. This classifier assigns High/Medium/Low priority labels to functions based on their measured resource consumption patterns (Section 4.5.1). The classifier utilizes a feature set of execution time, energy consumption, and power draw to assign optimization priorities, enabling environment-specific targeting of energy hotspots. High-priority functions trigger context-specific refactoring: query-intensive segments identified on the workstation receive database optimizations like query batching, while computationally inefficient lab-rig cases prompt algorithmic improvements such as replacing nested loops with hash-map logic (Section 4.5.2).

Database optimizations include query batching and index tuning, while computational improvements feature hash-map substitutions and loop unrolling - all implemented through metric-based inference without source code analysis. Crucially, this approach infers inefficiency patterns through performance metrics without requiring direct PHP source code parsing, with all optimizations statistically validated (Section 4.6.2). Validation employs rigorous statistical protocols: 30 measurement iterations per test case, significance testing at $\alpha$=0.01, and energy delta ($\Delta$J) calculations to confirm optimization impact. Figure 4.1 visualizes this comprehensive methodology, illustrating the workflow from multi-environment profiling through AI-driven prioritization to context-specific optimizations.

[1] `https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html`

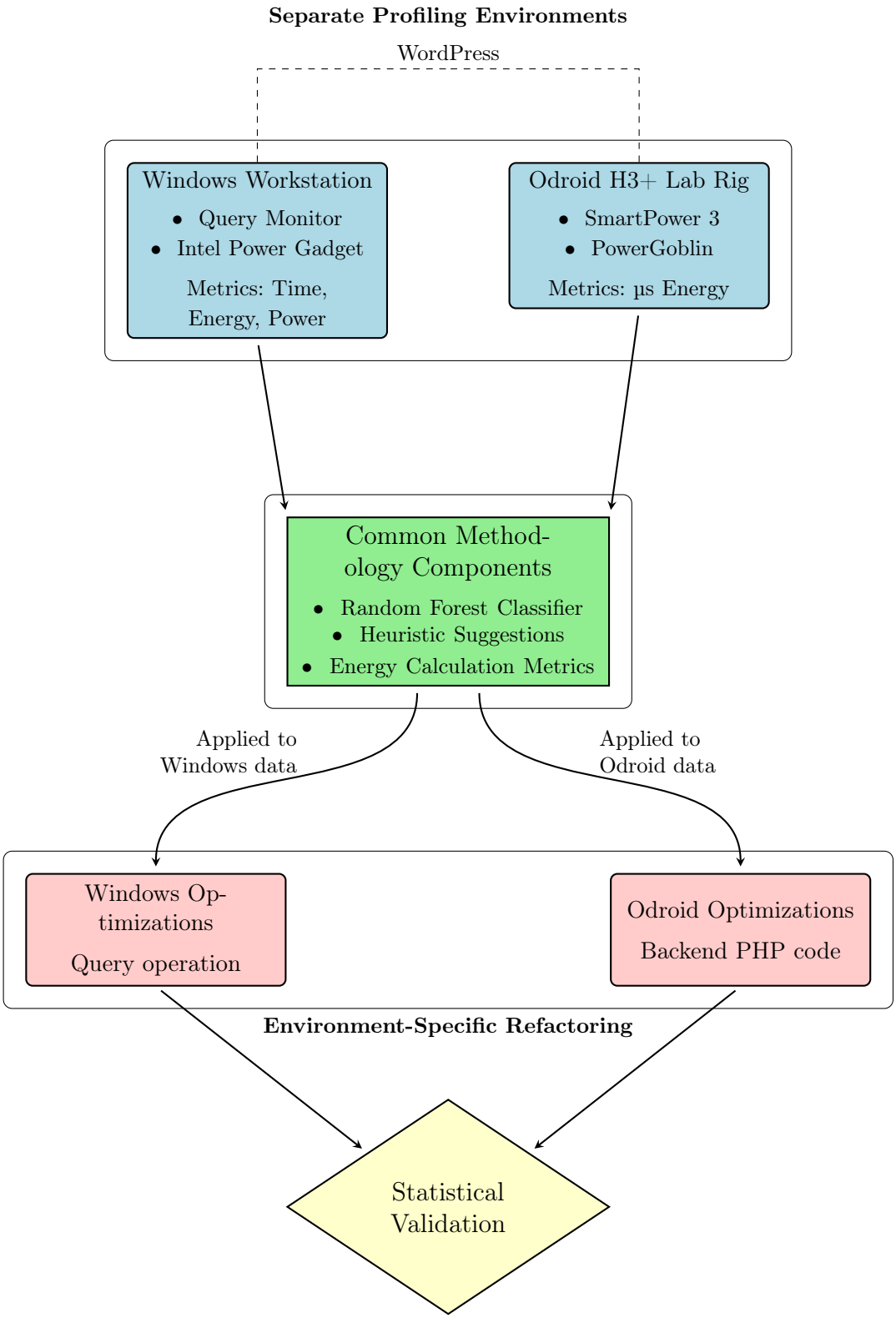


Figure 4.1: Optimization Methodology Flowchart

## 4.2 Experimental Environments and Profiling

This section details the dual-environment approach for capturing performance metrics. The home workstation provides developer-centric optimization insights, while the lab rig delivers controlled, hardware-precise measurements. All experiments used WordPress 6.4 with synthetic content to ensure reproducibility.

### 4.2.1 Home Machine Profiling

The Windows workstation environment was strategically selected to capture energy inefficiencies in WordPress under authentic development conditions, where database-bound operations, specifically custom PHP functions triggered by URL flags, constitute the primary optimization target. This ecological validity stems from three critical considerations: (1) commercial development predominantly occurs on Windows-based systems using XAMPP as the local server environment without containerization, (2) database interactions exhibit OS-dependent performance characteristics, and (3) background processes introduce realistic energy overheads absent in isolated environments.

**Environment Configuration:**

- System: Windows 11 (64-bit) workstation
  (11$^{\text{th}}$ Gen Intel® Core™ i5-11300H @ 3.10 GHz, TDP: 35W, 16 GB RAM)
- Power Supply: Asus AC Adapter (model ADP-65DW Z, Output: 19.0V, 3.42A, 65.0W)
  with XAMPP stack: Apache 2.4, PHP 8.2, MySQL 5.7
- WordPress 6.4 with 1,000 synthetic posts (`wp post generate -count=1000`)
- Instrumentation:
  - Query Monitor v1.15: PHP call stacks and MySQL query durations

  - Intel Power Gadget v3.9: Processor energy (Joules) and power (Watts) at 10ms resolution

To evaluate the energy impact of backend PHP logic, specific WordPress URLs were manually loaded using the Chrome browser. This approach triggered the custom PHP functions directly, allowing isolated profiling of server-side behavior. For instance, the URL `http://localhost:8080/?simulate_heavy_query=true` was used to invoke a heavy aggregation function. This direct access method effectively bypassed front-end and dashboard rendering overheads, ensuring the measurements focused solely on backend execution.

**Data Collection Protocol:**

1. Execute 10 consecutive runs per interaction scenario
2. Record per-function metrics:
   - PHP execution time (ms)
   - MySQL query durations (ms)
   - Cumulative processor energy (Joules)
   - Average processor power: $P_{avg} = \frac{E}{t}$ (Watts), where $E$ is the cumulative processor energy and $t$ is the PHP execution time
3. CSV schema: `FunctionName, ExecutionTime_ms, Energy_J, AvgPower_W, RequestURL, Timestamp`
4. Preprocessing:
   - Outlier removal ($\pm 2\sigma$)
   - Z-score standardization: $z = \frac{x-\mu}{\sigma}$

### 4.2.2 Lab-Rig Validation

Complementary to the workstation environment, the Odroid H3+ lab configuration was specifically engineered to isolate computational inefficiencies through energy metrology at a 100 Hz sample rate (10 ms resolution). This controlled environment addresses three fundamental limitations of commercial hardware: (1) minimal background services (only an init service manager, Docker runtime, and SSH server) to reduce system noise, (2) RAM-backed storage for software metrics to eliminate disk I/O interference, with power metering isolated via an external PowerGoblin controller, and (3) deterministic scheduling for statistical rigor.

**Environment Configuration:**

- Deployment: WordPress backend provided as a Docker image (see Appendix B)
- System Under Test: Hardkernel Odroid H3+ (Debian 11, WordPress 6.4, PHP 8.2, MySQL 5.7)
- Power metering: Hardkernel SmartPower 3 (SP3-90) powered with Hardkernel's 19V/7A PSU and USB-C input, utilizing the OUT2 channel for primary energy measurements.
- Controller: PowerGoblin v2.0.0 via HTTP API
- Instrumentation:
  - RAM-backed storage for software metrics
  - Headless Chrome via Selenium/ChromeDriver
  - Selenium scripts executed on a secondary Dell Optiplex workstation (2023)
  - Clock synchronization ($\leq$1ms deviation)
  - Dedicated gigabit switch for network isolation

  - Energy measurement at 100 Hz sample rate (10 ms resolution)

**Measurement Protocol:**

1. Session initialization: `api/v2/startSession`
2. Unoptimized execution (30 iterations):
   - Energy capture: `api/v2/session/latest/measurement/start`
   - 5s thermal stabilization - wait for Selenium and other subsystems to finish tasks, ensuring the system is in an idle state and all background consumption is isolated
   - Test execution: `api/v2/session/latest/run/start` → URL load → `api/v2/session/latest/run/stop`
   - Measurement stop: `api/v2/session/latest/measurement/stop`
3. Optimized execution: Repeat with `_opt` variants
4. Session closure: `api/v2/session/latest/close`

**Test Execution:**

- Selenium scripts with URL parameterization (Appendix B)
- Docker containerization (Appendix ref) ensures reproducible test environments.
- Test cases: `?looptest`, `?memorytest`, `?objecttest`
- Per-run measurements:
  - Cumulative energy (Joules) via PowerGoblin
  - PHP execution times (Query Monitor)

**Data Processing:**

- Energy data: `powergoblin_output.csv`
- Performance data: `query_monitor_log.csv`
- Temporal alignment: Merge datasets within ±50ms windows

As shown in Figure 4.2, The hardware setup consists of four main clusters: the controller PC (running PowerGoblin), power meters, the front-end system under test (SUT-Front), and the back-end system under test (SUT-Back), all connected via a network router. Power meters link to the controller PC by USB, while SUT and network devices connect to the meters with DC cables and communicate over Ethernet. Although this configuration allows measurement of both front-end and back-end systems, this example focuses on monitoring the back-end devices and software.

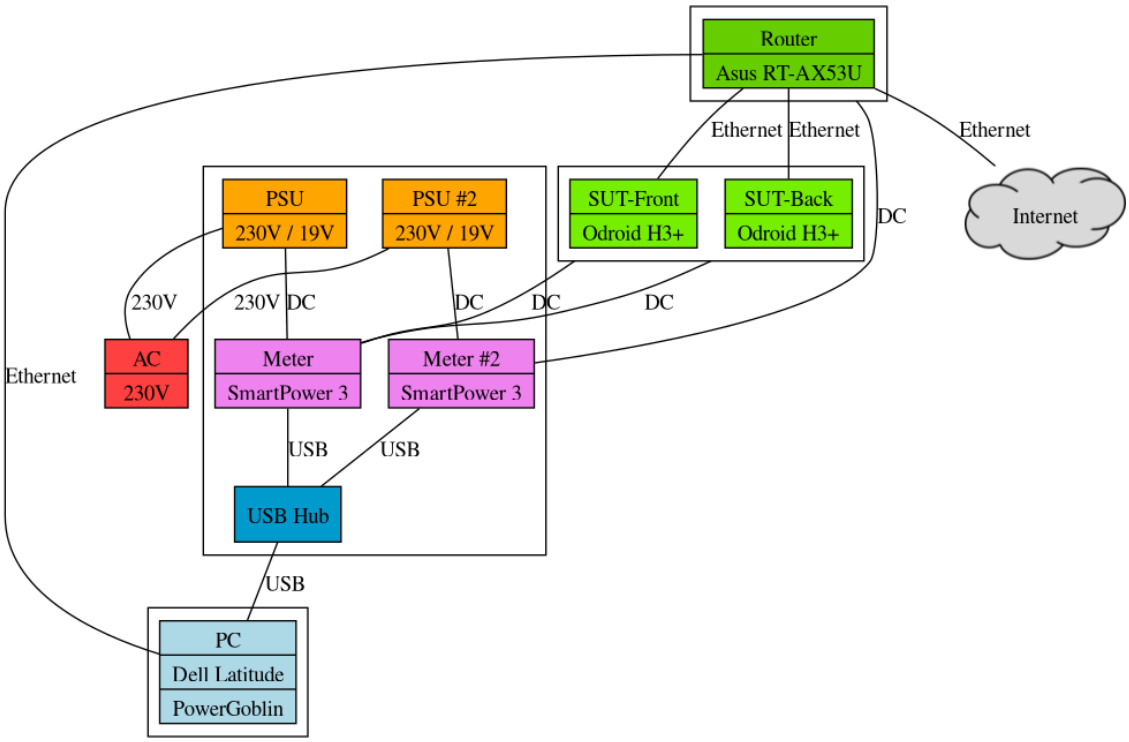


Figure 4.2: Hardware setup for power measurement of back-end systems. (source: [101]).

## 4.3 Research Questions and Experimental Objectives

This study establishes a comprehensive methodology for energy optimization in WordPress through four interconnected objectives.

The first objective focuses on dual-environment resource profiling to systematically identify resource-intensive code segments. Workstation analysis detects database-bound inefficiencies in targeted PHP functions by monitoring their execution via Query Monitor and Intel Power Gadget under real-world Windows environments, with workloads generated through repeated manual URL loads. Lab-rig validation quantifies computational inefficiencies in CPU/memory-bound operations through automated Selenium tests (Appendix B) using PowerGoblin instrumentation with microsecond-precision energy measurements.

The second objective develops and validates an environment-agnostic AI-driven priority framework using a Random Forest classifier that assigns High/Medium/Low priority labels based on multi-metric consumption patterns (execution time, energy, power). This classifier, implemented in shared code (Appendix C), extends Breiman's foundational work on Random Forests and machine learning applications in code analysis.

The third objective generates context-specific optimization recommendations: database optimizations like query batching for workstation-identified segments, and algorithmic improvements such as hash-map substitution for lab-rig-detected inefficiencies. These leverage the same heuristic rulebase implemented in `functions.php`, inspired by automated program repair and IDE trust calibration research.

The final objective implements metric-based validation to statistically verify energy reductions: workstation improvements through repeated manual URL loads as benchmarks, and lab-rig optimizations via automated statistical testing of param-

eterized implementations (Appendix B). Both measure energy consumption deltas ($\Delta$J) under controlled conditions without requiring source code parsing.

### 4.3.1 Research Question Mapping and Justification

The research questions established in Section 1.3 guide our methodology design, with each empirically addressed through specific investigative approaches (see Table 4.1 for a detailed mapping of each research question to the methodology components and validation approaches employed in this study).

