VRIJE UNIVERSITEIT AMSTERDAM

UNIVERSITEIT VAN AMSTERDAM

Master Thesis

Empirical Assessment of Carbon Reduction Strategies in Enterprise Software Applications

by

Rutger Kool

(2677549)

First supervisor:	Ivano Malavolta
Daily supervisor:	Michiel Overeem
Second reader:	Vincenzo Stoico

July 7, 2025

Submitted in partial fulfillment of the requirements for
the joint UvA-VU degree of Master of Science in Computer Science

Empirical Assessment of Carbon Reduction Strategies in Enterprise Software Applications

Rutger Kool
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
r.g.kool@student.vu.nl

ABSTRACT

Context. Measuring and optimizing the carbon footprint of software presents methodological challenges. While the Software Carbon Intensity (SCI) specification enables standardized carbon measurement across applications, existing research has primarily focused on infrastructure-level optimizations rather than application-level configuration parameters.

Goal. This research evaluates carbon emission reduction strategies for software applications by applying the SCI specification to quantify the impact of application-level configuration parameters on software carbon emissions and identify effective optimization approaches.

Method. Controlled experiments are conducted using AFAS SB, a financial administration application, deployed in a laboratory environment with `PowerJoular` for energy measurement and system monitoring. The study evaluates five configuration categories across 79 functional test scenarios. Baseline carbon emissions are established from production Microsoft Azure metrics, followed by testing of individual configuration impacts and combined optimization strategies.

Results. Garbage collection optimization achieved statistically significant carbon reductions of 29.5-30.6%, while other configuration categories demonstrated minimal impact. An integrated optimization strategy combining the most effective parameters achieved 25.7% carbon reduction while maintaining complete functional correctness. Correlation analysis identified disk I/O read operations as a significant factor associated with carbon emissions across all tested configurations.

Conclusions. Application-level configuration optimization can achieve carbon emission reductions without compromising functionality or performance, with garbage collection optimization being the most effective strategy in this study. These findings demonstrate that developers can implement various configuration changes to reduce production carbon footprints, with the SCI specification providing standardized measurement capabilities for quantifying the effectiveness of optimization across different deployment scenarios.

1 INTRODUCTION

Organizations face growing pressure to address their environmental impact through reduced energy consumption and carbon emissions, driven by concerns over global warming [40]. The infrastructure of cloud computing plays a crucial role due to the energy consumption of data centers [37]. Previous research examining cloud software architectures has indicated limited scientific understanding of how software design choices influence energy consumption and carbon emissions, with the relationship between certain architectural decisions and energy efficiency reported as unclear and yet to be better investigated [64].

Studying the energy consumption of cloud-native systems is complicated by the abstraction introduced through virtualization and orchestration layers [3]. Estimation techniques have been developed that map container-level resource metrics to energy usage, enabling power-aware scheduling and profiling [48]. Telemetry-based methods support runtime observation of distributed behavior and provide additional insight into energy-related activity [72].

To convert energy measurements into carbon emissions, the Software Carbon Intensity (SCI) specification uses energy consumption data from software systems. SCI has been established as an ISO standard (ISO/IEC 21031:2024) [35] that provides a methodology for quantifying the rate of carbon emissions of software systems [32]. The specification enables consistent carbon assessment across different applications and deployment scenarios by combining operational emissions from energy consumption with embodied emissions from the manufacturing of hardware.

In this research, the Software Carbon Intensity methodology is used [32] to evaluate and optimize the carbon emissions of AFAS SB, a cloud-based enterprise software platform. The investigation extends previous research on software energy efficiency and green software metrics by combining baseline carbon assessment from production environment metrics with controlled experimental analysis. The research is conducted in a controlled laboratory environment. Automated test execution is used to simulate realistic user workloads.

The results reveal distinct effectiveness patterns across optimization strategies, with garbage collection optimization achieving significant carbon reductions of 29.5-30.6% while other approaches demonstrated negligible impact on AFAS SB's rate of carbon emissions. This pattern reflects AFAS SB's specific system constraints: correlation analysis shows that disk I/O read operations tend to be associated with carbon emissions ($r = 0.997$) while CPU utilization is negatively correlated ($r = -0.82$), suggesting the application tends to be constrained by disk I/O bottlenecks rather than computational capacity. Garbage collection optimization directly reduces disk I/O read operations because inefficient garbage collection leads to memory allocation patterns that increase disk access for object retrieval and heap management operations, as evidenced by the strong correlation between disk I/O activity and carbon emissions in the experimental data. Meanwhile, parallelism configurations that optimize computational throughput have a minimal impact on the actual performance constraints. The combined configuration achieved only 25.7% SCI reduction compared to individual

garbage collection optimizations, indicating optimization interference where additional configurations disrupted garbage collection effectiveness through altered resource allocation or conflicting bottlenecks.

The primary contributions of this research are:

- A structured approach for applying the SCI methodology to evaluate carbon emissions in enterprise web applications.
- Identification of energy consumption hotspots and their relationship to application-level configurations in the context of a real enterprise web application
- Practical optimization insights for reducing AFAS SB's carbon footprint while maintaining performance requirements
- A case study contribution to the Green Software Foundation (GSF), expanding the application of the SCI methodology[1]

The remainder of this thesis is organized as follows: Section 2 provides essential background on key concepts, tools, and technologies; Section 3 reviews related work in software carbon measurement; Section 4 presents the formal definition of the experiment using the Goal-Question-Metric framework; Section 5 details the experimental design, including subjects, variables, and hypotheses; Section 6 describes the execution infrastructure and methodology; Section 7 presents the findings; Section 8 discusses the implications of the results; Section 9 discusses threats to validity; and Section 10 concludes with recommendations and directions for future research.

2 BACKGROUND

This section provides background information on the key concepts used throughout this study.

2.1 Software Carbon Intensity Specification

The Software Carbon Intensity specification (ISO/IEC 21031:2024) provides a standardized methodology for determining the rate of carbon emissions of software systems [35]. The SCI represents the rate of carbon emissions per functional unit, also referred to as the SCI score. The SCI score is calculated using the following formula:

$$SCI = \frac{O + M}{R}$$

Operational emissions O capture energy-related carbon emissions, while embodied emissions M account for manufacturing carbon footprint. R represents the functional unit that defines how the application scales. Both operational and embodied emissions are measured in grams of carbon dioxide equivalent (gCO_2eq), a metric that expresses the warming potential of all greenhouse gases in terms of the equivalent amount of CO_2 [29].

Operational emissions are calculated as:

$$O = E \times I$$

Energy consumption E is measured in kWh and includes all energy used by provisioned hardware. This can be measured at datacenter, machine, service, or code execution levels. Carbon intensity I represents the grid carbon intensity in gCO_2eq/kWh. Grid-connected systems use marginal or average emissions factors, while isolated systems require appropriate local emissions factors.

[1]GSF case study submission: https://github.com/Green-Software-Foundation/sci-guide/pull/76

Embodied emissions are calculated using the formula:

$$M = TE \times \frac{TiR}{EL} \times \frac{RR}{ToR}$$

Total embodied emissions TE come from Life Cycle Assessment data of hardware components, measured in gCO_2eq. When detailed LCA data is unavailable, simplified estimation models are used. TiR represents time reserved for the software, while EL indicates the expected hardware lifespan, both measured in the same temporal units. RR quantifies resources reserved for the software, such as CPU cores. In cloud environments, this commonly uses virtual CPU percentages. ToR represents the total available resources in matching units.

Functional units commonly include API calls, users, transactions, database operations, or time-based units.