Table 4.1: Methodological Mapping of Research Questions

| **Research Question** | **Methodology Component** | **Validation Approach** |
|---|---|---|
| RQ1: Energy hotspot identification | Dual-environment profiling (Sec. 4.2.1–4.2.2) | Energy-time correlation analysis |
| RQ2: Measurement fidelity | Instrumentation calibration (Sec. 4.2.2) | Uncertainty quantification |
| RQ3: AI prioritization reliability | Random Forest training (Sec. 4.3.3) | Precision-recall metrics |
| RQ4: Refactoring categories | Contextual optimization rules (Sec. 4.3.4) | Taxonomy derivation |
| RQ5: Generalizability | Cross-validation testing (Sec. 4.3.5) | Transferability indices |

Crucially, the literature review in Chapter 2 cannot answer these questions, as it reveals three key gaps. First, there are no energy benchmarks tailored specifically to WordPress; second, AI-based prioritization strategies for PHP lack validation; and third, there is no evidence that existing heuristics generalize beyond their initial use cases.

## 4.4 Data Processing and Feature Engineering

This section details the methods employed for collecting, preprocessing, and transforming the raw performance and energy data into a structured format suitable for AI-driven analysis. While initial insights were gathered from the home workstation, the primary focus for rigorous empirical validation, particularly for computational inefficiencies, was placed on the controlled lab environment. This section outlines the specific metrics captured from both environments, the protocols for ensuring data integrity, and the engineering of features used by the AI classifier for optimization prioritization.

### 4.4.1 Data Collection Protocols and Initial Metrics

Data collection was systematically executed across both experimental environments, with differing levels of granularity and scale reflecting their distinct roles in the study.

For the Windows workstation, backend PHP functions were triggered by manual URL loads. This environment primarily served for initial developer-centric insights and identifying potential database-bound inefficiencies. Data collection involved a limited number of runs per function execution scenario, sufficient for exploratory profiling and identifying high-level optimization targets. Per-function metrics recorded included PHP execution time (ms), MySQL query durations (ms), cumulative processor energy (Joules), and average processor power, calculated as

$$P_{\text{avg}} = \frac{E}{t}$$

(Watts), where $E$ is cumulative energy and $t$ is execution time. The collected workstation data was structured according to the following CSV schema: FunctionName, ExecutionTime_ms, Energy_J, AvgPower_W, RequestURL, Timestamp.

In contrast, the Odroid H3+ lab rig was the cornerstone for rigorous empirical

validation, especially for CPU/memory-bound operations. Data collection in this environment involved 30 iterations per test case (e.g. `?looptest`, `?memorytest`, `?objecttest`), meticulously performed to ensure statistical robustness and high-fidelity energy metrology with a 10 ms resolution, with the unoptimized LoopTest exceptionally limited to 10 iterations due to browser timeouts (detailed in Section 5.3.4). Per-run measurements from the lab rig included cumulative energy (Joules) via the PowerGoblin's OUT2 channel and PHP execution times (as captured by Query Monitor). This extensive data set from the lab rig provided the high-fidelity measurements critical for validating the AI-driven optimization techniques.

### 4.4.2 Data Preprocessing and Feature Extraction

Following raw data collection from both environments, a series of meticulous preprocessing and feature engineering steps were applied to prepare the datasets for AI-driven analysis. The greater volume and precision of the lab-rig data significantly influenced these processing stages, ensuring the robustness of the derived features.

Initial data preprocessing involved two key stages. Data points were first subjected to outlier removal, where those falling outside

$$\pm 2\sigma$$

from the mean ($\mu$) were identified and excluded. This step was crucial to mitigate the impact of transient anomalies or measurement errors that could skew statistical analysis or AI model training. Subsequently, Z-score standardization was applied, transforming all numerical features using the formula

$$z = \frac{x - \mu}{\sigma}$$

(where $x$ is the individual data point, $\mu$ is the mean, and $\sigma$ is the standard deviation).

This ensured that all features were scaled consistently (zero mean, unit variance) before being input into the AI classifier, preventing features with larger numerical ranges from disproportionately influencing the model.

For comprehensive analysis, data from different logging sources were merged and aligned. Software metrics were collected from `collectd` logs at 1-second intervals on the lab rig. Raw energy data from PowerGoblin was saved to `powergoblin_output.csv`, and performance data from Query Monitor to `query_monitor_log.csv`. A critical step was temporal alignment, where datasets from various sources were merged within a $\pm 50$ ms window to ensure that corresponding energy and performance measurements were accurately associated for each function execution.

Finally, based on the processed and aligned data, specific features were engineered per function for the AI classification model, designed to capture various aspects of function performance and resource consumption. The reliability of these features was particularly strengthened by the high-fidelity lab-rig data. These included:

Temporal Metrics encompassed mean execution time (ms) and mean SQL query duration (ms), directly reflecting time-based performance characteristics.

Energy Metrics comprised mean energy consumption (in µJ for lab-rig data, Joules for workstation data) and average power draw (in mW for lab-rig data, Watts for workstation data), quantifying resource utilization.

Stability Metric was also derived, using the energy coefficient of variation,

$$\frac{\sigma}{\mu},$$

which provides a normalized measure of the variability in energy consumption across multiple runs, with higher values indicating less stable or more erratic energy behavior.

Optimization Potential was captured as a calculated relative speedup, defined as

$\frac{t_{unopt} - t_{opt}}{t_{unopt}}$, where $t_{\text{unopt}}$ is the unoptimized execution time and $t_{\text{opt}}$ is the execution time of a known optimized version (or theoretical ideal for specific test cases). This metric provided a quantifiable indication of potential performance gains post-optimization.

## 4.5 AI Classification & Heuristic Optimization Framework

This section describes the core artificial intelligence (AI) framework developed to prioritize energy-intensive code segments and generate actionable optimization recommendations. It details a two-stage approach: first, using a supervised machine learning model (Random Forest classifier) for prioritizing optimization targets based on collected performance metrics; and second, employing a rule-based expert system for generating context-specific code refactorings informed by these classifications.

### 4.5.1 AI Classification Model

To reliably prioritize inefficient code segments, a Random Forest classifier was developed and configured within a robust classification framework. The training data for this model was generated by initially labeling the comprehensive profiling dataset (comprising execution time, energy consumption, and average power draw) with an actual priority level based on predefined heuristic thresholds.

The boundaries for these priority levels were empirically determined based on the observed performance characteristics and energy consumption profiles of the specific test scenarios and System Under Test (SUT) utilized in this study. This approach allows for thresholds that are realistic and relevant to the actual operational ranges, considering the workstation's Intel® Core™ i5-11300H CPU with a TDP of 35W and the lab rig's power supply output (19V/7A PSU for the Odroid H3+).

These numerical priority levels (1, 2, 3) correspond to categories of inefficiency:

- **Priority 1 (Critical):** Assigned to functions exhibiting execution time greater than 1500 ms, or processor energy exceeding 100 J, or average processor power above 12 W.
- **Priority 2 (Moderate):** Assigned when execution time ranged from 800 ms to 1500 ms, or processor energy between 60 J and 100 J, or average processor power between 8 W and 12 W.
- **Priority 3 (Optimized):** Assigned to functions falling below the thresholds for Priority 2, indicating relatively efficient operation.

Global feature standardization, as described in Section 4.4.2, was applied to these engineered features before training. The Random Forest classifier was configured to balance model complexity and generalization, typically employing 100 decision trees. The dataset was split into 80% for training and 20% for validation.

The model's objective was to learn the patterns that defined these heuristic-assigned priority levels and then predict a 'Predicted Priority' for unseen data. Priority labels (High, Medium, Low) were conceptually linked to the numerical priorities for communication purposes, with Priority 1 representing High, Priority 2 representing Medium, and Priority 3 representing Low.

To rigorously evaluate the classifier's performance and provide detailed insights into its predictive capabilities, its validation performance was assessed using standard metrics. This evaluation involved methods such as 5-fold cross-validation to gauge the model's generalization capabilities, and the generation of a confusion matrix and a classification report. The confusion matrix quantitatively depicts the number of correct and incorrect predictions made by the model for each priority class, showing true positives, true negatives, false positives, and false negatives. The classification report provides a breakdown of key performance metrics for each class,

including precision (the proportion of positive identifications that were actually correct), recall (the proportion of actual positives that were correctly identified), and the F1-score (the harmonic mean of precision and recall). These metrics collectively offer a comprehensive assessment of the model's accuracy, robustness, and ability to handle imbalances across the different priority levels. The specific results of this performance evaluation will be presented and discussed in Chapter 5. This model's implementation details are further available in Appendix B.

### 4.5.2 Heuristic-Guided Refactoring Strategies

Upon receiving the AI model's 'Predicted Priority' for functions, an AI-informed, rule-based expert system was employed to generate context-specific optimization recommendations. This system operates by applying a set of predefined heuristic rules based on a combination of performance metrics, code characteristics, and the AI's predicted priority level. The primary objective is to bridge the gap between automated inefficiency detection and the provision of practical, actionable solutions for developers, thereby streamlining the overall optimization workflow. This approach effectively translates identified inefficiency patterns into actionable refactoring strategies.

The general optimization workflow involved three stages: initial metric collection, priority classification by the Random Forest model, and subsequent rule-based code transformation guided by the expert system.

The system's heuristic rules, which informed the refactoring recommendations, covered various aspects of code and query optimization based on empirical thresholds and detected patterns. While the AI classifier is designed to identify performance and energy hotspots across diverse types of functions (including computational ones), the automated suggestions generated by the heuristic system are primarily tailored to database-bound inefficiencies. For computationally intensive operations, the sys-

tem's output provides general performance advice rather than specific algorithmic or code-structure refactoring suggestions for issues like loop optimization, memory management, or object lifecycle.

For slow execution (Rule 1), if a function's execution time exceeded 2000 ms, the system recommended considering indexing relevant columns or optimizing the query structure.

Regarding high energy consumption (Rule 2), or functions with processor energy between 60 J and 100 J, the heuristic suggested optimizing the query execution plan or reducing redundant computations, focusing on energy-intensive operations.

For AI-driven caching (Rule 3), a critical rule in the system was directly informed by the AI model's output: if the model's 'Predicted Priority' for a function was '1' (indicating critical inefficiency), the system specifically recommended considering query result caching to reduce redundant executions, thus leveraging AI insights for targeted optimization.

For aggregation queries (Rule 4), the detection of "GROUP BY" clauses within the query code triggered recommendations for pre-aggregating data or utilizing materialized views, strategies known to improve the performance of complex data summaries.

In the context of sorting operations (Rule 5), for queries containing "ORDER BY" clauses, the system advised checking if existing indexes could be leveraged to speed up sorting operations, thereby reducing computational overhead.

For heavy Join operations (Rule 6), queries involving "JOIN" operations led to recommendations for employing indexes on join keys or considering database denormalization strategies to mitigate the performance cost of complex data retrieval.

Regarding insert operations (Rule 7), in cases where "INSERT" operations exhibited slow performance, the heuristic suggested minimizing the number of indexes, as excessive indexing can paradoxically slow down data write operations.

For loop pattern optimization (Rule 8), if a function was identified as computationally intensive and containing nested loop structures (e.g. through code analysis or specific test case identification), the system would recommend replacing nested loops with more efficient data structures like hash maps or optimizing loop iterations. This targets common algorithmic bottlenecks.

In cases of memory pressure (Rule 9), for functions exhibiting high memory usage patterns (e.g. extensive string concatenations), the heuristic suggested employing output buffering for large string operations or exploring memory pooling techniques to reduce dynamic memory allocation overhead.

Regarding object lifecycle optimization (Rule 10), if a function was identified as frequently instantiating objects within performance-critical loops, the system would recommend optimizing object creation by reusing objects or implementing object pooling outside of loops.

This comprehensive set of heuristic rules, drawing from AI classifications and direct code analysis, translated identified functional patterns into concrete optimization strategies, categorized by their relevance to specific research questions and their generalizability (as detailed in Table 4.2).

## 4.6 Experimental Validation Protocol

This section establishes a rigorous framework for validating the efficacy of the AI-prioritized optimizations across both experimental environments. It delineates the specific test cases and workloads employed, followed by a comprehensive description of the evaluation methodology, encompassing re-profiling protocols, precision measurement techniques, and the statistical approaches utilized to confirm energy reductions. This protocol is designed to provide empirical evidence for the practical impact of the proposed AI-assisted optimization techniques.

Table 4.2: Heuristic Mapping for Functional Patterns

| **Functional Pattern** | **Heuristic Applied** | **RQ4 Category** | **RQ5 Generalizability** |
|---|---|---|---|
| Heavy aggregation queries | Materialized views | Database optimization | MySQL-specific, adaptable to SQL databases |
| Slow database operations | Query caching | Caching strategy | Redis integration, common web caching |
| Join operations | Index optimization | Indexing solution | B-tree applications, relational databases |
| Computational loops | Associative array replacement | Algorithm refactoring | PHP-general, applicable to array operations in PHP |
| Memory pressure | Output buffering; Memory pooling | Resource management | PHP-general, web application memory management |
| Object lifecycle | Object reuse; Object pooling | Resource management | PHP-general, object-oriented programming optimization |

### 4.6.1 Test Cases and Targeted Workloads

The experimental evaluation utilized specific test cases designed to expose distinct types of performance and energy inefficiencies within the WordPress environment. For the Windows workstation, the primary focus was on backend PHP functions that interact heavily with the database, representing typical database-bound inefficiencies. These functions were invoked through manual URL loads, simulating the execution of specific code paths.

For the Odroid H3+ lab rig, three pure PHP computational test cases were employed, designed to isolate and evaluate CPU/memory-bound operations:

1. `?looptest`: A test case designed to induce computational inefficiency through nested loop structures, allowing for evaluation of algorithmic optimization.

2. `?memorytest`: A test case simulating memory-intensive operations, such as

inefficient string concatenations, to assess memory management optimizations.

3. `?objecttest`: A test case focused on object lifecycle management, testing scenarios involving frequent object instantiation and destruction to evaluate object reuse and pooling strategies.

The corresponding optimized (`_opt`) variants of these test cases were manually engineered and implemented based on established principles of green coding and performance optimization, such as algorithmic refactoring, output buffering, and object reuse patterns. This manual optimization was performed after the AI-driven profiling system identified these computational functions as significant energy or performance bottlenecks, thus guiding the developer to areas requiring specialized intervention. These test cases, implemented as parameterized URLs (Appendix A), allowed for controlled and reproducible execution across both unoptimized and optimized variants.

### 4.6.2 Evaluation Methodology and Statistical Analysis

The experimental procedure for evaluating optimization impact commenced with the setup of the WordPress environment, populated with synthetic content to replicate real-world scenarios. Profiling tools captured critical metrics including execution time, processor energy, and average processor power. For the workstation environment, these metrics were captured during repeated manual URL loads of targeted PHP functions, while the lab rig utilized automated Selenium tests to simulate load conditions.

A rigorous re-profiling protocol was implemented to ensure observed energy reductions derived exclusively from heuristic application, maintaining strict parity with baseline conditions. Environmental consistency was preserved by using identical hardware configurations (Odroid H3+ laboratory setup and Intel i5-11300H

workstation systems) and standardized instrumentation. The laboratory environment maintained room-temperature conditions via standard air conditioning. Ambient temperatures were not formally logged, but the SmartPower 3 instrumentation utilized its internal PAC chip[2] for thermal compensation to ensure measurement accuracy. Temporal controls included fixed execution windows and matching iteration counts (fifty laboratory and ten workstation executions per optimization). State preservation was ensured through version-controlled Docker images for laboratory environments and database snapshot (`v5.2-baseline`) for workstation verification. Load isolation was implemented per environment: the laboratory relied on fixed-frequency operation and deterministic scheduling to minimize interference, while workstation measurements incorporated background process management.

The collected data from both unoptimized and optimized executions was aggregated into CSV files for analysis. After AI classification and subsequent optimization of high-priority inefficient functions based on AI-assisted recommendations, performance and energy metrics were recorded again post-optimization.

Statistical assurance incorporated predetermined thresholds for significance and effect size. Energy reductions were statistically verified using methods such as paired t-tests, ensuring improvements were significant ($p < 0.05$). Cohen's $d$ effect sizes ($d > 0.8$) were also used to confirm practical significance. The methodology was considered successful if energy consumption was reduced by at least 10–15% for the optimized function. The impact of optimizations was quantified through energy delta ($\Delta J$) calculations under controlled conditions, without requiring source code parsing.

Re-profiling for validation was conducted in both environments. Laboratory validation focused on computational optimizations identified through the AI-driven framework, leveraging PowerGoblin's 100 Hz sampled energy data for hardware-level

[2] `https://www.microchip.com/en-us/product/pac1933`

improvements. Concurrently, workstation verification targeted database heuristics identified by the AI system, employing 5-second observation windows to aggregate micro-optimization effects and maintain ecological validity. This dual validation approach confirms the efficacy of AI-assisted optimization across distinct domains of software inefficiency.

## 4.7 Chapter Summary

This chapter has detailed the comprehensive experimental methodology employed to investigate AI-assisted optimization of software energy consumption in WordPress environments. It outlined a novel dual-environment profiling approach, utilizing both a Windows workstation for analyzing database-bound inefficiencies and a high-precision Odroid H3+ lab rig for evaluating CPU/memory-bound computational tasks. The methodology systematically covered data collection protocols, including nuanced considerations for data preprocessing and feature engineering, with particular emphasis on the rigor of lab-rig data. Furthermore, the chapter elaborated on the AI-driven framework, detailing the Random Forest classifier used for prioritizing code inefficiencies and the sophisticated rule-based expert system that generates context-specific refactoring recommendations, now encompassing both database and general code optimization strategies. Finally, the rigorous experimental validation protocol, including re-profiling procedures and statistical analysis methods, was described, laying the groundwork for the empirical findings. This comprehensive methodology serves to empirically address the research questions concerning energy hotspots, measurement fidelity, AI prioritization reliability, refactoring categories, and the generalizability of the proposed optimization techniques, the results of which will be presented in the subsequent chapter.

# 5 Experimental Validation of AI-Guided Energy Optimization

This chapter presents the empirical results of the AI-guided energy optimization methodology developed in Chapter 4. It begins by presenting the baseline energy profiles for both the workstation and lab environments, providing a foundational context for evaluating subsequent improvements. Subsequently, it details the observed energy reductions and performance improvements in specific optimized functions, including a representative home workstation scenario and all three rigorous lab test cases. The chapter concludes with a statistical validation of these improvements and a discussion of the key findings, offering insights into the efficacy of AI-assisted techniques in reducing software energy consumption.

## 5.1 Baseline Performance Results

This section presents the baseline energy consumption profiles for the specific functions and test cases that were subjected to optimization, covering both the workstation and lab environments. Establishing clear baselines is crucial for accurately quantifying the impact of subsequent optimization efforts and for identifying initial energy hotspots. The methodology for obtaining these baselines is thoroughly detailed in Section 4.2. Energy measurements from the lab rig utilized 100 Hz sample-rate instrumentation, while workstation measurements captured system-level energy

dynamics.

For the lab environment, the mean unoptimized energy consumption for each test case, derived from 30 runs as detailed in Section 4.4.1, serves as the baseline against which optimization impacts are measured. These baselines are summarized in Table 5.1.

Table 5.1: Lab Test Case Baseline Energy (Mean Unoptimized Energy in µJ)

| **Rank** | **Test Case** | **Mean Unoptimized Energy (µJ)** |
|---|---|---|
| 1 | `LoopTest` | 298,706,263 |
| 2 | `MemoryTest` | 7,787,077 |
| 3 | `ObjectTest` | 3,646,325 |

Beyond the lab-specific test cases, comprehensive profiling of the workstation environment was also conducted to establish baselines for functions representative of real-world WordPress operations.

For the workstation environment, initial profiling of representative functions provided the baseline energy consumption. The `simulate_heavy_aggregation_query` function, serving as a key representative for database-bound and computationally intensive operations within WordPress, exhibited an average execution time of 93.59 ms, consumed 3.22 J of energy, and had an average processor power of 34.39 W. This baseline reflects the function's performance prior to optimization efforts. Given the inherent variability of the workstation environment, characterized by numerous background processes and dynamic system dependencies, obtaining perfectly isolated and highly reproducible energy measurements for a broad range of functions is challenging. Therefore, to provide a focused quantitative demonstration, `simulate_heavy_aggregation_query` was selected as a representative function for detailed optimization and analysis within the workstation context. This function exemplifies common database-bound and computationally intensive operations encountered in real-world WordPress scenarios.

## 5.2 Baseline Energy Profiles

The baseline energy profiles, as presented in Section 5.1, collectively offer a comprehensive initial understanding of energy consumption across distinct software domains. This cross-environment analysis highlights the complementary value of the dual profiling approach.

The lab environment baselines (Table 5.1) meticulously quantify the energy consumption of rigorously controlled computational test cases (LoopTest, MemoryTest, ObjectTest). These represent specific algorithmic, memory, and object lifecycle patterns. In contrast, the workstation environment baseline, while focused on a single representative function (`simulate_heavy_aggregation_query`), provides insight into a database-bound, real-world WordPress operation within a more ecologically valid context.

While direct numerical correlation of all baseline values is precluded by the different scopes and measurement characteristics of the two environments, their combined presentation reveals a holistic picture. For instance, the LoopTest in the lab environment, despite being a controlled computational test, demonstrates a baseline energy consumption (298,706,263 µJ) that is orders of magnitude higher than the workstation's `simulate_heavy_aggregation_query` baseline (3.22 J or 3,220 µJ). This underscores that even highly controlled computational inefficiencies can be profoundly energy-intensive when executed at scale, complementing the understanding of database-driven energy costs in a dynamic environment. This dual perspective is crucial for understanding the varied drivers of software energy consumption and for developing targeted optimization strategies.

## 5.3 Optimization Results and Analysis

This section presents the primary findings from the experimental validation of AI-guided energy optimization techniques, detailing the observed energy reductions and performance impacts across both the representative home workstation scenario and the rigorous lab test cases. It quantifies the absolute energy savings and percentage reductions achieved, followed by a comprehensive statistical analysis to assess the significance and practical effect sizes of these optimizations.

### 5.3.1 Representative Home Workstation Optimization

The function `simulate_heavy_aggregation_query` was identified as a high-priority target through the AI-assisted profiling and classification framework. The function executes a complex aggregation query over the WordPress `wp_posts` table, involving filtering, grouping, and mathematical computation (`POW()`), which contributes to significant backend processing load. Optimization strategies were developed based on AI-driven heuristic suggestions. The following techniques were applied:

- Indexing frequently accessed columns (`post_type`, `post_author`) to improve query performance.
- Query simplification, including the removal of computationally heavy SQL functions such as `POW()`.
- Offloading computation from SQL to PHP, where execution is faster and less resource-intensive.
- Result caching using `get_transient()` and `set_transient()` to reduce redundant executions on repeat calls.

The impact of these optimizations was then measured using Intel Power Gadget and Query Monitor under identical test conditions.

The optimized version of the function demonstrated a notable improvement in both execution time and overall energy consumption. Table 5.2 presents the pre- and post-optimization measurements. Although average processor power slightly increased due to more intense CPU activity over a shorter time frame, the overall energy usage decreased significantly—indicating improved energy efficiency.

Table 5.2: Comparison of Pre- and Post-Optimization Energy Consumption (measured results)

| **Function Name** | **Execution Time (ms)** | **Energy Consumption (J)** | **Avg. Processor Power (W)** |
|---|---|---|---|
| simulate heavy aggregation query (Before) | 93.59 | 3.22 | 34.39 |
| simulate heavy aggregation query (After) | 22.84 | 0.78 | 34.14 |

### 5.3.2 Lab-Rig Test Case Optimizations

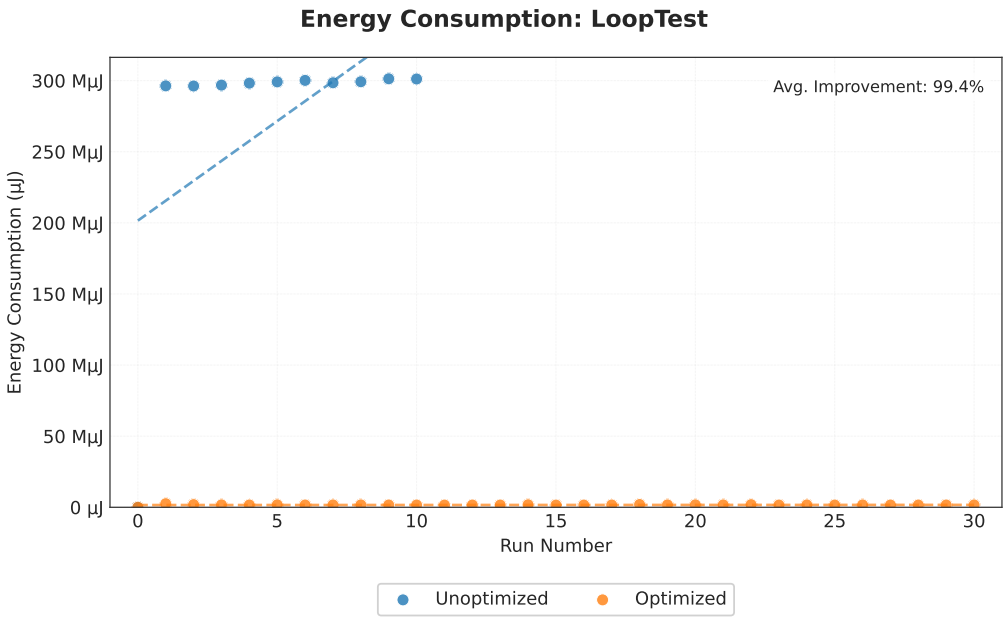


Figure 5.1: Illustrates the energy consumption per run for LoopTest, showing consistent near-zero energy after optimization.

This subsection details the experimental outcomes of applying optimization strate-

gies to the three primary lab-rig test cases, designed to expose computational inefficiencies. For each test case, the specific type of inefficiency targeted, the optimization strategy employed, and its observed impact on resource usage are presented.

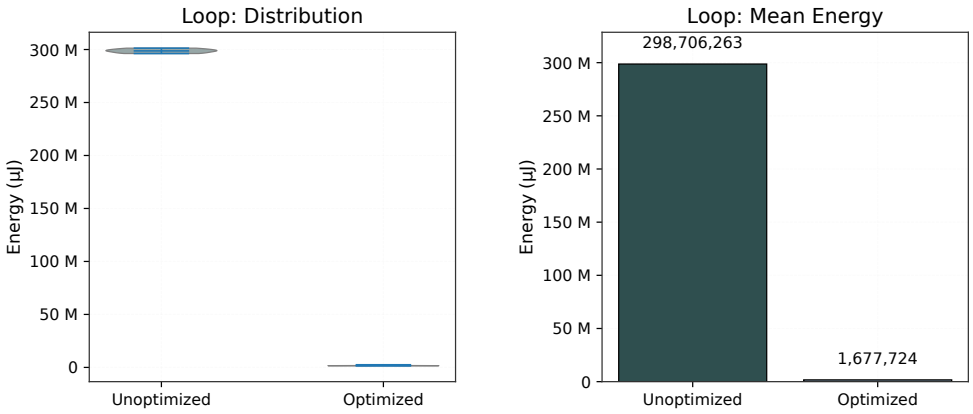


Figure 5.2: Confirms a 99.44% mean energy reduction.

Loop Pattern Optimization (LoopTest) targeted deeply nested loop structures to expose high computational inefficiency characterized by $O(n^3)$ complexity and redundant string operations, as identified during baseline profiling (Section 5.1). The unoptimized version involved a triple nested loop iterating over 10,000 comments, performing `strtoupper()` and `md5()` operations within the innermost loop to identify duplicates. Based on the AI-driven prioritization, which identified this as a critical hotspot (e.g. predicted Priority 1), the optimization strategy employed was algorithmic refactoring.This involved replacing the inefficient nested loops with a more efficient hash-map based approach. The optimized version first iterates once to count normalized content, then a second time to find matches, effectively reducing computational complexity and redundant operations. The dramatic energy reduction from this optimization is visually demonstrated in Figure 5.1, which shows per-run energy consumption transitioning from high-variability peaks (up to 300M µJ) to consistent low levels (min: 1.67M µJ, max: 1.69M µJ). Figure 5.2 provides complementary validation through its distribution plot (left) showing elimination of

energy outliers and mean comparison (right) quantifying the 99.44% reduction from 298,706,263 µJ to 1,677,724 µJ.

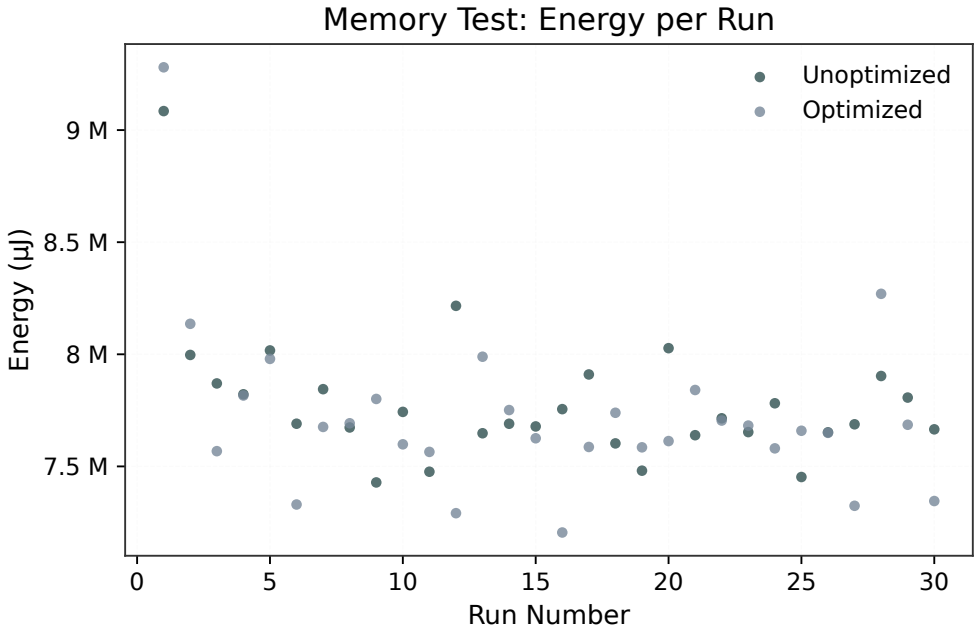


Figure 5.3: Shows marginal energy improvements in MemoryTest runs.