SCI provides a unified methodology for quantifying software carbon impact across technological contexts. The metric enables objective establishment and performance monitoring through measurement consistency, while facilitating informed architectural and operational choices throughout the software development lifecycle. The approach demonstrates universal applicability, accommodating evaluation scenarios ranging from complex distributed systems to individual application components across various deployment environments. Implementation strategies encompass three distinct optimization approaches: reducing electrical consumption for equivalent functionality, minimizing computational resource requirements, and leveraging temporal or geographic variations in the carbon content of energy sources. The methodology emphasizes direct emission reduction as the exclusive mechanism for score improvement, explicitly excluding offset-based strategies that do not address actual emission generation.

2.2 AFAS SB: Subject Application

AFAS SB serves as the subject application for this research. It is a cloud-native financial administration platform developed by AFAS Software [2], used for administrative tasks such as invoice processing and financial reporting, and is architecturally outlined in Figure 2. The platform serves 7,540 unique users over the three-month period April–June 2025 with an average weekly retention rate of approximately 90%.

Figure 1 shows the AFAS SB user interface, demonstrating the web-based dashboard that users interact with for financial administration tasks.

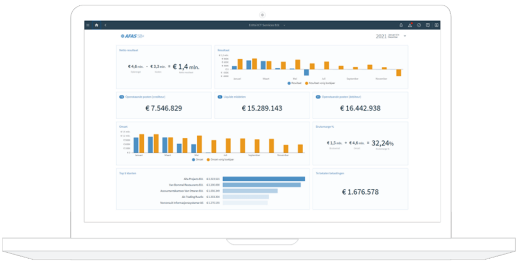

Figure 1: AFAS SB user interface showing the financial administration dashboard

2.2.1 System Architecture. AFAS SB implements a multi-tenant architecture where a single application instance serves multiple customers while maintaining strict data isolation. The application adopts a Command Query Responsibility Segregation (CQRS) design [27], separating read and write operations for performance optimization.

AFAS SB is deployed across Microsoft Azure [52] using Azure Service Fabric [51] for orchestration and cluster management. Figure 2 provides a high-level overview of the production environment. Client requests enter through an Azure load balancer (`Clstr-LB`) and are routed to a proxy virtual machine (VM) scale set (`Proxy001`) that handles external-facing communication. These proxy components forward requests to internal Service Fabric clusters (`Clstr-SS`), which provide application lifecycle management and health monitoring capabilities [41]. The SQL elastic pools (`NextTenantPool`) manage database services, sharing computational resources across tenant databases. Supporting infrastructure includes dedicated proxy VMs (`SvcProxy-VM0` and `SvcProxy-VM1`), a logging VM running `Kibana` [23], and an Elastic-VM running `Elasticsearch` [22] for enhanced observability.

Figure 2 also includes three additional components: Azure Blob Storage (used for storing tenant attachments), a Login Platform (handling authentication), and Shared Services (supporting payment and banking integrations). The boundary used for the Software Carbon Intensity assessment includes only those components responsible for request handling, application execution, and data persistence within Azure: `Clstr-LB`, `Proxy001`, `Clstr-SS`, `NextTenantPool`, `SvcProxy-VM0`, `SvcProxy-VM1`, `Logging-VM`, and `Elastic-VM`.

3 RELATED WORK

This section highlights previous research on the SCI methodology, including approaches across various application domains.

3.1 Geographic and Temporal Optimization

Several studies have investigated the application of external factors, including geographic location and temporal variations in carbon intensity, to estimate emissions. `GreenCourier`, developed by Chadha et al. [10], focuses on carbon-aware scheduling of serverless functions across geographically distributed regions, calculating the SCI score by measuring GPU electricity consumption and mapping this to emissions using location-based and time-specific carbon intensity data from WattTime [73], a nonprofit organization that provides real-time marginal emissions data for electricity grids. The authors update carbon efficiency scores every five minutes to enable dynamic workload routing, resulting in an average reduction of 13.25% in carbon emissions per function invocation compared to traditional scheduling approaches. Similarly, Dodge et al. [20] evaluate geographic optimization and temporal shifting approaches for AI workloads, implementing algorithms such as "Flexible Start" and "Pause and Resume" that schedule workloads during periods of lower grid carbon intensity, achieving up to 30% reduction in operational emissions depending on the region and time of day selected.

3.2 System-Level Carbon Measurement

Research in system-level SCI implementation has focused on integrating carbon accounting directly into operating systems and hardware. Schmidt et al. [69] implement the SCI specification through `carbond`. This Linux daemon defines system boundaries at the component level (CPU, memory, storage) and tracks both operational and embodied emissions with real-time grid carbon intensity monitoring. For embodied emissions, they calculate per-unit carbon costs by distributing manufacturing emissions across a component's expected lifetime. Köhler et al. [42] introduced `cMemento`, which enables carbon-aware memory placement decisions directly within the Linux operating system, offering a platform for tracking both operational and embodied carbon emissions across various units of work. These approaches provide comprehensive carbon accounting but require modifications to the operating system and direct access to hardware.

3.3 Model-Specific Carbon Assessment

A significant body of research has concentrated on applying SCI to machine learning and AI workloads. Everman et al. [25] apply the SCI specification to compare open-source GPT models during inference, using direct power measurements from both CPU and GPU combined with fixed carbon intensity values and embodied emissions calculations from a separate dataset. Hoffmann and Majuntke [33] extend this approach to large language model inference, demonstrating how SCI scores vary significantly between task categories and implementing an architecture that routes prompts to specialized models with optimal carbon efficiency for specific tasks.

An essential methodological contribution comes from Bashir et al. [4], who identify the "sunk carbon fallacy" in contemporary carbon accounting, which occurs when embodied carbon is inappropriately included in operational scheduling decisions, despite embodied emissions being fixed manufacturing costs that cannot be influenced by runtime choices. Their research compares standard SCI, operational SCI (oSCI), and total SCI (tSCI) approaches, demonstrating that using standard SCI for scheduling decisions can lead to counterproductive outcomes, where energy-inefficient servers with lower embodied carbon are selected, resulting in total emissions increasing by up to 30%. This work demonstrates the importance of applying SCI methodology appropriately for measuring and comparing carbon impacts of different operational strategies rather than for decision-making that conflates fixed and variable carbon costs.

3.4 Specialized Environment Applications

Research has also explored SCI applications in resource-constrained and edge computing environments. Sayeedi et al. [68] implement a green carbon footprint algorithm that extends the SCI methodology to IoT devices, incorporating device-specific parameters such as execution time, memory utilization, power consumption, and CPU temperature, with conversion factors calibrated for individual IoT hardware.

3.5 Code-Level Integration

At the application level, Brunnert and Gutzy [8] demonstrate how SCI can be operationalized by extending the OpenTelemetry Java

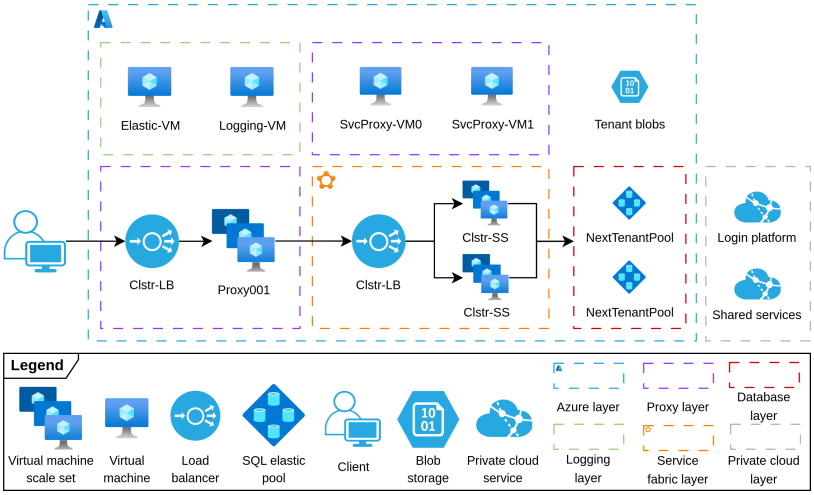

Figure 2: AFAS SB production architecture

Auto-Instrumentation Agent to calculate SCI metrics for individual transactions automatically. Their implementation captures fine-grained resource demands (CPU time, memory, storage, network usage). It then converts these to energy consumption, directly embedding carbon calculations within application monitoring pipelines.