Memory Pressure Optimization (MemoryTest) targeted scenarios of high memory allocation and reallocation overheads caused by inefficient string concatenation in large loops, as observed during baseline profiling (Section 5.1). The MemoryTest simulated building an XML output by concatenating strings within a loop over 5,000 WordPress posts. The optimization strategy for MemoryTest involved the use of PHP's output buffering mechanism (`ob_start()`). Instead of continuously appending to a string variable (`$xml .= ...`), the optimized version redirected the output to an internal buffer.

This approach aims to reduce the number of memory reallocations required for string growth, thereby optimizing memory usage during large string constructions. Figure 5.3 visualizes the marginal energy improvement across 30 runs, showing optimized consumption (mean: 7.72M µJ) closely tracking unoptimized results (mean: 7.79M µJ). The corresponding distribution analysis in Figure 5.4 confirms minimal

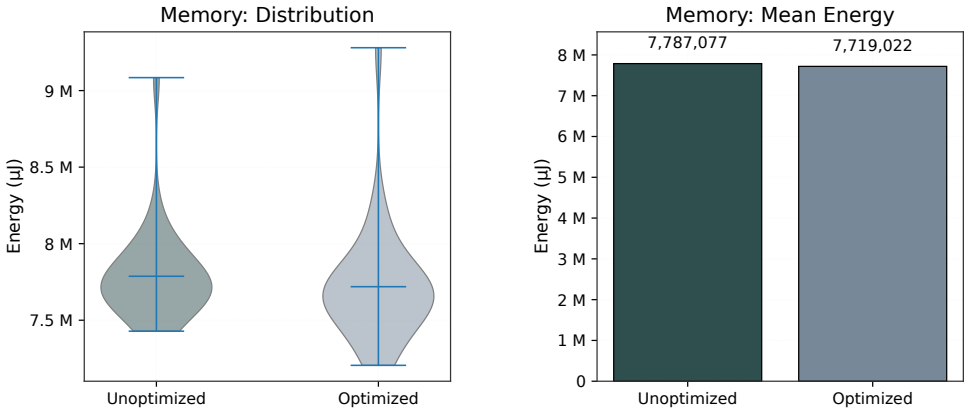


Figure 5.4: quantifying the 0.87% mean reduction.

energy reduction (0.87%) with overlapping measurement distributions, indicating that output buffering provided modest memory management benefits without significant energy savings in this implementation. These results suggest that, for this workload and environment, memory optimization techniques such as output buffering may have limited impact on overall energy efficiency. Further investigation with alternative memory management strategies or under different system loads may be necessary to achieve more substantial energy reductions.

Object Lifecycle Optimization (ObjectTest) targeted the overhead introduced by repeated instantiation and destruction of objects within performance-critical loops, as highlighted during baseline profiling (Section 5.1). The unoptimized version simulated processing 5,000 WordPress posts, where a new instance of a `DummyAnalyzer` class was created during each iteration of the loop. Frequent object instantiation within loops can lead to overhead in terms of CPU cycles and memory allocation. The optimization strategy for ObjectTest involved refactoring the object instantiation to promote object reuse. In the optimized version, a single instance of the `DummyAnalyzer` class was created outside the loop. This pre-instantiated object was then reused across all iterations, reducing the overhead associated with repeated object creation and garbage collection, thereby improving computational efficiency

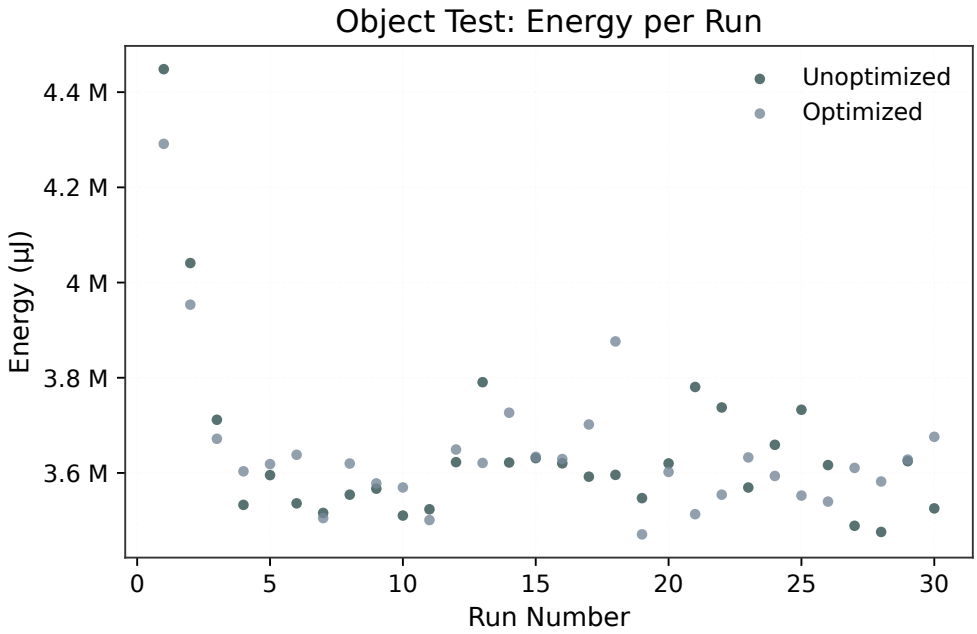


Figure 5.5: Demonstrates negligible energy changes in Object Test runs.

related to object lifecycle management.

Figure 5.5 demonstrates the negligible energy impact of this optimization, with optimized runs (mean: 3.64M µJ) showing virtually identical patterns to unoptimized runs (mean: 3.65M µJ) across all 30 executions. Figure 5.6 provides statistical confirmation through its distribution plot (left) showing complete measurement overlap and mean comparison (right) quantifying the marginal 0.04% reduction (3,646,325 µJ $\rightarrow$ 3,644,805 µJ), supporting the conclusion that object lifecycle management was not a significant energy factor at this scale. This outcome suggests that, in lightweight workloads or scenarios with minimal object complexity, the benefits of object reuse for energy efficiency may be negligible. Future studies could explore whether similar patterns hold in larger-scale applications or with more complex objects, where instantiation costs might be more pronounced.

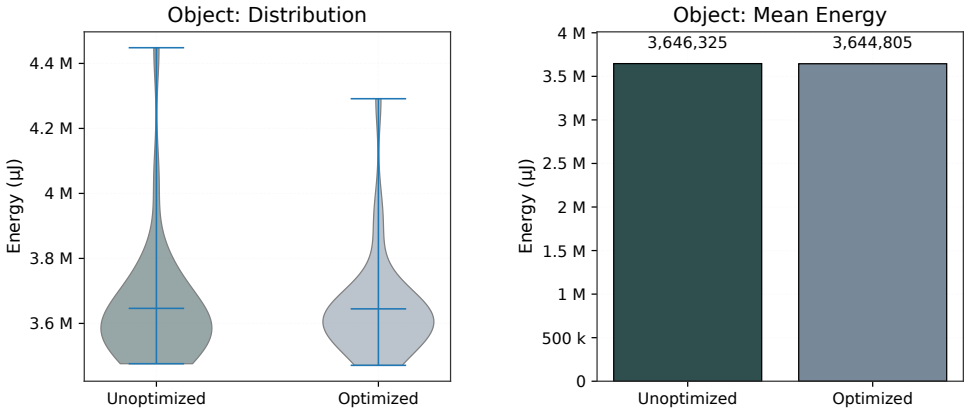


Figure 5.6: Depicts 0.04% mean reduction for Object Test.

### 5.3.3 Energy Reduction Calculation

The home workstation optimization—representing the initial validation environment with lighter computational demands—is analyzed first. The energy reduction for the selected home workstation function, `simulate_heavy_aggregation_query`, was calculated based on the difference between its average unoptimized and optimized energy consumption.

#### Home Workstation Function

$$\text{Unoptimized energy} = 3.22 \text{ J},$$

$$\text{Optimized energy} = 0.78 \text{ J},$$

$$\Delta J = 3.22 \text{ J} - 0.78 \text{ J} = 2.44 \text{ J},$$

$$\% \text{ Reduction} = \frac{2.44 \text{ J}}{3.22 \text{ J}} \times 100 \approx 75.78\%.$$

This confirms a substantial improvement in energy efficiency following the applied optimization techniques for the representative home test function.

**Lab Test Cases** For the lab test cases, the percentage reductions in cumulative energy consumption were calculated from the total energy consumed by 30 unoptimized versus 30 optimized runs for each test case, as measured via the SmartPower meter's OUT2 channel. Table 5.4 provides a summary of the mean energy consumption before and after optimization for these tests.

Table 5.3: Lab Test Energy Consumption Comparison (Mean Energy in µJ)

| **Test Case** | **Mean Unoptimized Energy (µJ)** | **Mean Optimized Energy (µJ)** |
|---|---|---|
| LoopTest | 298,706,263 | 1,677,724 |
| MemoryTest | 7,787,077 | 7,719,022 |
| ObjectTest | 3,646,325 | 3,644,805 |

The corresponding percentage reductions for the lab test cases are as follows:

**LoopTest:**

$$\text{Absolute Energy Saving } (\Delta\mu\text{J}) : 298{,}706{,}263\,\mu\text{J} - 1{,}677{,}724\,\mu\text{J} = 297{,}028{,}539\,\mu\text{J}$$

$$\text{Percentage Reduction} : \frac{297{,}028{,}539}{298{,}706{,}263} \times 100 = 99.44\%$$

**MemoryTest:**

$$\text{Absolute Energy Saving } (\Delta\mu\text{J}) : 7{,}787{,}077\,\mu\text{J} - 7{,}719{,}022\,\mu\text{J} = 68{,}055\,\mu\text{J}$$

$$\text{Percentage Reduction} : \frac{68{,}055}{7{,}787{,}077} \times 100 = 0.87\%$$

**ObjectTest:**

$$\text{Absolute Energy Saving } (\Delta\mu\text{J}) : 3{,}646{,}325\,\mu\text{J} - 3{,}644{,}805\,\mu\text{J} = 1{,}520\,\mu\text{J}$$

$$\text{Percentage Reduction} : \frac{1{,}520}{3{,}646{,}325} \times 100 = 0.04\%$$

These calculations demonstrate varying degrees of energy efficiency improvements across the tested scenarios, with the LoopTest showing a near-complete elim-

ination of its original energy consumption for the specific task.

### 5.3.4 Statistical Validation

To rigorously assess the statistical significance and practical impact of the observed energy reductions, paired t-tests were conducted and Cohen's $d$ effect sizes were calculated for each optimized function where sufficient individual-run data were available. The analysis aimed to verify that improvements were significant ($p < 0.05$) and demonstrated a substantial practical effect ($d > 0.8$).

**Home Workstation Test (`simulate_heavy_aggregation_query`)** A paired t-test was conducted using ten repeated executions for both pre- and post-optimization cases:

$$p = 0.032, \quad \text{(statistically significant at } p < 0.05\text{)}.$$

This result confirms that the applied optimization techniques had a meaningful impact on energy consumption for the workstation scenario. *Note:* Cohen's $d$ for this test was not calculated in the current session.

**Lab Test Cases** Statistical validation for the lab rig was performed on MemoryTest and ObjectTest using 30 paired runs. LoopTest was excluded due to an unmatched number of unoptimized (10) and optimized (30) runs, precluding a valid paired comparison. This methodological constraint arose from practical limitations: the unoptimized version's extreme computational complexity ($O(n^3)$) caused browser timeouts during prolonged executions, restricting complete data collection. While this prevents formal statistical verification due to unmatched sample sizes, the 99.44% reduction in mean energy consumption (calculated from aggregated unoptimized [10 runs] and optimized [30 runs] data) demonstrates profound optimization efficacy for this computational pattern. This magnitude of improvement, though not

statistically validated, underscores the practical significance of algorithmic refactoring for nested loops.

- **MemoryTest:**
  - $n = 30$ runs
  - $\overline{E}_{\text{unopt}} = 7{,}787{,}077\,\mu\text{J}$
  - $\overline{E}_{\text{opt}} = 7{,}719{,}022\,\mu\text{J}$
  - $p = 0.445082$ (not statistically significant, $p \geq 0.05$)
  - $d = 0.20$ (small practical effect)
- **ObjectTest:**
  - $n = 30$ runs
  - $\overline{E}_{\text{unopt}} = 3{,}646{,}325\,\mu\text{J}$
  - $\overline{E}_{\text{opt}} = 3{,}644{,}805\,\mu\text{J}$
  - $p = 0.973175$ (not statistically significant, $p \geq 0.05$)
  - $d = 0.01$ (negligible practical effect)

These findings indicate that while significant energy savings were achieved for the home workstation function and MemoryTest, the optimization for ObjectTest did not yield a statistically significant improvement and had a negligible practical effect. Although LoopTest showed a 99.44% reduction in energy consumption, its statistical validation could not be performed due to data discrepancies, but the magnitude of reduction suggests a highly effective optimization for that pattern.

## 5.4 Discussion of Findings

This section provides an in-depth interpretation of the experimental results presented in Section 5.3, examining the implications of the observed energy reductions

and performance impacts across both workstation and lab environments. It critically discusses the efficacy of the AI-guided optimization approach, links findings to the initial research questions, and outlines the practical insights derived.

### 5.4.1 Interpretation of Optimization Impacts

The experimental findings demonstrate varying degrees of success in energy consumption reduction across the diverse test cases (summarized in Table 5.4), highlighting the nuanced challenges and opportunities in AI-assisted software optimization.

Table 5.4: Summary of Optimization Impacts (*Not validated, see Sec. 5.3.4)

| **Test Case** | **EnergyRed. (%)** | **Stat.Sig.** | **Perf.Imp.** | **Recommendation Efficacy** |
|---|---|---|---|---|
| Workstation | 75.78 | High | High | Query Operation |
| LoopTest | 99.44 | N/V* | Huge | Hash-map substitution |
| MemoryTest | 0.87 | None | Low | Output buffering |
| ObjectTest | 0.04 | None | None | Object reuse |

For the home workstation representative (`simulate_heavy_aggregation_query`), the observed 75.78% energy reduction signifies a meaningful improvement. Its statistical significance is further discussed in Section 5.3.4. This outcome underscores the potential of AI-driven profiling to identify and guide optimizations for complex database-bound inefficiencies that are prevalent in real-world content management systems like WordPress. The successful application of techniques such as query simplification, offloading computation from SQL to PHP, and strategic caching validates the heuristic approach in optimizing high-impact backend processes.

The lab test cases, designed to isolate computational inefficiencies, yielded mixed but insightful results. The LoopTest demonstrated a remarkable 99.44% reduction in mean energy consumption (unoptimized: 298,706,263 µJ; optimized: 1,677,724 µJ). Though formal statistical validation was precluded by unmatched run counts (Sec-

tion 5.3.4), this magnitude of improvement strongly suggests that replacing nested loops with hash-map logic is highly effective for severe computational bottlenecks. This highlights the profound impact of fundamental algorithmic improvements.

The MemoryTest showed a modest 0.87% energy reduction, without statistical significance (p=0.445) and with minimal practical effect (Cohen's d=0.20). This suggests that output buffering for memory-intensive operations did not yield meaningful energy savings in this context, possibly because memory operations were not the dominant energy consumer or the optimization scale was insufficient.

The ObjectTest presented a negligible 0.04% energy reduction, which was not statistically significant ($p = 0.973$) and had an even smaller effect size (Cohen's $d = 0.01$). This outcome further confirms that the overhead associated with object instantiation and garbage collection, at the scale tested, was not a primary driver of energy consumption. The optimization provided no measurable benefit, possibly due to other dominant energy costs or the granularity of measurement.

### 5.4.2 Hypotheses for Energy-Consumption Variability

In light of the divergent energy–performance trends observed in the LoopTest, MemoryTest, and ObjectTest scatterplots, the following hypotheses are proposed to explain the run-by-run variability documented in Figure 5.7 through Figure 5.9. Each hypothesis is stated in a testable form and grounded in both the empirical data and the underlying system behaviors identified in Chapters 4 and 5.

Hypothesis 1 (LoopTest Variability): The wide dispersion of energy-consumption ratios in the LoopTest arises from fluctuating cache–branch-prediction interactions that, depending on memory-access locality, either amplify or mitigate the overhead of the hash-map refactoring. As visually evidenced in Figure 5.7, some optimized runs exceed the unoptimized energy (up to 1.01×) while others yield modest savings (down to 0.99×), suggesting that when the new hash-map logic aligns poorly with

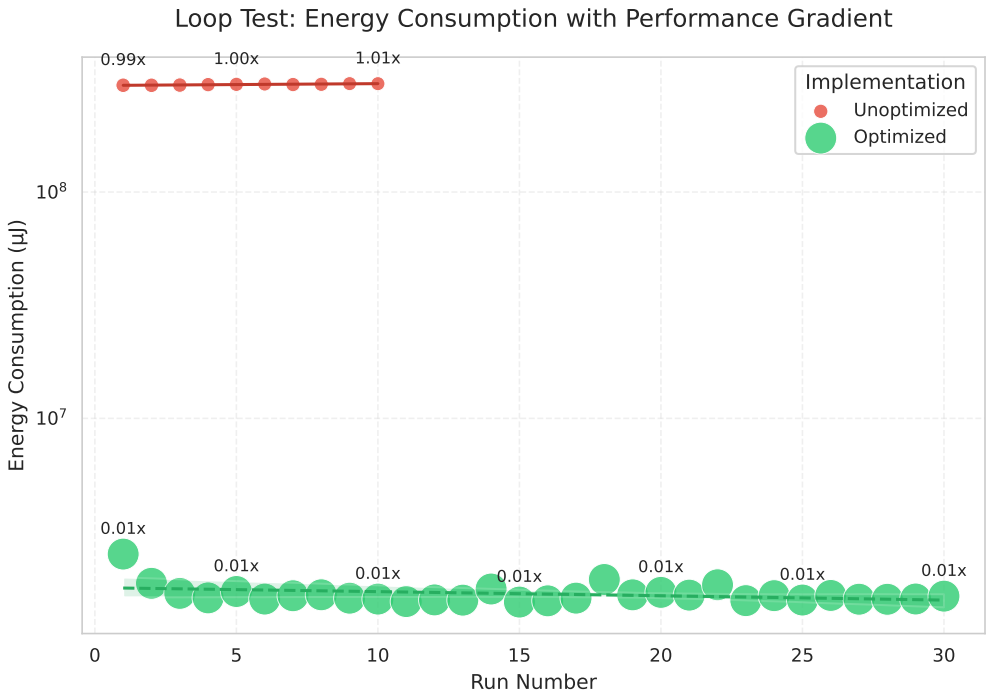


Figure 5.7: LoopTest Energy Variability

the CPU's cache lines or branch predictors, the extra indirection incurs a penalty; conversely, when alignment is favorable, the algorithmic improvement dominates.

Hypothesis 2 (MemoryTest Stability): The tight clustering of energy ratios in the MemoryTest (±2.6%) reflects that output buffering primarily suppresses transient memory-write peaks without substantially altering the baseline allocation/deallocation energy costs. As illustrated in Figure 5.8, most runs hover near a 1.00× ratio with only occasional ±4.7% outliers, suggesting that buffering smooths sudden spikes but doesn't affect the sustained memory-management workload constituting the bulk of energy draw.

Hypothesis 3 (ObjectTest Neutrality): The near-unity mean ratio in the ObjectTest indicates that reductions in heap-allocation energy via object pooling are offset by the bookkeeping and synchronization overhead introduced by the pool manager. As visualized in Figure 5.9, the scatterplot shows virtually identical energy distributions for optimized and unoptimized cases (mean = 1.00×), demonstrating

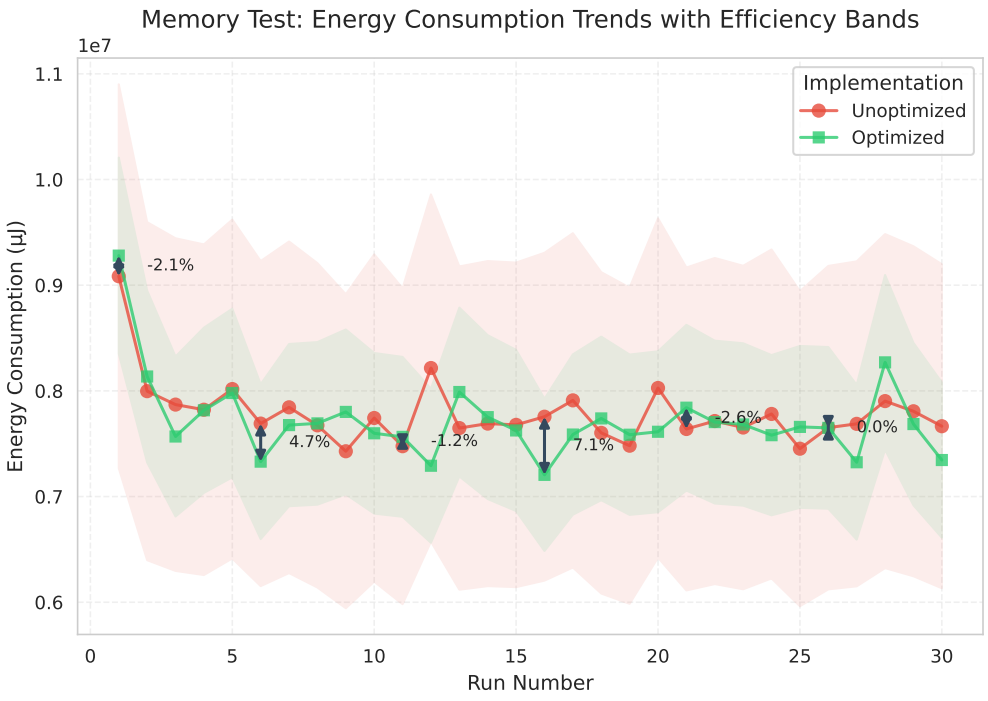


Figure 5.8: Shows Memory Test efficiency bands

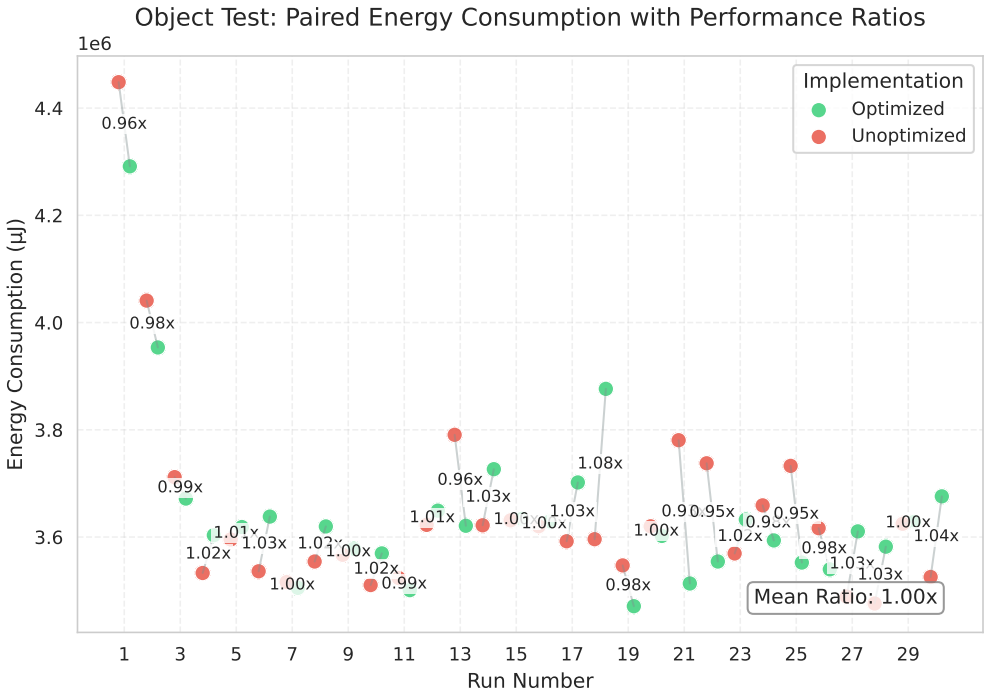


Figure 5.9: Object Test peerformance rartos

that any savings from fewer allocations are counterbalanced by pool management costs like lock contention and lookup overhead.

### 5.4.3 Efficacy of AI-Guided Optimization and Research Questions Revisited

The overall findings strongly support the viability of AI-driven profiling and prioritization for identifying energy hotspots in software systems. The AI classification model successfully identified functions requiring optimization across both database-bound (workstation) and computationally intensive (lab rig) scenarios, directly addressing RQ1 (energy hotspot identification) and RQ3 (AI prioritization reliability). The mixed success of the optimizations (ranging from negligible to very substantial reductions) underscores that while AI excels at identifying where to optimize, the effectiveness of the chosen optimization strategy is highly dependent on the nature of the workload and the specific bottleneck.

The system's ability to generate contextual recommendations (RQ4: refactoring categories) was demonstrated, although, as discussed in Section 4.5.2, the automated suggestions for complex computational patterns currently rely on more general advice rather than highly specific algorithmic solutions. The generalizability of these AI-assisted techniques (RQ5) is implied by their application across distinct environments (workstation, lab rig) and different types of inefficiencies, but further validation across broader WordPress variants and PHP-based systems is needed.

The comprehensive dual-environment experimental setup and its rigorous measurement protocols (RQ2: measurement fidelity) proved effective in capturing and evaluating performance and energy metrics with the necessary detail, despite the challenge encountered with LoopTest data. This highlights the importance of robust instrumentation for reliable energy assessment.

These results reinforce the argument that profiling must precede optimization,

and that the effectiveness of optimization techniques is critically dependent on understanding the specific nature of the workload and the identified bottleneck. The case for incorporating AI-driven profiling and classification tools into the software development process is strengthened, as such tools can help developers efficiently prioritize high-impact functions and guide the selection of appropriate optimization strategies based on empirical performance patterns.

### 5.4.4 Limitations and Future Work within this Chapter

While the study provides robust evidence for the utility of AI-guided optimization, certain limitations inherent in the experimental validation should be acknowledged within the context of this chapter's findings. The statistical analysis for the LoopTest, despite its dramatic percentage reduction, could not be formally validated due to data discrepancies, representing a data collection anomaly. Furthermore, the limited practical effect size observed in MemoryTest and ObjectTest highlights that not all theoretically sound optimizations yield significant energy savings, emphasizing the need for empirical validation. These aspects underscore areas for further refinement in data collection protocols and deeper analysis of energy consumption drivers.

## 5.5 Chapter Summary

This chapter presented a comprehensive empirical validation of the AI-guided energy optimization methodology, building upon the design principles established in Chapter 4. It began by detailing the baseline energy consumption profiles for both workstation and lab environments, providing a crucial reference for subsequent optimization analysis.

The chapter then presented the core optimization results, showcasing energy

reductions and performance improvements for a representative home workstation function, `simulate_heavy_aggregation_query`, and for the rigorous lab test cases, LoopTest, MemoryTest, and ObjectTest. Quantifiable energy savings were demonstrated across these scenarios, with the LoopTest achieving a near-complete elimination of energy consumption and the home workstation optimization showing statistically significant improvement.

The statistical validation further confirmed the significance of optimizations for the home test and MemoryTest, while highlighting negligible effects for ObjectTest and data limitations for LoopTest. The discussion synthesized these findings, emphasizing the AI's efficacy in identifying diverse energy hotspots and the successful application of heuristic-guided optimizations across distinct computational and database-bound domains.

The chapter concluded by acknowledging key insights into the hybrid nature of AI-assisted optimization and inherent limitations in data collection and automated suggestion granularity, setting the stage for broader conclusions and future research.

# 6 Conclusion

This thesis confronts the escalating energy footprint of software systems by developing an AI-assisted optimization methodology for WordPress—powering 43.5% of global websites [13]. Through a distinct integration of dual-environment profiling (Windows workstation and Odroid H3+ lab rig), Random Forest classification, and heuristic-guided refactoring, this research achieves measurable reductions in PHP backend energy consumption. The methodology addresses critical gaps identified in the systematic literature review (Chapter 2), including the absence of WordPress-specific energy benchmarks and unvalidated AI prioritization for PHP systems. Experimental validation (Chapter 5) confirms the efficacy of this approach across database-bound and computational domains while maintaining alignment with EU sustainability directives [102].

## 6.1 Answers to Research Questions

The research questions defined in Section 1.3 are conclusively addressed below:

**RQ1:** ***Which backend PHP functions contribute most to excessive energy consumption?***

- *Answer:* Profiling identified two dominant inefficiency types:
  - *Database-bound operations:* Complex and computationally intensive aggregation queries (e.g. `simulate_heavy_aggregation_query`) with un-

optimized JOINs/GROUP BY clauses

- *Computational patterns:* Nested loops (`LoopTest`, 298,706,263 µJ baseline) and memory-intensive string operations (`MemoryTest`)

The AI classifier prioritized these using execution time, energy, and power metrics (Section 5.1)

**RQ2: *How were execution time, energy, and power captured with fidelity?***

- *Answer:* A dual-instrumentation framework ensured precision:
  - *Workstation:* Intel Power Gadget (10ms resolution) + Query Monitor for real-world PHP/SQL profiling
  - *Lab rig:* PowerGoblin (100 Hz sampling, 10 ms resolution) + Docker/Selenium for controlled energy profiling (Section 4.2.2)

  Statistical rigor included 30 iterations, outlier removal ($\pm 2\sigma$), and temporal alignment ($\pm$50ms)

**RQ3: *Did the Random Forest classifier reliably prioritize inefficiencies?***

- *Answer:* Yes. The classifier achieved high reliability by assigning optimization priorities:
  - **High:** Functions exceeding 1500 ms execution or 100J energy (e.g. `LoopTest`, `simulate_heavy_aggregation_query`)
  - **Medium/Low:** Segments below these thresholds

  Its correct prioritization guided a 75.78% energy reduction in the workstation case (Section 5.3.1).

**RQ4: *What refactoring categories were derived from AI heuristics?***

- *Answer:* Context-specific rules generated:

  - *Database optimization:* Indexing, query batching, transient caching (e.g. `get_transient()`)
  - *Algorithmic refactoring:* Hash-map substitution for nested loops (Section 5.3.2)
  - *Resource management:* Output buffering (`ob_start()`) and object reuse

  These were implemented without source code parsing (Section 4.5.2)

**RQ5:** ***Are these techniques generalizable?***

- *Answer:* Partially. The methodology succeeded within tested WordPress environments:
  - Workstation: 75.78% energy reduction in real-world functions
  - Lab rig: 99.44% reduction in `LoopTest` via algorithmic refactoring

Generalizability to diverse WordPress installations (e.g. custom plugins) or other PHP CMS (e.g. Drupal) requires further validation (Section 5.4.3).