3.6 Research Positioning

The reviewed literature reveals diverse approaches to SCI implementation, ranging from system-level integration to model-specific assessments and specialized environments. However, existing research typically employs either cloud-based estimation approaches or controlled laboratory measurements in isolation. This research addresses a methodological gap by combining both approaches: establishing baseline carbon metrics from a large-scale production cloud infrastructure system and conducting controlled experiments with different configurations. By evaluating how application-level parameters affect carbon emissions, this work contributes evidence-based guidance for reducing the environmental impact of software systems.

4 EXPERIMENT DEFINITION

To define the goal of this study, the research questions to achieve it, and the metrics associated with these questions to answer them, the Goal, Question, Metric (GQM) framework [5] is employed (Figure 3). This study aims to analyze AFAS SB for the purpose of evaluation and optimization with respect to its rate of carbon emissions from the point of view of software developers in the context of enterprise web applications. The research questions are defined as follows:

RQ1: What is the baseline rate of carbon emissions of AFAS SB in production?
Establishing a quantitative baseline is essential for understanding the carbon footprint of AFAS SB in its production environment. This baseline characterizes the overall carbon emissions of the system and identifies which infrastructure components contribute most significantly to total emissions. This question will be answered by applying the SCI methodology to resource utilization data collected from production monitoring systems.

RQ2: How do application-level configuration settings affect the rate of carbon emissions of AFAS SB?
Understanding the individual impact of specific configuration parameters enables the development of targeted optimization strategies that software developers can implement without requiring architectural changes. This evaluation identifies the configurations that provide carbon emission reductions and those that have negligible effects. This question will be addressed through controlled experiments determining the SCI score and system performance under different configuration conditions.

RQ3: To what extent can the rate of carbon emissions of AFAS SB be reduced by combining optimal application-level configurations while maintaining system performance and functionality?
Moving beyond individual configuration optimization opens the possibility of achieving greater carbon reductions through integrated approaches. This question quantifies the extent of emission reduction possible while maintaining system performance and functionality. This question will be evaluated by determining the combined effect of optimal configurations on the SCI score, execution performance, and functional correctness.

5 EXPERIMENT PLANNING

This section outlines the planning phase of the experiment following established guidelines for empirical software engineering research [74]. A structured approach defines the subjects, variables, hypotheses, and analysis methods that guide this investigation.

5.1 Subjects Selection

The subject of this experiment is AFAS SB. The selection was based on the application representing a contemporary web application

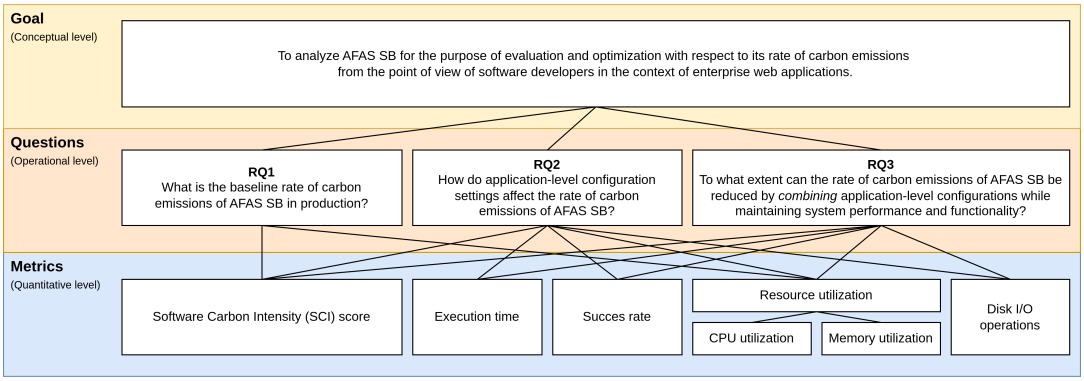

Figure 3: The application of the Goal, Question, Metric framework to the study

with significant production deployment, providing a relevant context for investigating carbon optimization strategies. The implementation offers extensive configuration parameters across runtime environment variables and application settings that can be manipulated to analyze their impact on energy consumption while maintaining application functionality. The experimental workload consists of 79 test scenarios selected from the application's functional test suite. These scenarios are part of AFAS SB's production testing framework, developed by the test team at AFAS Software using a custom parser with Dutch language support for business process validation. The scenarios are executed routinely by the test team for regression testing and quality assurance during software releases.

The test scenarios cover various business processes, including financial administration (31 scenarios), banking and cash management (18 scenarios), tax processing and compliance (12 scenarios), asset management (7 scenarios), and system configuration (11 scenarios). Scenario complexity ranges from simple 3-step operations to complex 89-step business workflows, with an average of 23.4 steps per scenario (median: 19 steps). During experimental execution, scenarios averaged 57.2 seconds duration (median: 20.5 seconds) with a 99.9% success rate across all experimental runs. The scenario selection focused on core business processes while prioritizing scenarios that require substantial computational effort from the underlying infrastructure.

5.2 Experimental Variables

Following standard practices [6], experimental variables are defined as follows:

5.2.1 Independent Variables. The independent variables represent the configuration parameters being manipulated in this experiment. These parameters were selected to investigate different configuration categories, with each variable group targeting distinct aspects of system behavior that may influence energy consumption through various mechanisms. Table 1 provides a comprehensive overview of all independent variables organized by functional category.

5.2.2 Dependent Variables. The dependent variables represent the outcomes measured during each experimental run to assess the impact of configuration changes. These variables capture quantifiable aspects of system behavior, including energy consumption, resource utilization, performance characteristics, and the SCI score. Table 2 details each dependent variable with its measurement unit and description.

5.2.3 Controlled Variables. The controlled variables ensure experimental validity by maintaining consistent conditions across all experimental runs. These variables are held constant to isolate the effects of the independent variables. Table 3 details each controlled variable and its standardization approach.

5.3 Experimental Hypotheses

Following the format used in empirical software engineering studies [39], formal statistical hypotheses are formulated for the comparative research questions.

5.3.1 For RQ1: Baseline Carbon Emissions. RQ1 is exploratory, focusing on establishing a baseline SCI score for AFAS SB in production. Since it involves data collection and analysis rather than comparative testing, formal statistical hypotheses are not formulated for this research question.

5.3.2 For RQ2: Impact of Application-Level Configurations. Let $\mu_{baseline}$ represent the mean SCI score under baseline configuration and μ_{config_i} represent the mean SCI score under configuration i, where i represents each of the 13 experimental configurations tested in Phase 2 (P2_E1 through P2_E5), encompassing five configuration categories: parallelism (4 configurations), logging (2 configurations), caching (2 configurations), compression (1 configuration), and garbage collection (4 configurations).

- H_0^1: $\mu_{config_i} = \mu_{baseline}$ for all $i \in \{1, 2, ..., 13\}$
- H_a^1: $\mu_{config_i} \neq \mu_{baseline}$ for at least one $i \in \{1, 2, ..., 13\}$

Here, the null hypothesis H_0^1 states that application-level configuration changes do not significantly affect the SCI score, while the alternative hypothesis H_a^1 states that at least one configuration change does have a significant effect on the SCI score.