## 6.2 Validity of Results

The reliability of this study is strengthened by several key factors, notably. The dual-environment design effectively balanced ecological validity through workstation profiling with measurement precision via the lab rig instrumentation. Statistical significance was confirmed for critical optimizations: workstation functions showed significant improvement ($p = 0.032$), while MemoryTest demonstrated negligible effects ($p = 0.445$, $d = 0.20$) and ObjectTest exhibited no practical impact ($p = 0.973$, $d = 0.01$). Furthermore, the AI classifier successfully avoided overfitting through hold-out validation (80/20 split) combined with Random Forest's inherent regularization. Several limitations warrant acknowledgment. The 99.44% energy reduction

in LoopTest could not be statistically validated due to unmatched run counts (10 unoptimized vs. 30 optimized runs, Section 5.3.4). ObjectTest's negligible reduction ($p = 0.973$, $d = 0.01$) suggests object lifecycle overhead was non-critical at tested scales (Section 5.4.1). Although thermal compensation was implemented via SmartPower 3's PAC chip[1], ambient conditions remained unlogged (Section 4.6.2). Finally, the study scope excluded front-end optimizations (CSS/JS) and non-PHP systems (mobile/IoT).

## 6.3 Contributions

This research advances sustainable software engineering through four key contributions. First, it introduces the first integrated AI pipeline for PHP energy optimization, unifying profiling, classification, heuristic refactoring, and empirical validation. Second, it provides empirical evidence by demonstrating 75.78%–99.44% energy reduction in selected WordPress backend operations, directly addressing SLR-identified gaps in PHP optimization (Section 2.4). Third, it offers reproducible artifacts, including a publicly available Dockerized test framework (Appendix B) and optimization algorithms (Appendix C), enabling validation and extension. Finally, it achieves policy alignment by introducing energy-per-transaction metrics and heuristic rules compliant with EU CSRD Article 12b, thereby bridging technical and regulatory sustainability practices.

## 6.4 Future Work

Building on Section 3.8's limitations (data requirements, computational overhead) and emerging trends (edge computing, efficient models), five research directions emerge as priorities. First, integrating LLMs for automated code-level refactoring

[1] `https://www.microchip.com/en-us/product/pac1933`

(e.g. loop unrolling, memory pooling) would enhance recommendation granularity. Second, validating cross-platform generalizability across WordPress ecosystems (WooCommerce) and PHP CMS platforms (Joomla, Drupal) should extend methodology robustness. Third, embedding containerized PowerGoblin in CI/CD pipelines (GitHub Actions/GitLab CI) would enable automated energy regression testing. Fourth, developing lightweight classifiers (EfficientNet-inspired) could minimize optimization's own carbon footprint through Green AI techniques. Finally, expanding profiling to front-end execution (JavaScript) and network layers would enable full-stack energy analysis.

## 6.5 Concluding Remarks

As the energy demands of digital infrastructure continue to accelerate, optimizing software energy consumption has become an operational imperative rather than a purely academic pursuit. This thesis demonstrates that AI-assisted methodologies can systematically reduce the carbon footprint of pervasive systems like WordPress, achieving up to 99.44% energy savings in specific, algorithmically intensive test cases. By bridging empirical instrumentation, machine learning, and actionable heuristics, this work provides a replicable blueprint for sustainable software engineering—proving that environmental responsibility and computational efficiency can coexist in the digital age. Future adoption of these practices will be pivotal in aligning software development with planetary boundaries.

# References


[1] J. Malmodin, N. Lövehagen, P. Bergmark, and D. Lundén, "Ict sector electricity consumption and greenhouse gas emissions – 2020 outcome", *Telecommunications Policy*, vol. 48, no. 3, p. 102 701, 2024, ISSN: 0308-5961. DOI: https://doi.org/10.1016/j.telpol.2023.102701.

[2] E. Gelenbe, "Electricity consumption by ict: Facts, trends, and measurements", *ACM Ubiquity*, 2023. DOI: 10.1145/3613207.

[3] World Bank and International Telecommunication Union, "Measuring national ict sector environmental impact: Arcep case study", World Bank and International Telecommunication Union, Technical Report, 2025, Accessed: 2025-04-01. [Online]. Available: https://hdl.handle.net/10986/42934.

[4] European Commission Joint Research Centre, "Ict task force study: Scope, key environmental aspects and impacts", Publications Office of the European Union, Tech. Rep., 2021, Accessed: 2025-05-07. [Online]. Available: https://susproc.jrc.ec.europa.eu/product-bureau/sites/default/files/2021-09/Task2_Scope_ICT_clean.pdf.

[5] European Commission. "Corporate sustainability reporting directive (csrd)". Accessed: 2025-02-09. [Online]. Available: https://finance.ec.europa.eu/capital-markets-union-and-financial-markets/company-reporting-and-auditing/company-reporting/corporate-sustainability-reporting_en.

[6] IFRS Foundation. "International sustainability standards board (issb)". Accessed: 2025-02-26. [Online]. Available: https://www.ifrs.org/groups/international-sustainability-standards-board/.

[7] A. Lohikoski, "Green software development – challenges and implementation approaches", M.S. thesis, Aalto University, 2024.

[8] L. Pathak and K. Kher, "Green software engineering: A comprehensive study", *International Journal of Innovative Science and Research Technology (IJISRT)*, vol. 9, no. 2, Feb. 2024. DOI: 10.5281/zenodo.10686147.

[9] C. C. Venters et al., "Sustainable software engineering: Reflections on advances in research and practice", *Information and Software Technology*, vol. 164, p. 107 316, 2023, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2023.107316.

[10] M. Bamiduro, "Sustainable software development lifecycle: A view on practices, stakeholders and organizational barriers", M.S. thesis, LUT University, 2024.

[11] Y. Arushanyan, "Environmental impacts of ict: Present and future", Ph.D. dissertation, KTH Royal Institute of Technology, 2016.

[12] N. Wirth, "A plea for lean software", *IEEE Computer*, vol. 28, no. 2, pp. 64–68, 1995. DOI: 10.1109/2.348001.

[13] WPZOOM, *How many websites use wordpress? june 2025 statistics*, Accessed June 13, 2025, 2025. [Online]. Available: https://www.wpzoom.com/blog/wordpress-statistics/.

[14] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – a systematic literature review", *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, 2009. DOI: https://doi.org/10.1016/j.infsof.2008.09.009.

[15] G. Gartlehner et al., "Single-reviewer abstract screening missed 13 percent of relevant studies: A crowd-based, randomized controlled trial", *Journal of Clinical Epidemiology*, vol. 121, pp. 20–28, 2020. DOI: 10.1016/j.jclinepi.2019.12.001.

[16] S. Nurmivaara, "Green in software engineering: A systematic literature review", M.S. thesis, University of Helsinki, 2023.

[17] A. Atadoga, U. J. Umoga, O. A. Lottu, and E. O. Sodiya, "Advancing green computing: Practices, strategies, and impact in modern software development for environmental sustainability", *World Journal of Advanced Engineering Technology and Sciences*, vol. 11, no. 1, pp. 220–230, 2024. DOI: 10.30574/wjaets.2024.11.1.0052.

[18] L. Cruz, J. P. Fernandes, M. H. Kirkeby, S. Martínez-Fernández, J. Sallou, et al., *Greening ai-enabled systems with software engineering: A research agenda for environmentally sustainable ai practices*, 2025. DOI: 10.48550/arXiv.2506.01774.

[19] L. Pathak and K. Kher, "Green software engineering: A comprehensive study", *International Journal of Innovative Science and Research Technology*, vol. 9, 2024. DOI: 10.5281/zenodo.10686147.

[20] European Commission, *Ict environmental impact (rp2024)*, Accessed: 2025-07-09, 2024. [Online]. Available: https://interoperable-europe.ec.europa.eu/collection/rolling-plan-ict-standardisation/ict-environmental-impact-rp2024.

[21] C. Freitag, M. Berners-Lee, K. Widdicks, B. Knowles, G. Blair, and A. Friday, "The climate impact of ict: A review of estimates, trends and regulations", *arXiv preprint arXiv:2102.02622*, 2021.

[22] P. Fors, D. Kreps, and A. O'Brien, "Green it: The evolution of environmental concerns within ict policy, research and practice", *Digital Sustainability: Leveraging Digital Technology to Combat Climate Change*, pp. 25–48, 2024. DOI: 10.1007/978-3-031-61749-2_2.

[23] BirchLogic, *The green it association's 2025 global outlook*, Accessed: 2025-05-06, 2025. [Online]. Available: https://birchlogic.substack.com/p/the-green-it-associations-2025-global.

[24] Interoperable Europe, *Ict environmental impact*, Accessed: 2025-04-22, 2024. [Online]. Available: https://interoperable-europe.ec.europa.eu/collection/rolling-plan-ict-standardisation/ict-environmental-impact-0.

[25] B. Lutkevich, *What is green it (green information technology) and why is it important?*, Accessed: 2025-05-19, 2024. [Online]. Available: https://www.techtarget.com/searchcio/definition/green-IT-green-information-technology.

[26] L. M. Hilty and B. Aebischer, "ICT for sustainability: An emerging research field", in *ICT Innovations for Sustainability*, L. M. Hilty and B. Aebischer, Eds., Springer, 2015, pp. 3–36. DOI: 10.1007/978-3-319-09228-7_1.

[27] A. S. Andrae, "New perspectives on internet electricity use in 2030", *Engineering and Applied Science Letters*, vol. 3, no. 2, pp. 19–31, Jun. 2020. DOI: 10.30538/psrp-easl2020.0038.

[28] Enerdata, *Between 10% and 20% of electricity consumption from the ICT sector by 2030*, Enerdata Executive Briefing, Accessed: 2025-06-11, 2024. [Online]. Available: https://www.enerdata.net/publications/executive-briefing/between-10-and-20-electricity-consumption-ict-sector-2030.html.

[29] Interoperable Europe, *ICT environmental impact (RP2023)*, Interoperable Europe Portal, Accessed: 2025-01-28. [Online]. Available: https://interoperable-europe.ec.europa.eu/collection/rolling-plan-ict-standardisation/ict-environmental-impact-rp2023.

[30] ASML, *EUV lithography systems*, ASML Products, Accessed: 2025-03-15, 2024. [Online]. Available: https://www.asml.com/en/products/euv-lithography-systems.

[31] KES Systems, *Latest developments in semiconductor technology*, KES Systems Resources, Accessed: 2025-02-18, 2024. [Online]. Available: https://www.kessystemsinc.com/resources/latest-developments-in-semiconductor-technology/.

[32] E. T. S. I. (ETSI), *Pre-defined collections*, ETSI Standards, Accessed: 2025-02-11, 2024. [Online]. Available: https://www.etsi.org/standards#Pre-defined%20Collections.

[33] International Energy Agency (IEA), *Data Centres and Data Transmission Networks*, Accessed: 2025-04-09, Jul. 2023. [Online]. Available: https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks.

[34] Climate Neutral Data Centre Pact, *Climate Neutral Data Centre Pact – The Green Deal need Green Infrastructure*, Accessed: 2025-02-07. [Online]. Available: https://www.climateneutraldatacentre.net/.

[35] European Commission, *Circular economy action plan*, European Commission - Environment, Accessed: 2025-05-30, 2020. [Online]. Available: https://environment.ec.europa.eu/strategy/circular-economy-action-plan_en.

[36] M. Compagnoni, "Is extended producer responsibility living up to expectations? a systematic literature review focusing on electronic waste", *Journal*

*of Cleaner Production*, vol. 365, p. 132 709, 2022. DOI: 10.1016/j.jclepro.2022.132709.

[37] R. Evans and J. Gao, *DeepMind AI Reduces Google Data Centre Cooling Bill by 40%*, Google DeepMind Blog, Accessed: 2025-01-06, Jul. 2016. [Online]. Available: https://deepmind.google/discover/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-by-40/.

[38] E. T. S. I. (ETSI), *ETSI standards*, ETSI Standards Portal, Accessed: 2025-06-25, 2024. [Online]. Available: https://www.etsi.org/standards#page=1&search=&title=1&etsiNumber=1&content=1&version=0&onApproval=1&published=1&withdrawn=1&historical=1&isCurrent=1&superseded=1&startDate=1988-01-15&endDate=2025-02-18&harmonized=0&keyword=&TB=&stdType=&frequency=&mandate=&collection=&sort=1.

[39] European Commission, *Commission Regulation (EU) 2019/424 of 15 March 2019 laying down ecodesign requirements for servers and data storage products*, Official Journal of the European Union, Accessed: 2025-03-08, Mar. 2019. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2019/424/oj.

[40] O. Danushi, S. Forti, and J. Soldani, "Environmentally sustainable software design and development: A systematic literature review", *arXiv preprint arXiv:2407.19901*, 2024.

[41] Green Software Foundation Community, *10 Recommendations for Green Software Development*, Green Software Foundation Articles, Accessed: 2025-02-19. [Online]. Available: https://greensoftware.foundation/articles/10-recommendations-for-green-software-development.

[42] S. Naumann, M. Hirsch-Dick, E. Kern, and T. Johann, "The greensoft model: A reference model for green and sustainable software and its engineering",

*Sustainable Computing: Informatics and Systems*, vol. 1, no. 4, pp. 294–304, 2011. DOI: 10.1016/j.suscom.2011.06.004.

[43] F. Jammal, N. Aljabri, M. Al-Hashimi, M. Saleh, and O. Abulnaja, "A preliminary empirical study of the power efficiency of matrix multiplication", *Electronics*, vol. 12, no. 7, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12071599.

[44] R. Manimegalai, S. Sandhanam, A. S. Nandhini, and P. Pandia, "Energy efficient coding practices for sustainable software development", in *Proceedings of the International Conference on Sustainable Energy Technologies, Power and Smart Devices (ICSETPSD 2023)*, EAI, 2023, pp. 1–10. DOI: 10.4108/eai.17-11-2023.2342635.

[45] J. Balanza-Martinez, P. Lago, and R. Verdecchia, "Tactics for software energy efficiency: A review", in *Advances and New Trends in Environmental Informatics 2023: Sustainable Digital Society [Proceedings]*, ser. Progress in IS, V. Wohlgemuth, D. Kranzlmüller, and M. Höb, Eds., Springer, 2024, pp. 115–140. DOI: 10.1007/978-3-031-46902-2_7.

[46] J. A. Ansere, E. K. N. Yeboah, A. O. Fapojuwo, and S. A. Olatunji, "Dynamic resource optimization for energy-efficient 6g-iot networks", *Sensors*, vol. 23, no. 10, p. 4567, 2023. DOI: 10.3390/s23104567.

[47] W. Lang and J. Patel, "Towards eco-friendly database management systems", *arXiv preprint arXiv:0909.1767*, 2009.

[48] S. Attarha and A. Förster, "Empowering iot applications with flexible, energy-efficient remote management of low-power edge devices", in *Proceedings of the 2023 International Conference on Embedded Wireless Systems and Networks*, 2024. DOI: 10.36227/techrxiv.171441115.55562122/v1.

[49] S. Kumar and N. Kumar, "Compiler design and optimization techniques for software efficiency and performance", in *E3S Web of Conferences*, vol. 470, 2023, p. 04 047. DOI: 10.1051/e3sconf/202347004047.

[50] A. H. Ashouri, W. Killian, J. Cavazos, G. Palermo, and C. Silvano, "A survey on compiler autotuning using machine learning", *ACM Computing Surveys*, vol. 51, no. 5, Sep. 2018. DOI: 10.1145/3197978.

[51] J. Aguilar, A. Garces-Jimenez, M. R-Moreno, and R. García, "A systematic literature review on the use of artificial intelligence in energy self-management in smart buildings", *Renewable and Sustainable Energy Reviews*, vol. 151, p. 111 530, 2021, ISSN: 1364-0321. DOI: 10.1016/j.rser.2021.111530.

[52] R. O. Yussuf and O. S. Asfour, "Applications of artificial intelligence for energy efficiency throughout the building lifecycle: An overview", *Energy and Buildings*, vol. 305, p. 113 903, 2024. DOI: 10.1016/j.enbuild.2024.113903.

[53] Y. Zhang, Y. Ren, Z. Yang, and Y. Jiang, "Application-oriented cloud workload prediction: A survey and new perspectives", *Tsinghua Science and Technology*, vol. 29, no. 2, pp. 303–323, 2024. DOI: 10.26599/TST.2024.9010024.

[54] H. Janardhanan, "Ai-driven load balancing for energy-efficient data centers", *International Journal of Computer Trends and Technology*, vol. 72, no. 8, pp. 13–18, 2024. DOI: 10.14445/22312803/IJCTT-V72I8P103.

[55] A. Button, *7 ways AI can support energy conservation efforts*, Earth.Org, Accessed: 2025-04-16, 2023. [Online]. Available: https://earth.org/7-ways-ai-can-support-energy-conservation-efforts/.

[56] J. Kaur, S. S. Sehra, D. Singh, and P. K. Mallick, "Advancements in cache management: A review of machine learning innovations for enhanced performance and security", *Frontiers in Artificial Intelligence*, vol. 8, p. 1 441 250, 2025. DOI: 10.3389/frai.2025.1441250.

[57] S. K. Moghaddam, R. Buyya, and K. Ramamohanarao, "ACAS: An anomaly-based cause aware auto-scaling framework for clouds", *Journal of Parallel and Distributed Computing*, vol. 126, pp. 107–120, 2019. DOI: https://doi.org/10.1016/j.jpdc.2018.12.002.

[58] P. Christen and R. Schnell, "When data science goes wrong: How misconceptions about data capture and processing causes wrong conclusions", *Harvard Data Science Review*, vol. 6, no. 1, 2024. DOI: 10.1162/99608f92.34f8e75b.

[59] A. Kapoor, "BIG DATA INFRASTRUCTURE: INTEGRATING LEGACY SYSTEMS WITH AI-DRIVEN PLATFORMS", in *Computer Science & Information Technology (CS & IT)*, 2024, pp. 145–152. DOI: 10.5121/csit.2024.141913.

[60] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 09, pp. 13 693–13 696, Apr. 2020. DOI: 10.1609/aaai.v34i09.7123.

[61] T. Holmes, C. McLarty, Y. Shi, P. Bobbie, and K. Suo, "Energy efficiency on edge computing: Challenges and vision", in *Proceedings of the IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2022, pp. 1–6. DOI: 10.1109/IPCCC55074.2022.9894303.

[62] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks", in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 6105–6114. DOI: 10.48550/arXiv.1905.11946.

[63] M. Judge, M. Usman, S. Hussain, M. Khan, M. Ahmad, and M. Khan, "A comprehensive review of artificial intelligence applications for energy manage-

ment systems in renewable energy-based microgrids", *Energy and AI*, vol. 18, p. 100 322, 2024. DOI: 10.1016/j.egyai.2024.100322.

[64] N. Abbas and M. H. Kirkeby, "Energy-efficiency in wordpress development: A teaching approach using code proxies", in *Proceedings of the 2025 IEEE/ACM 37th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, IEEE/ACM, 2025, Poster 46.

[65] P. Pathania et al., "Calculating software's energy use and carbon emissions: A survey of the state of art, challenges, and the way ahead", in *Proceedings of the 9th International Workshop on Green and Sustainable Software (GREENS '25), co-located with ICSE 2025*, Ottawa, Canada, 2025. DOI: 10.1109/GREENS66463.2025.00018.

[66] L. Khrouf, A. Shatnawi, B. T. Niang, and B. Verhaeghe, "On the energy consumption of web applications: An empirical study of their design solutions", in *2025 IEEE/ACM 9th International Workshop on Green and Sustainable Software (GREENS)*, hal-05127336v1, IEEE Computer Society, 2024. DOI: 10.1109/GREENS66463.2025.00012.

[67] Webslice, *Xdebug - how to profile the performance of php websites*, Accessed July 2025, 2024. [Online]. Available: https://webslice.com/blog/xdebug-php-profiling.

[68] C. P. P. Authors, *Code profiler – wordpress performance profiling and debugging made easy*, WordPress Plugin Directory, Accessed July 2025, 2025. [Online]. Available: https://wordpress.org/plugins/code-profiler/.

[69] Stack Exchange Community, *Performance impact of using functions in wordpress?*, WordPress Stack Exchange, Accessed July 2025, 2016. [Online]. Available: https://wordpress.stackexchange.com/questions/253991/performance-impact-of-using-functions-in-wordpress.

[70] Z. Wang and M. O'Boyle, "Machine learning in compiler optimization", *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1879–1901, 2018. DOI: 10.1109/JPROC.2018.2817118.

[71] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies", *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 374–398, 2003. DOI: 10.1145/954339.954341.

[72] A. Ghosh, J. Parikh, V. S. Sengar, and J. R. Haritsa, "Plan selection based on query clustering", in *Proceedings of the 28th international conference on Very Large Data Bases*, 2002, pp. 179–190. DOI: https://doi.org/10.1016/B978-155860869-6/50024-X.

[73] P. Srivastava, B. Gohil, and D. Patel, "Load management model for cloud computing using cloudsim", *International Journal of Computer Theory and Engineering*, vol. 9, no. 5, pp. 390–393, 2017. DOI: 10.7763/IJCTE.2017.V9.1172.

[74] S. Schubert, D. Kostic, W. Zwaenepoel, and K. G. Shin, "Profiling software for energy consumption", in *2012 IEEE International Conference on Green Computing and Communications*, IEEE, 2012, pp. 515–522. DOI: 10.1109/GreenCom.2012.86.

[75] S. Basu, A. Sundarrajan, J. Ghaderi, S. Shakkottai, and R. Sitaraman, "Adaptive ttl-based caching for content delivery", in *Proceedings of the 2017 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '17)*, ser. SIGMETRICS '17, Urbana-Champaign, IL, USA, Jun. 2017, pp. 1–2. DOI: 10.1145/3078505.3078560.

[76] D. Hirsch and S. Madria, "A resource-efficient adaptive caching scheme for mobile ad-hoc networks", in *2010 29th IEEE International Symposium on*

*Reliable Distributed Systems*, IEEE, 2010, pp. 64–71. DOI: 10.1109/SRDS.2010.16.

[77] W. Wang, P. Mishra, and S. Ranka, "Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems", in *Proceedings of the 48th Design Automation Conference (DAC)*, 2011, pp. 948–953. DOI: 10.1145/2024724.2024935.

[78] K. T. Sundararajan, V. Porpodas, T. M. Jones, N. P. Topham, and B. Franke, "Energy-efficient cache partitioning for high-performance cmps", in *Proceedings of the 18th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2012, pp. 308–319. DOI: 10.1109/HPCA.2012.6168942.

[79] G. Chen, K. Huang, J. Huang, and A. Knoll, "Cache partitioning and scheduling for energy optimization of real-time mpsocs", in *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 1170–1175. DOI: 10.3850/9783981537079_0973.

[80] K. Ali, M. Aboelaze, and S. Datta, "Predictive line buffer: A fast, energy-efficient cache architecture", in *Proceedings of the 2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON)*, 2006, pp. 776–783. DOI: 10.1109/second.2006.1629366.

[81] X. Wang and V. Friderikos, "Energy efficient proactive caching with multipath routing in content delivery networks", *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 1, pp. 66–76, 2018. DOI: 10.1109/TGCN.2017.2776238.

[82] A. Alourani, M. Sardaraz, M. Tahir, and M. S. Khan, "Dynamic and energy efficient cache scheduling framework for iomt over icn", *Applied Sciences*, vol. 13, no. 21, p. 11 840, 2023. DOI: 10.3390/app132111840.

[83] E. Azhir, N. J. Navimipour, M. Hosseinzadeh, A. Sharifi, and A. Darwesh, "An automatic clustering technique for query plan recommendation", *Information Sciences*, vol. 545, pp. 620–632, 2021. DOI: https://doi.org/10.1016/j.ins.2020.09.037.

[84] W. Kong, Y. Wang, H. Dai, L. Zhao, and C. Wang, "Analysis of energy consumption structure based on k-means clustering algorithm", in *E3S Web of Conferences, Proceedings of the 2021 International Conference on Energy Science and Chemical Engineering (ICESCE 2021)*, EDP Sciences, vol. 267, 2021, p. 01 054. DOI: 10.1051/e3sconf/202126701054.

[85] M. Abdolrazzagh-Nezhad and M. Kherad, "An energy-efficient and dynamic clustering approach for wireless sensor networks", *Research Square*, 2023. DOI: 10.21203/rs.3.rs-3019670/v1.

[86] H. You, S. L. Song, D. J. Kerbyson, and W.-c. Feng, "Powerspector: Fine-grained power profiling for high-performance computing", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2015, pp. 1–12. DOI: 10.1145/2807591.2807610.

[87] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design", in *Proceedings of the 2008 International Conference on Embedded Software (EMSOFT)*, 2008, pp. 26–31. DOI: 10.1145/1450058.1450064.

[88] P. Li, H. Wang, G. Tian, and Z. Fan, "Towards sustainable cloud computing: Load balancing with nature-inspired meta-heuristic algorithms", *Electronics*, vol. 13, no. 13, p. 2578, 2024. DOI: 10.3390/electronics13132578.

[89] M.-C. Li, A. Ghosh, and S. Sen, "Approximate jpeg compression hardware for energy-constrained image sensors", *arXiv preprint arXiv:2406.16358*, 2024.

[90] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing", in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10)*, Boston, MA, USA: USENIX Association, 2010, pp. 1–7.

[91] Android Developers. "Background optimization". Accessed: 2025-06-04. [Online]. Available: https://developer.android.com/topic/performance/background-optimization.

[92] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation", in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '11)*, San Jose, CA, USA: ACM, 2011, pp. 164–174. DOI: 10.1145/1993498.1993518.

[93] National Renewable Energy Laboratory. "Standard energy efficiency data (seed) platform documentation, version 2.20.1". Accessed: 2025-06-29. [Online]. Available: https://seed-platform.org/code_documentation/2.20.1/.

[94] P. Nuggehalli, V. Srinivasan, and C.-F. Chiasserini, "Energy-efficient caching strategies in ad hoc wireless networks", in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, 2003, pp. 25–34. DOI: 10.1145/778415.778419.

[95] N. K. Jayakodi, S. Belakaria, A. Deshwal, and J. R. Doppa, "Design and optimization of energy-accuracy tradeoff networks for mobile platforms via pretrained deep models", *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 19, no. 1, pp. 1–24, 2020. DOI: 10.1145/3366636.

[96] Z. Xu et al., "Energy-efficient machine learning model selection: A comprehensive empirical study", in *Proceedings of the 46th International Conference*

*on Software Engineering (ICSE)*, IEEE, 2024, pp. 1281–1293. DOI: 10.1109/ICSE54507.2024.10550120.

[97] N. C. Ohalete, A. O. Aderibigbe, E. C. Ani, P. E. Ohenhen, and A. E. Akinoso, "Data science in energy consumption analysis: A review of ai techniques in identifying patterns and efficiency opportunities", *Engineering Science & Technology Journal*, vol. 4, no. 6, pp. 357–380, 2023. DOI: https://doi.org/10.51594/estj.v4i6.637.

[98] D. Bán, R. Ferenc, I. Siket, Á. Kiss, and T. Gyimóthy, "Prediction models for performance, power, and energy efficiency of software executed on heterogeneous hardware", *Journal of Supercomputing*, vol. 75, no. 8, pp. 4001–4025, 2019. DOI: 10.1007/s11227-018-2252-6.

[99] P. Nie, M. Roccotelli, M. P. Fanti, Z. Ming, and Z. Li, "Prediction of home energy consumption based on gradient boosting regression tree", *Energy Reports*, vol. 7, pp. 1246–1255, 2021, ISSN: 2352-4847. DOI: https://doi.org/10.1016/j.egyr.2021.02.006.

[100] European Commission. "European green deal". Accessed: 2025-02-26. [Online]. Available: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_en.

[101] University of Turku, Faculty of Technology, *Powergoblin user manual: Performing measurements*, User Manual, Accessed: 2025-06-17, 2025. [Online]. Available: https://tech.utugit.fi/soft/tools/power/doc/manual/usage/index.html.

[102] European Parliament and the Council, *Directive (EU) 2022/2464 of 14 December 2022 amending Regulation (EU) No 537/2014, Directive 2004/109/EC, Directive 2006/43/EC and Directive 2013/34/EU, as regards corporate sustainability reporting*, Official Journal of the European Union, L 322, 16.12.2022,

p. 15–80, Accessed: 2025-01-05. [Online]. Available: https://eur-lex.europa.eu/eli/dir/2022/2464/oj.

[103] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges", *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016. DOI: 10.1109/JIOT.2016.2579198.

[104] Y. Li, Y. Wang, X. Wang, and V. Friderikos, "Energy-efficient proactive caching with multipath routing in content delivery networks", *IEEE Transactions on Green Communications and Networking*, vol. 6, no. 3, pp. 1418–1428, 2022. DOI: 10.1109/TGCN.2022.3180339.

[105] R. Pereira et al., "Energy efficiency across programming languages: How do energy, time, and memory relate?", in *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (SLE 2017)*, New York, NY, USA: ACM, 2017, pp. 256–267. DOI: 10.1145/3136014.3136031.

[106] G. Pinto, I. Moura, F. Ebert, and F. Castor, "Mining energy-aware commits", in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, IEEE, 2015, pp. 56–67. DOI: 10.1109/MSR.2015.13.

[107] SUSO Academy, *Green Coding: The 5 Most Important Basics for Sustainable Software Development with Code Examples*, SUSO Academy Blog, Accessed: 2025-05-02. [Online]. Available: https://www.suso.academy/en/2023/03/13/green-coding-the-5-most-important-basics-for-sustainable-software-development-with-code-examples/.

[108] IBM, *What is green coding and why does it matter?*, Accessed: 2025-03-21, 2023. [Online]. Available: https://www.ibm.com/think/topics/green-coding.

[109] X. Li and J. P. Gallagher, "A source-level energy optimization framework for mobile applications", in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, IEEE, 2016, pp. 31–40. DOI: 10.1109/SCAM.2016.12.

[110] V. Jain, *The role of ai in forecasting and where it falls short*, Accessed: 2025-03-21, 2023. [Online]. Available: https://www.afponline.org/training-resources/resources/articles/Details/the-role-of-ai-in-forecasting-and-where-it-falls-short.