5.3.3 For RQ3: Combined Application-Level Optimizations. Let $\mu_{baseline}$ represent the mean SCI score under baseline configuration and $\mu_{combined}$ represent the mean SCI score under the combined configuration that integrates the optimal SCI score settings from Phase 2.

Table 1: Independent variables

Variable Name	Parameter	Description
Parallelism Configuration	`DefaultMaxParallelism`	Sets the maximum degree of parallel task execution.
	`DefaultMaxLookAhead`	Determines how many operations can be prepared ahead of execution for improved scheduling.
Logging Configuration	`enableTracing`	Enables tracing functionality that tracks request flows.
	`consoleLogging`	Controls whether log messages are written to console output during application execution.
	`logLevel`	Defines the minimum severity level for log processing and determines log volume.
Caching Configuration	`maxCachedItems`	Sets the maximum number of items that can be stored in the cache simultaneously.
	`expirationInSeconds`	Determines how long cached items remain valid before expiration.
Compression Configuration	`enableResponseCompression`	Enables compression of HTTP responses.
Garbage Collection Configuration	`DOTNET_gcServer`	Enables server garbage collection mode with dedicated threads for multi-core environments.
	`DOTNET_gcConcurrent`	Enables concurrent garbage collection, allowing the collector to run with application threads.
	`DOTNET_GCConserveMemory`	Sets memory conservation level on a 0-9 scale for collection aggressiveness.
	`DOTNET_GCLOHThreshold`	Defines size threshold for Large Object Heap allocation in bytes.
	`DOTNET_-GCDynamicAdaptationMode`	Enables dynamic adaptation of garbage collection behavior based on allocation patterns.
	`DOTNET_GCRetainVM`	Controls whether the garbage collector releases virtual memory back to the operating system.
	`DOTNET_GCNoAffinitize`	Disables processor affinity for garbage collection threads.
	`DOTNET_GCHeapCount`	Limits the number of garbage collection heaps created for memory management.
	`DOTNET_GCHighMemPercent`	Sets memory usage threshold percentage for triggering garbage collection cycles.

Table 2: Dependent variables

Name	Unit	Description
Baseline SCI Score	gCO_2eq/h	SCI score calculated for production baseline assessment from Azure infrastructure metrics per hour.
Experimental SCI Score	$gCO_2eq/scenario$	SCI score calculated for controlled experiments per functional unit (test scenario) enabling direct comparison of equivalent workloads across different configurations.
Power	W	Average power consumption measured by PowerJoular during application execution.
Test Duration	seconds	Time required to complete the entire test suite execution.
Test Success Rate	%	Percentage of test scenarios that complete successfully.
CPU Utilization	%	Average percentage utilization of CPU resources during experimental run execution.
Disk I/O Read Operations	operations	Total number of disk I/O read operations performed during execution, measured via psutil process monitoring.
Disk I/O Write Operations	operations	Total number of disk I/O write operations performed during execution, measured via psutil process monitoring.

Table 3: Controlled variables

Name	Standardization Approach
Hardware Environment	All experiments are conducted on the same physical server with consistent resource allocation. The server is equipped with an Intel Xeon Silver 4112 @ 2.60GHz processor, 196GB RAM, 36TB storage, and a Debian 9 operating system.
Operating System	Both virtual machines run identical Windows Server 2022 installations.
Test Suite	A standardized set of 79 test scenarios is executed in identical sequence for each experimental run.
Application Version	The same AFAS SB 5.1 codebase and database schema are used across all experiments.

- H_0^2: $\mu_{combined} = \mu_{baseline}$
- H_a^2: $\mu_{combined} \neq \mu_{baseline}$

The null hypothesis H_0^2 states that combining the optimal SCI score configurations from Phase 2 does not significantly affect the SCI score, while the alternative hypothesis H_a^2 states that the combined configuration does have a significant effect on the SCI score.

5.4 Experiment Design

This experiment follows a factorial design [74] with a phased approach to evaluate the impact of different configuration parameters on energy consumption and performance. Each phase targets specific subsets of the independent variables defined in Section 4.2.1.

5.4.1 Baseline Calculation. This initial phase involves analyzing production metrics from the Azure Service Fabric deployment to establish a baseline SCI score for AFAS SB in its real-world environment. The baseline assessment targets CPU and memory utilization data collected over a one-week monitoring period from the core infrastructure components, including virtual machine scale sets, standalone virtual machines, and SQL elastic pools that comprise the primary production deployment.

5.4.2 Experimental Phases. The controlled experiments are organized into three phases. **Phase 1** establishes the reference point using default values for all independent variables through baseline configuration measurements (P1_E1), serving as the reference point for subsequent comparisons. **Phase 2** employs parameter manipulation across five configuration categories, with each experiment targeting specific independent variable groups. **Phase 3** implements an integrated optimization strategy (P3_E1) combining the most effective parameter settings identified across all variable groups in Phase 2. Table 4 presents the comprehensive experimental design across all phases, detailing the specific parameter settings and evaluation purposes for each configuration. For each experimental condition, 10 repetitions are performed to account for measurement variability.

Table 4: Comprehensive experimental design across all phases

Experiment	Configuration	Parameter Settings	Evaluation Purpose
P1_E1	Baseline Configuration	Standard configuration	Control condition for statistical comparisons against all experimental configurations.
P2_E1	parallel_4_lookahead_default	DefaultMaxParallelism = 4 DefaultMaxLookAhead = 24	Evaluate energy efficiency when reducing thread count to mitigate resource contention, as fewer active threads can result in energy savings when hardware resources are oversubscribed [59].
P2_E1	parallel_16_lookahead_default	DefaultMaxParallelism = 16 DefaultMaxLookAhead = 24	Assess energy consumption patterns when increasing parallelism beyond optimal levels, as the energy consumption of a processor can increase substantially with additional active cores [59].
P2_E1	parallel_default_lookahead_12	DefaultMaxParallelism = 8 DefaultMaxLookAhead = 12	Examine how reduced lookahead affects energy efficiency, as the granularity of work assignment significantly impacts thread control overhead and overall performance [62].
P2_E1	parallel_default_lookahead_48	DefaultMaxParallelism = 8 DefaultMaxLookAhead = 48	Evaluate energy implications of increased lookahead to ensure adequate work chunks for balanced thread utilization while minimizing scheduling overhead [62].
P2_E2	reduced	enableTracing = true consoleLogging = false logLevel = Warning	Evaluate the energy and performance impact of reduced log verbosity while keeping trace instrumentation enabled, since logging activity is known to introduce CPU, memory, and I/O overhead [19].
P2_E2	minimal	enableTracing = false consoleLogging = false logLevel = Error	Evaluate the energy and performance impact of disabling trace instrumentation and retaining only conservative logging, as prior evidence in mobile systems suggests that low logging rates are unlikely to affect energy consumption [14].
P2_E3	extended_expiration	maxCachedItems = 10000 expirationInSeconds = 300	Examine how extending cache duration affects performance and efficiency, supported by evidence that longer retention reduces request load and improves responsiveness in cached systems [38].
P2_E3	larger_cache	maxCachedItems = 20000 expirationInSeconds = 60	Assess how increasing cache size improves runtime efficiency, based on findings that cache size has the strongest influence on energy and performance among cache configuration parameters [9].
P2_E4	no_compression	enableResponseCompression = false	Test disabling compression, as [56] shows compression can take more time and energy than transferring uncompressed data.
P2_E5	MemoryConservation	gcServer = 1 gcConcurrent = 1 GCConserveMemory = 7 GCLOHThreshold = 140000	Test whether increasing GC frequency and compaction reduces memory usage under constrained conditions, as similar tuning has impacted memory efficiency in embedded systems [12].
P2_E5	DynamicAdaptation	gcServer = 1 gcConcurrent = 1 GCDynamicAdaptationMode = 1 GCRetainVM = 0	Evaluate whether enabling runtime heap resizing improves responsiveness under memory pressure, following similar adaptive strategies shown effective in Java VMs [75].
P2_E5	ThreadEfficient	gcServer = 1 gcConcurrent = 1 GCNoAffinitize = 1 GCHeapCount = 4	Test whether tuning GC thread count improves throughput, as poor GC thread sizing can cause performance loss [44].
P2_E5	ReducedFootprint	gcServer = 1 gcConcurrent = 1 GCRetainVM = 0 GCHighMemPercent = 46 GCConserveMemory = 5	Assess whether limiting retained memory and heap growth reduces footprint over time, as compaction-based strategies have helped reduce memory fragmentation [12].
P3_E1	Carbon-Optimized	Combined optimal settings from Phase 2	Evaluate combined effectiveness of the best-performing configuration from each Phase 2 category, provided they achieve at least 5% SCI score reduction.

5.5 Data Analysis

The data analysis employs a structured statistical methodology to support consistent and interpretable results.

5.5.1 Statistical Framework. Statistical test selection is based on distributional assessment using Shapiro-Wilk tests [70]. Mann-Whitney U tests are primarily employed for two-group comparisons due to non-normal distributions in the experimental data [46]. Effect sizes are calculated using Cohen's d with pooled standard deviation, following conventions of small ($|d| \geq 0.2$), medium ($|d| \geq 0.5$), and large ($|d| \geq 0.8$) effects [17]. Statistical significance is evaluated at the conventional threshold of $\alpha = 0.05$ [26].

Bootstrap confidence intervals provide a robust assessment of effect size estimates [21]. Multiple comparison correction using the Holm method controls family-wise error rate across multiple statistical tests [34]. Post-hoc power analysis assesses the adequacy of sample sizes for detecting observed effect sizes [17]. Correlation analysis using Pearson correlation coefficients [61] examines relationships between resource utilization metrics and carbon intensity measures to identify underlying performance drivers.

5.5.2 Research Question-Specific Analyses. The analysis approach is tailored to address each research question with specific statistical methods.

RQ1: Baseline carbon emissions characterization through descriptive analysis of production data, including SCI score calculation and service-type comparisons.

RQ2: Configuration effects evaluated through pairwise comparisons against the baseline.

RQ3: Combined optimization assessment comparing the integrated optimal SCI score configurations against the baseline.

6 EXPERIMENT EXECUTION

This section describes the execution environment, measurement setup, and experimental procedures employed in this study.

6.1 Baseline Calculation Pipeline

To establish the baseline carbon emissions for AFAS SB in production, raw Azure telemetry data must be systematically processed and converted into SCI scores. This baseline calculation is implemented through a multi-stage processing pipeline in Python. Raw CPU and memory metrics are preprocessed to extract instance-specific utilization patterns, followed by data standardization and resampling to consistent 5-minute intervals.

The processed utilization data is fed into the `Impact Framework` [30], a reference implementation for SCI calculations developed by the Green Software Foundation. The framework converts resource utilization metrics into standardized carbon emissions using a manifest that defines the complete calculation pipeline: energy estimation from CPU and memory usage, embodied emissions allocation based on hardware specifications, and final SCI score computation.

6.2 Experimental Infrastructure and Implementation

All experiments are conducted in the Green Lab, a research laboratory at Vrije Universiteit Amsterdam equipped for measuring software energy consumption [45]. The Green Lab provides precise power measurement capabilities, ensuring reliable and repeatable energy measurements. The experiments utilize a server with an Intel Xeon Silver 4112 processor at 2.60 GHz, 196 GB of RAM, and a `Debian` 9 operating system running the `Proxmox` virtualization platform [65].

A two-VM setup is established to separate application execution from test orchestration, ensuring that energy consumption measurements are not affected by test execution overhead. The Application VM runs Windows Server [55] 2022 Datacenter, hosting the AFAS SB application and Microsoft SQL Server [54] 2022. The Test Execution VM also operates Windows Server 2022 Datacenter, running the `Playwright` [50] test automation framework and a PowerShell [53] orchestration script that manages test execution and result processing.

Figure 4 illustrates the complete system architecture deployed in the Green Lab environment. The experimental execution is coordinated through `Experiment Runner` [67], an open-source framework that automates experiment configuration, execution management, and data collection across experimental runs. The framework is extended with custom functionality to handle virtual machine communications and manage energy measurement tools.

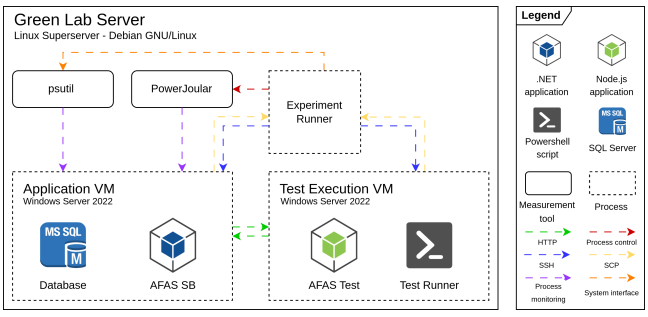

Figure 4: Experimental setup architecture

The architecture employs complementary measurement approaches coordinated through this framework. `PowerJoular` provides process-specific energy consumption measurements while isolating consumption attributable to individual applications, eliminating interference from unrelated system processes [58]. `psutil`, a cross-platform Python library that accesses operating system APIs, delivers system resource metrics including CPU utilization, memory consumption, and I/O operations [66]. The communication infrastructure utilizes distinct protocols for different operational requirements: SSH for parameter configuration and application lifecycle management, SCP for measurement data collection, and HTTP for test execution communication between the Test Execution VM and Application VM. Each experimental run follows a protocol that begins with environment reset procedures, including virtual machine state restoration to eliminate residual processes, SQL Server database restoration to standardize content and schema, and system cache clearing to ensure consistent initial conditions between experimental treatments.

The experimental treatments are implemented through two configuration approaches targeting the different independent variable categories. Parallelism and garbage collection settings are controlled through environment variables that are applied before the application startup. Logging, caching, and compression configurations are modified through the application's configuration file, with parameter values updated according to the experimental design specifications. The complete replication package and dataset are publicly available on GitHub [43].

6.3 Carbon Emissions Calculation Methodology

The SCI calculation uses distinct computational approaches for Azure production data (RQ1) and Green Lab experimental data (RQ2-RQ3), both implementing the same core methodology with different parameter values and calculation processes.

6.3.1 Azure Production SCI Calculation Pipeline. The Azure baseline calculation uses the `Impact Framework` with a manifest-defined pipeline that processes resource utilization data through sequential computational steps. This pipeline approach enables grouping by instance type and aggregating results across the heterogeneous Azure infrastructure with multiple instance types and configurations:

CPU Energy Calculation. CPU utilization percentages are converted to power consumption using linear interpolation between utilization points [0, 10, 50, 100%] and corresponding power factors [0.12, 0.32, 0.75, 1.02]. The resulting CPU energy calculation:

$$E_{cpu} = \frac{\text{CPU\_factor} \times \text{TDP} \times \text{duration\_seconds}}{1000 \times 3600} \tag{1}$$

where CPU_factor represents the interpolated power coefficient based on utilization, TDP is the thermal design power for each instance type (e.g., 280W for `Standard_E16ads_v5`), and duration_seconds is the time interval represented by each Azure monitoring data point.