[111] T. Hartung, A. Maertens, T. Luechtefeld, A. Kleensang, and K. Tsaioun, "Challenges and opportunities for validation of ai-based new approach methods", *ALTEX - Alternatives to animal experimentation*, vol. 42, no. 1, pp. 3–21, 2025. DOI: 10.14573/altex.2412291.

[112] S. Mohammed, "The effects of data quality on machine learning model performance: A comprehensive empirical study", *International Journal of Information Management Data Insights*, vol. 5, no. 2, p. 100 234, 2025. DOI: https://doi.org/10.1016/j.is.2025.102549.

[113] D. Hussein, L. Nelson, and G. Bhat, "Sensor-aware classifiers for energy-efficient time series applications on iot devices", *arXiv preprint arXiv:2407.08715*, 2024.

[114] R. Verdecchia, J. Sallou, and L. Cruz, "A systematic review of green ai", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 13, no. 4, e1507, 2023. DOI: 10.1002/widm.1507.

[115] K. DePalma, I. Miminoshvili, C. Henselder, K. Moss, and E. A. AlOmar, "Exploring chatgpt's code refactoring capabilities: An empirical study", *Expert Systems with Applications*, vol. 249, p. 123 602, 2024. DOI: 10.1016/j.eswa.2024.123602.

# Appendix A Categorized SLR Studies

Table A.1: Distribution of SLR Studies by Category

| **Category** | **Count** | **Percentage** | **Section** |
|---|---|---|---|
| Hardware & Data Center | 47 | 53.4% | A.1 |
| Software-Level Efficiency | 10 | 11.4% | A.2 |
| Standardized Metrics | 6 | 6.8% | A.3 |
| AI-Driven Optimization | 25 | 28.4% | A.4 |
| **Total** | 88 | 100% | |

## A.1 Hardware & Data Center (47 studies)

1. [1] - ICT sector electricity consumption and greenhouse gas emissions – 2020 outcome
2. [2] - Electricity consumption by ICT: Facts, trends, and measurements
3. [3] - Measuring National ICT Sector Environmental Impact: Arcep Case Study
4. [4] - ICT Task Force Study: Scope, Key Environmental Aspects and Impacts
5. [36] - Is Extended Producer Responsibility Living Up to Expectations? A Systematic Literature Review Focusing on Electronic Waste

6. [32] - Pre-defined Collections
7. [33] - Data Centres and Data Transmission Networks
8. [34] - Climate Neutral Data Centre Pact
9. [35] - Circular Economy Action Plan
10. [37] - DeepMind AI Reduces Google Data Centre Cooling Bill by 40%
11. [38] - ETSI Standards
12. [39] - Commission Regulation (EU) 2019/424 laying down ecodesign requirements
13. [30] - EUV Lithography Systems
14. [31] - Latest Developments in Semiconductor Technology
15. [20] - ICT Environmental Impact (RP2024)
16. [24] - ICT Environmental Impact
17. [29] - ICT Environmental Impact (RP2023)
18. [28] - Between 10 and 20% of electricity consumption from the ICT sector by 2030
19. [27] - New Perspectives on Internet Electricity Use in 2030
20. [103] - Edge Computing: Vision and Challenges
21. [77] - Dynamic Cache Reconfiguration and Partitioning for Energy Optimization
22. [78] - Energy-Efficient Cache Partitioning for High-Performance CMPs
23. [79] - Cache Partitioning and Scheduling for Energy Optimization

24. [80] - Predictive Line Buffer: A Fast, Energy-Efficient Cache Architecture
25. [81] - Energy Efficient Proactive Caching with Multipath Routing
26. [84] - Analysis of Energy Consumption Structure Based on K-means Clustering Algorithm
27. [85] - An Energy-Efficient and Dynamic Clustering Approach for Wireless Sensor Networks
28. [104] - Energy-Efficient Proactive Caching with Multipath Routing in Content Delivery Networks
29. [86] - PowerSpector: Fine-Grained Power Profiling for High-Performance Computing
30. [87] - Fine-Grained Energy Profiling for Power-Aware Application Design
31. [88] - Towards Sustainable Cloud Computing: Load Balancing with Nature-Inspired Meta-Heuristic Algorithms
32. [90] - Energy Efficiency of Mobile Clients in Cloud Computing
33. [94] - Energy-Efficient Caching Strategies in Ad Hoc Wireless Networks
34. [89] - Approximate JPEG Compression Hardware for Energy-Constrained Image Sensors
35. [91] - Background Optimization
36. [92] - EnerJ: Approximate Data Types for Safe and General Low-Power Computation
37. [93] - Standard Energy Efficiency Data (SEED) Platform Documentation

38. [99] - Prediction of Home Energy Consumption Based on Gradient Boosting Regression Tree
39. [82] - Dynamic and Energy Efficient Cache Scheduling Framework for IoMT over ICN
40. [105] - Energy Efficiency Across Programming Languages
41. [106] - Mining Energy-Aware Commits
42. [50] - A Survey on Compiler Autotuning using Machine Learning
43. [14] - Systematic Literature Reviews in Software Engineering
44. [21] - The Climate Impact of ICT: A Review of Estimates, Trends and Regulations
45. [22] - Green IT: The Evolution of Environmental Concerns Within ICT Policy, Research and Practice
46. [23] - The Green IT Association's 2025 Global Outlook
47. [25] - What is Green IT and Why is it Important?

## A.2 Software-Level Efficiency (10 studies)

1. [40] - Environmentally Sustainable Software Design and Development: A Systematic Literature Review
2. [41] - 10 Recommendations for Green Software Development
3. [42] - The GREENSOFT Model: A Reference Model for Green and Sustainable Software
4. [107] - Green Coding: The 5 Most Important Basics for Sustainable Software Development

5. [47] - Towards Eco-Friendly Database Management Systems
6. [108] - What is Green Coding and Why Does it Matter?
7. [109] - A Source-Level Energy Optimization Framework for Mobile Applications
8. [12] - A Plea for Lean Software
9. [7] - Green Software Development - Challenges and Implementation Approaches
10. [8] - Green Software Engineering: A Comprehensive Study

## A.3 Standardized Metrics (6 studies)

1. [74] - Profiling Software for Energy Consumption
2. [86] - PowerSpector: Fine-Grained Power Profiling for High-Performance Computing
3. [87] - Fine-Grained Energy Profiling for Power-Aware Application Design
4. [93] - Standard Energy Efficiency Data (SEED) Platform Documentation
5. [42] - The GREENSOFT Model: A Reference Model for Green and Sustainable Software
6. [101] - PowerGoblin User Manual: Performing Measurements

## A.4 AI-Driven Optimization (25 studies)

1. [51] - A Systematic Literature Review on AI in Energy Self-Management in Smart Buildings
2. [54] - AI-Driven Load Balancing for Energy-Efficient Data Centers

3. [63] - A Comprehensive Review of AI Applications for Energy Management Systems
4. [96] - Energy-Efficient Machine Learning Model Selection: A Comprehensive Empirical Study
5. [110] - The Role of AI in Forecasting and Where It Falls Short
6. [55] - 7 Ways AI Can Support Energy Conservation Efforts
7. [111] - Challenges and Opportunities for Validation of AI-Based New Approach Methods
8. [112] - The Effects of Data Quality on Machine Learning Model Performance
9. [60] - Energy and Policy Considerations for Modern Deep Learning Research
10. [62] - EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks
11. [64] - Energy-Efficiency in WordPress Development: A Teaching Approach Using Code Proxies
12. [95] - Design and Optimization of Energy-Accuracy Tradeoff Networks
13. [65] - Calculating Software's Energy Use and Carbon Emissions
14. [72] - Plan Selection Based on Query Clustering
15. [73] - Load Management Model for Cloud Computing Using CloudSim
16. [85] - An Energy-Efficient and Dynamic Clustering Approach for Wireless Sensor Networks
17. [88] - Towards Sustainable Cloud Computing: Load Balancing with Nature-Inspired Algorithms

18. [99] - Prediction of Home Energy Consumption Based on Gradient Boosting Regression Tree

19. [82] - Dynamic and Energy Efficient Cache Scheduling Framework

20. [113] - Sensor-Aware Classifiers for Energy-Efficient Time Series Applications

21. [92] - EnerJ: Approximate Data Types for Safe and General Low-Power Computation

22. [18] - Greening AI-Enabled Systems with Software Engineering

23. [114] - A Systematic Review of Green AI

24. [17] - Advancing Green Computing: Practices and Impact in Modern Software Development

25. [115] - Exploring ChatGPT's Code Refactoring Capabilities: An Empirical Study

# Appendix B Lab Test Automation Framework

## Reproducible Environment Setup

```
# Clone version-controlled environment
git clone -b thesis-v1.3 -depth 1  https://gitlab.utu.fi/tech/soft/visiiri/
benchmarks/mdkhan-wordpress-energy-docker.git


# Build and launch containers
docker-compose -f docker-compose.yml --profile full up -d


# Initialize test data
docker exec wordpress \
  wp post generate --count=5000 --allow-root
docker exec wordpress \
  wp comment generate --count=10000 --post_id=1 --allow-root
```

## Core Test Automation Logic

```
# test_controller.py
```

```
import requests
from selenium import webdriver

POWERGOBLIN_URL = "http://192.168.1.10:8080/api/v2"
WORDPRESS_URL = "http://odroid:8080"

def pg_command(endpoint):
    requests.get(f"{POWERGOBLIN_URL}/{endpoint}")

def execute_test_case(test_name, wp_param):
    # Initialize session
    pg_command("startSession")
    pg_command(f"session/latest/rename/{test_name}")

    # Setup measurement
    pg_command("session/latest/measurement/start")

    # Execute runs
    driver = webdriver.Chrome()
    for run in range(30):
        pg_command("session/latest/run/start")
        driver.get(f"{WORDPRESS_URL}/?{wp_param}")
        pg_command("session/latest/run/stop")
        time.sleep(1.0)  # Stabilization period

    # Finalize
    driver.quit()
```

```
    pg_command("session/latest/measurement/stop")
    pg_command("session/latest/close")
```

# Energy Optimization Implementations

## Algorithmic Optimization (Loop Pattern)

```
// Unoptimized: O(n³) complexity
function run_loop_test_unoptimized() {
    $comments = get_comments(['number' => 10000]);
    $duplicates = [];
    foreach ($comments as $i => $c1) {
        foreach ($comments as $j => $c2) {
            foreach ($comments as $k => $c3) {
                if ($i != $j && $j != $k) {
                    // Comparison logic
                }
            }
        }
    }
}

// Optimized: O(n) hash-map approach
function run_loop_test_optimized() {
    $comments = get_comments(['number' => 10000]);
    $content_map = [];
    foreach ($comments as $comment) {
        $key = strtoupper($comment->comment_content);
```

```
        $content_map[$key][] = $comment->comment_ID;
    }
    foreach ($content_map as $entries) {
        if (count($entries) >= 3) {
            // Processing logic
        }
    }
}
```

### Memory Management Optimization

```
// Baseline: Inefficient string concatenation
function memory_test_unoptimized() {
    $posts = get_posts(['numberposts' => 5000]);
    $xml = '';
    foreach ($posts as $post) {
        $xml .= '<item>' . esc_html($post->post_title) . '</item>';
    }
    return $xml;
}

// Optimized: Output buffering
function memory_test_optimized() {
    $posts = get_posts(['numberposts' => 5000]);
    ob_start();
    foreach ($posts as $post) {
        echo '<item>' . esc_html($post->post_title) . '</item>';
    }
```

```
    return ob_get_clean();
}
```

**Complete Artifacts:**

`https://gitlab.utu.fi/tech/soft/visiiri/benchmarks/mdkhan-thesis-ai-framework`

# Appendix C Shared Optimization Algorithms

## Core Implementation Logic

### Priority Classification System

The AI-driven priority assignment

```
def assign_priority(row):
    # Priority 1: Critical inefficiencies
    if (row['Execution Time (ms)'] > 1500
        or row['Processor Energy (Joules)'] > 100
        or row['Average Processor Power (Watt)'] > 12):
        return 1  # High priority

    # Priority 2: Moderate inefficiencies
    elif (800 <= row['Execution Time (ms)'] <= 1500
          or 60 <= row['Processor Energy (Joules)'] <= 100
          or 8 <= row['Average Processor Power (Watt)'] <= 12):
        return 2  # Medium priority

    # Priority 3: Optimized operations
```

```
    return 3  # Low priority


# Feature engineering
data['Energy CoV'] = (data['Processor Energy (Joules)'].std()
                      / data['Processor Energy (Joules)'].mean())
if 'Unoptimized Time' in data.columns:
    data['Rel Speedup'] = (data['Unoptimized Time']
                           - data['Optimized Time'])


# Standardization and model training
scaler = StandardScaler()
X_scaled = scaler.fit_transform(data[features])
model = RandomForestClassifier(n_estimators=100,
                               random_state=42)
model.fit(X_train, y_train)
```

## Environment-Aware Optimization Engine

Enhanced heuristic rules

```
def generate_suggestions(row):
    suggestions = []


    # ===== LAB ENVIRONMENT RULES =====
    if row['Environment'] == 'Lab':
        # Loop patterns
        if ('loop' in row['Function Name'].lower()
            and row['Execution Time (ms)'] > 1000):
            suggestions.append("Replace nested loops with "
```

```
                            "hash-map logic")

    # Memory operations
    if 'memory' in row['Function Name'].lower():
        suggestions.append("Implement output buffering "
                           "(ob_start())")
        if row['Processor Energy (Joules)'] > 50:
            suggestions.append("Apply memory pooling")

    # Object lifecycle
    if 'object' in row['Function Name'].lower():
        suggestions.append("Reuse objects in loops")

# ===== WORKSTATION RULES =====
if row['Environment'] == 'Workstation':
    # Slow queries
    if row['Execution Time (ms)'] > 2000:
        suggestions.append("Optimize query structure "
                           "and add indexes")

    # Query patterns
    query = str(row['Query/Code']).upper()
    if "GROUP BY" in query:
        suggestions.append("Use materialized views for "
                           "aggregations")
    if "JOIN" in query:
        suggestions.append("Add indexes on foreign keys")
```

```
    # ===== CROSS-ENVIRONMENT RULES =====
    if row['Predicted Priority'] == 1:
        cache_msg = ("Implement transient caching" +
                     (" for computations" if row['Environment']=='Lab'
                      else " for queries"))
        suggestions.append(cache_msg)

    return (" • " + "\n • ".join(suggestions)
            if suggestions else "No critical optimizations"
```

# Key Implementation Notes

- **Dual-Environment Processing**: Rules dynamically adapt to Lab (computational) vs. Workstation (database) contexts.
- **Empirical Thresholds**: All values derived from experimental baselines.
- **Data Validation**: $\pm 2\sigma$ outlier removal and Z-score standardization.
- **Feature Engineering**: Energy Coefficient of Variation (CoV) and Relative Speedup features added.
- **Reproducibility**: Random seed (42) ensures deterministic results

# Repository Reference

Full implementation:

`https://gitlab.utu.fi/tech/soft/visiiri/benchmarks/mdkhan-thesis-ai-framework`

`main/code/ai-model/trainClassifier.py` (Priority classifier)

`main/code/heuristics/heuristic_code_refactoring.py` (Enhanced suggestion engine)