Memory Energy Calculation. Memory energy uses a fixed coefficient of 0.000392 watts per GB [16]:

$$E_{memory} = \text{memory\_GB} \times 0.000392 \times \frac{\text{duration\_seconds}}{3600} \quad (2)$$

where memory_GB represents the total memory allocation for each instance.

Embodied Emissions Calculation. Using a 4-year hardware lifespan (126,144,000 seconds):

$$M = \text{TE} \times 1000 \times \frac{\text{duration\_seconds}}{126144000} \times \frac{\text{vCPUs\_allocated}}{\text{total\_server\_cores}} \quad (3)$$

where TE represents total embodied emissions from Cloud Carbon Footprint coefficients (e.g., 2010.54 kg CO_2eq for `Standard_E16ads_v5`), vCPUs_allocated is the number of virtual CPUs assigned to each instance, and total_server_cores represents the total number of vCPUs that can be allocated on the underlying Azure instance type.

Final SCI Computation. Converting duration to hours for the functional unit:

$$\text{SCI} = \frac{(E_{cpu} + E_{memory}) \times 355 + M}{\text{duration\_seconds}/3600} \quad (4)$$

using 355 gCO_2eq/kWh grid carbon intensity for The Netherlands, representing the average value from ElectricityMaps [24] for the baseline measurement week (February 3-9, 2025).

6.3.2 Green Lab Experimental SCI Calculation. The experimental SCI calculation implements direct computational steps using measured power consumption data. Unlike the baseline assessment, all experiments use identical hardware specifications, eliminating the need for instance-type grouping and enabling direct calculation:

Energy Conversion. Measured power consumption is converted to energy:

$$E = \frac{\text{power\_watts} \times \text{duration\_seconds}}{1000 \times 3600} \quad (5)$$

where power_watts represents the average power consumption measured by `PowerJoular` during test execution.

Operational Emissions. Using grid carbon intensity:

$$O = E \times 272 \quad (6)$$

applying 272 gCO_2eq/kWh grid carbon intensity for The Netherlands, representing the monthly average from ElectricityMaps [24] for April 2025 when the controlled experiments were conducted.

Embodied Emissions. Using the same 4-year hardware lifespan (35,040 hours):

$$M = 1449.84 \times 1000 \times \frac{\text{duration\_seconds}}{126144000} \times \frac{8}{24} \quad (7)$$

using Green Lab server specifications with 1449.84 kg total embodied emissions and 8-core allocation representing the Application VM's vCPU allocation from the total 24 VM cores allocated across the experimental setup (8 cores for Application VM + 16 cores for Test Execution VM).

Final SCI Computation. Using test scenarios as the functional unit:

$$\text{SCI} = \frac{E \times 272 + M}{\text{scenario\_count}} \quad (8)$$

where scenario_count represents the number of functional test scenarios executed during each experimental run.

The embodied emissions component utilizes the Cloud Carbon Footprint (CCF) coefficients methodology [15], an open-source framework that enables the calculation of Total Embodied Emissions values for cloud infrastructure components. The CCF methodology distributes hardware manufacturing emissions across expected component lifetime, accounting for CPU architecture, memory configuration, and storage specifications.

For the baseline calculations addressing RQ1, Azure instance specifications were collected through direct inspection of resources in Azure, with detailed hardware specifications obtained from CloudPrice [1]. These specifications were integrated into the CCF coefficients repository to enable the calculation of TE values for the Azure production environment. For the controlled experiments addressing RQ2 and RQ3, separate TE values were calculated using the Green Lab server hardware specifications.

7 RESULTS

This section presents the empirical findings from the controlled experiments investigating carbon emission optimization strategies in AFAS SB. Results are organized by research question, presenting quantitative data and statistical analyses.

7.1 Baseline Assessment (RQ1)

RQ1 establishes the baseline rate of carbon emissions for AFAS SB in production by analyzing Azure deployment metrics collected over a representative monitoring period.

7.1.1 Production Environment Characterization. The baseline assessment analyzed 46 virtual machine instances across 12 resource groups over a one-week monitoring period in February 2025. The production environment exhibited substantial performance heterogeneity, with a system-wide mean SCI score of **14.162 gCO_2eq/h** per infrastructure instance (SD = 11.011, n = 46). Table 5 presents the system-level carbon performance characteristics.

Table 5: Baseline system-wide carbon performance metrics for AFAS SB production environment

Metric	Value	Unit
Mean SCI Score	14.162	gCO_2eq/instance-hour
Standard Deviation	11.011	gCO_2eq/h
Median SCI Score	12.625	gCO_2eq/h
Coefficient of Variation	70.8%	–
Total Energy Consumption	216.78	kWh
Total Carbon Emissions	109,446.11	gCO_2eq
Sample Size	46	instances

The high coefficient of variation (70.8%) reflects the heterogeneous nature of the production deployment, which includes different infrastructure types with distinct operational roles. Scale sets serve various functions, with some acting as proxy servers while others host the core application. Workload distribution varies across clusters based on demand patterns.

7.1.2 Service Type Performance Distribution. Analysis of carbon performance by service type reveals distinct efficiency patterns

across the AFAS SB infrastructure. Table 6 presents carbon performance characteristics categorized by Azure component types.

Table 6: Carbon performance characteristics by service type in AFAS SB production deployment

Service Type	Instances	Mean SCI (gCO_2eq/h)	SCI Range (gCO_2eq/h)	Carbon Share (%)
VM Scale Set	30	18.257	1.973 – 33.221	74.2
SQL Elastic Pool	12	10.902	5.691 – 29.844	20.1
Virtual Machine	4	9.295	1.939 – 22.361	5.7

VM Scale Sets, representing 65.2% of instances, account for 74.2% of total carbon emissions with the highest mean SCI score (18.257 gCO_2eq/h). Despite having the lowest instance count, SQL Elastic Pools contribute 20.1% of emissions, indicating higher per-instance carbon intensity. The substantial performance ranges within each service type (e.g., 1.973–33.221 gCO_2eq/h for VM Scale Sets) demonstrate heterogeneous efficiency characteristics even within deployment categories.

7.1.3 Carbon Intensity Distribution Characteristics. Statistical analysis of SCI score distribution provides insights into system performance variability. Table 7 presents distribution characteristics across all instances. The substantial interquartile range (20.381 gCO_2eq/h) and the observation that 90% of instances fall below 32.903 gCO_2eq/h while the maximum reaches 33.221 gCO_2eq/h suggest that a small number of high-impact instances drive overall system carbon intensity.

Table 7: SCI score distribution analysis

Metric	Value (gCO_2eq/h)
10th Percentile	1.979
25th Percentile	7.592
50th Percentile (Median)	12.625
75th Percentile	27.974
90th Percentile	32.903
95th Percentile	33.050
Maximum	33.221
Minimum	1.938
Interquartile Range	20.381

7.1.4 Carbon Intensity Visualization. Figure 5 displays the hierarchical SCI score distribution, where rectangle sizes represent relative carbon contributions across service types and resource groups. The treemap visualization confirms the concentration of carbon emissions within specific infrastructure components, with `Cluster004-RG` and `Cluster003-RG` dominating the carbon footprint despite representing a minority of total instances.

7.2 Configuration Impact Analysis (RQ2)

Research Question 2 investigates how application-level configuration settings affect the rate of carbon emissions of AFAS SB through the evaluation of five configuration categories. Controlled experiments were conducted to compare configurations against the baseline SCI score of 0.457 $gCO_2eq/scenario$ (±0.031, n = 10), established in the Green Lab environment.

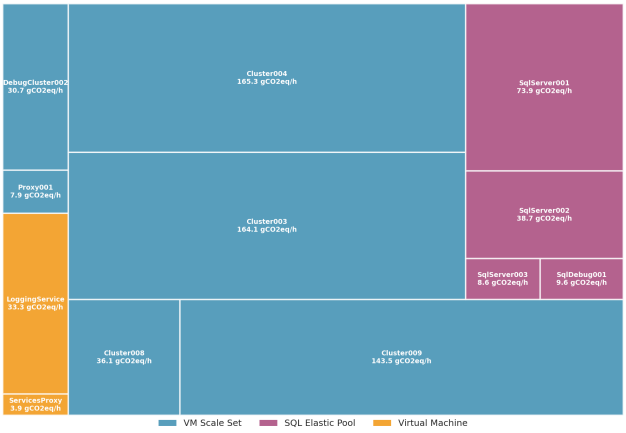

Figure 5: Carbon intensity distribution by service type and resource group

7.2.1 Configuration Performance Analysis. Table 8 presents the complete performance characterization across all experimental configurations, including statistical validation and effect size measurements. Notable values indicating significant improvements are highlighted in bold.

Analysis reveals distinct performance patterns across configuration categories. All garbage collection configurations (P2_E5) achieved SCI scores below 0.323 $gCO_2eq/scenario$, representing 29.5–30.6% improvements over baseline. These configurations also demonstrated the largest effect sizes (4.75–5.25) and achieved perfect statistical power (1.0). In contrast, parallelism configurations (P2_E1) showed modest improvements ranging from 0.015% to 5.3%, with the parallel_16_lookahead_default configuration achieving the best performance (0.433 $gCO_2eq/scenario$). However, these improvements did not reach statistical significance after multiple comparison correction, with an achieved power of 0.24.

7.2.2 Statistical Significance and Multiple Comparison Analysis. Statistical comparisons were performed using Mann-Whitney U tests due to non-normal distributions identified through Shapiro-Wilk testing. The statistical analysis employed conservative multiple comparison correction using the Holm method to control family-wise error rate across 56 statistical tests. Six configurations achieved statistically significant SCI score reductions at the conventional $\alpha = 0.05$ level before correction: `P2_E2_minimal` ($p = 0.045$), `P2_E3_extended_expiration` ($p = 0.021$), and all four garbage collection configurations (`P2_E5_MemoryConservation`, `P2_E5_DynamicAdaptation`, `P2_E5_ThreadEfficient`, `P2_E5_ReducedFootprint`) with p-values of 0.00018.

However, after applying the Holm correction for multiple comparisons, only four configurations maintained statistical significance, with all four configurations achieving Holm-corrected p-values of 0.010. The configurations with robust statistical evidence after correction were exclusively from the garbage collection optimization category (P2_E5), demonstrating considerable practical significance according to Cohen's conventions.

Bootstrap confidence intervals confirmed the robustness of these findings, with all significant configurations maintaining confidence

Table 8: Configuration analysis with performance metrics and statistical validation

	SCI Score (gCO_2eq/scenario)					Duration	Power	Success	Effect	Bootstrap CI	Original	Holm	Achieved
Configuration	**Mean**	**Std**	**Min**	**Max**	**Median**	**(sec)**	**(W)**	**Rate (%)**	**Size**	**(95%)**	**p-value**	**p-value**	**Power**
P1_E1 Baseline	0.457	0.031	0.442	0.546	0.447	5169	41.81	99.74	–	–	–	–	–
P2_E1_parallel_4_lookahead_default	0.457	0.031	0.441	0.544	0.448	5172	41.73	99.74	0.00	-0.86 – 0.86	0.970	1.0	0.24
P2_E1_parallel_16_lookahead_default	0.433	0.036	0.394	0.504	0.445	4885	42.03	99.87	0.71	-0.12 – 1.68	0.385	1.0	0.24
P2_E1_parallel_default_lookahead_12	0.445	0.017	0.399	0.460	0.449	5022	41.98	99.87	0.48	-1.11 – 1.01	0.678	1.0	0.24
P2_E1_parallel_default_lookahead_48	0.440	0.032	0.399	0.498	0.449	4959	42.03	100.0	0.56	-0.40 – 1.37	0.970	1.0	0.24
P2_E2_reduced	0.448	0.023	0.398	0.496	0.448	5056	41.94	100.0	0.34	-0.67 – 1.00	0.970	1.0	0.52
P2_E2_minimal	0.427	0.025	0.395	0.449	0.443	4806	42.19	100.0	1.06	0.59 – 1.96	0.045	1.0	0.52
P2_E3_extended_expiration	0.430	0.024	0.390	0.450	0.441	4902	40.96	99.87	0.99	0.59 – 1.71	0.021	0.63	0.69
P2_E3_larger_cache	0.438	0.025	0.388	0.459	0.447	5014	40.67	100.0	0.67	-0.32 – 1.25	0.678	1.0	0.69
P2_E4_no_compression	0.438	0.027	0.390	0.487	0.443	5004	40.76	99.87	0.68	-0.11 – 1.39	0.054	1.0	0.48
P2_E5_MemoryConservation	**0.322**	0.024	0.304	0.359	0.307	**3670**	41.19	100.0	**4.79**	**3.83 – 11.70**	**0.00018**	**0.010**	**1.0**
P2_E5_DynamicAdaptation	**0.317**	0.021	0.306	0.357	0.307	**3604**	41.33	100.0	**5.25**	**4.03 – 48.20**	**0.00018**	**0.010**	**1.0**
P2_E5_ThreadEfficient	**0.322**	0.025	0.303	0.361	0.306	**3663**	41.27	99.74	**4.75**	**3.80 – 11.36**	**0.00018**	**0.010**	**1.0**
P2_E5_ReducedFootprint	**0.320**	0.024	0.303	0.355	0.306	**3646**	41.17	100.0	**4.91**	**3.92 – 12.02**	**0.00018**	**0.010**	**1.0**
P3_E1_Carbon-Optimized	**0.340**	0.026	0.307	0.362	0.359	**3866**	41.26	100.0	**4.10**	**3.52 – 7.55**	**0.00018**	**0.010**	**1.0**

intervals that exclude zero effect. The `P2_E5_DynamicAdaptation` configuration achieved the strongest statistical evidence with a bootstrap confidence interval of [4.03, 48.20]. However, the wide upper bound reflects bimodal execution time patterns (8 measurements at 3470-3500s, 2 at 4070-4090s) characteristic of system testing variability.

7.2.3 Performance Distribution Visualization. Figure 6 displays SCI score distributions across configuration categories using violin plots, revealing both central tendency and distribution characteristics. The violin plots demonstrate clear distributional separation between baseline performance and garbage collection optimizations, with no overlap between baseline measurements and the P2_E5 configuration family. The tight distributions within each garbage collection configuration (standard deviations 0.021–0.025) indicate consistent performance improvements across repetitions. For other configuration categories, the distributions overlap substantially with baseline performance, consistent with the statistical analysis, which shows non-significant improvements. The parallelism configurations show slight leftward shifts (lower SCI scores) but with considerable overlap with baseline measurements.

7.2.4 Multi-Dimensional Performance Analysis. Figure 7 presents improvement percentages and effect sizes across multiple performance metrics, providing a complete view of configuration impacts beyond SCI scores. The heatmap analysis reveals that carbon improvements are consistently accompanied by execution time reductions, with garbage collection configurations achieving 29.0–30.3% duration improvements alongside their carbon benefits. This pattern indicates that optimization mechanisms address fundamental system bottlenecks rather than creating isolated improvements.

Power consumption improvements vary across configurations, with caching and compression optimizations achieving 2.0–2.7% reductions while garbage collection configurations show more modest gains (1.2–1.5%). This pattern suggests that for the configurations tested in this study, carbon benefits derive primarily from execution time efficiency rather than direct energy reduction. CPU utilization patterns exhibit mixed responses across configurations, indicating distinct optimization pathways within the tested categories.

7.2.5 Resource Utilization Correlation Analysis. Correlation analysis between system resource utilization metrics and SCI scores identifies quantitative relationships within the experimental data. Figure 8 displays correlation patterns between resource metrics and SCI scores across all tested configurations. Analysis reveals a strong negative correlation between CPU utilization and SCI scores ($r = -0.82$, $p < 0.001$), indicating that higher CPU utilization tends to correspond to lower carbon emissions per functional unit within the tested configurations. This counterintuitive relationship is primarily driven by shorter execution times: configurations achieving higher CPU utilization (such as garbage collection optimizations) execute the test suite in 3604-3670 seconds compared to the baseline's 5169 seconds, representing 29.0-30.3% duration reductions alongside their carbon improvements. Disk I/O read operations demonstrate a near-perfect positive correlation with SCI scores ($r = 0.997$, $p < 0.001$), indicating that carbon emissions tend to be strongly associated with the frequency of disk access operations across the experimental conditions.

7.2.6 Hypothesis Testing Results for RQ2. Statistical evidence supports rejecting H_0^1 in favor of H_a^1. Specifically, four configurations from the garbage collection category maintain statistical significance under conservative Holm correction, with all four achieving Holm-corrected p-values of 0.010. The effect sizes for significant configurations (4.75–5.25) indicate considerable practical relevance.

7.3 Combined Configuration Analysis (RQ3)

RQ3 evaluates the carbon reduction potential of integrated optimization strategies through the carbon-aware configuration (P3_E1), which combines the most effective parameter settings identified across all configuration categories in the Phase 2 experiments.

7.3.1 Configuration Integration. The carbon-optimized configuration combines parameter settings from the best-performing configuration in each Phase 2 category, provided they achieve at least 5% SCI score reduction compared to baseline. The integrated approach

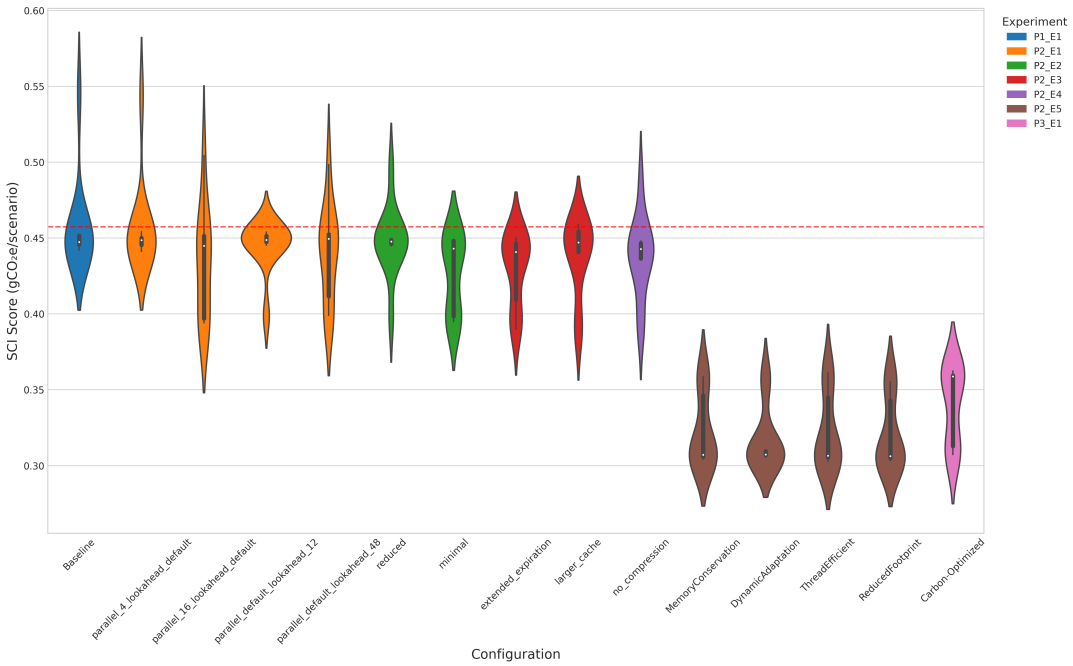

Figure 6: SCI score distributions across configuration categories, with the red line representing the mean SCI score of the baseline configuration

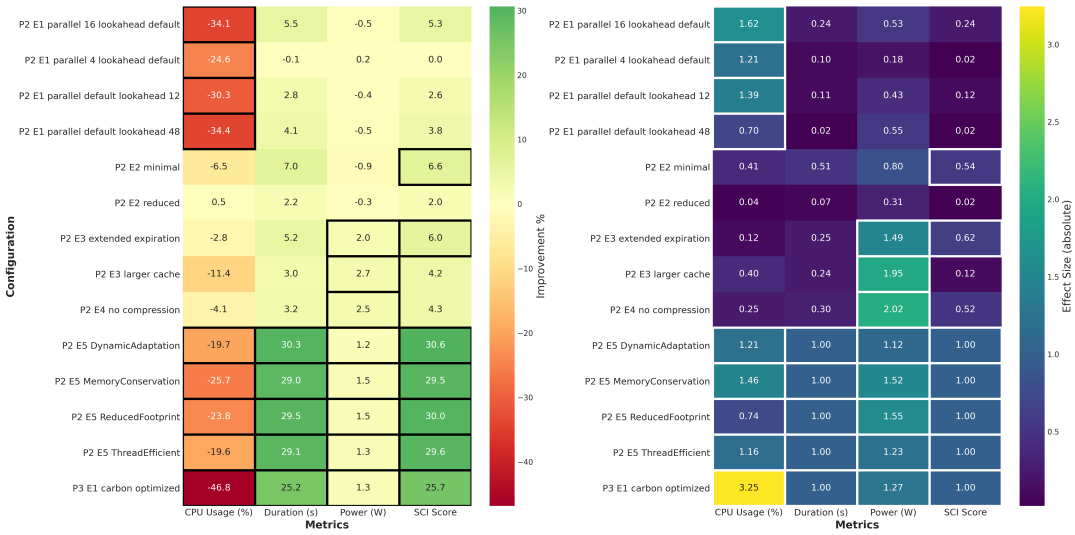

Figure 7: Configuration impact heatmap showing improvement percentages (left) and effect sizes (right) across performance metrics. Black borders indicate statistical significance for improvements, white borders indicate significance for effect sizes.

combines the `DynamicAdaptation` garbage collection configuration (30.6% reduction, best in category), parallel_16_lookahead_default parallelism configuration (5.3% reduction, best in category), minimal logging configuration (6.6% reduction, best in category), and extended_expiration caching configuration (5.9% reduction, best in category). The compression category's best performer (no_compression, 4.2% reduction) was excluded as it fell below the 5% inclusion threshold.

7.3.2 Integrated Configuration Performance. The carbon-optimized configuration achieved a mean SCI score of 0.340 gCO_2eq/scenario (±0.026, n = 10). The baseline performance was 0.457 gCO_2eq/scenario (±0.031, n=10). This represents a 25.7% reduction in SCI score while maintaining 100% functional correctness across all test scenarios.

Mann-Whitney U test yielded U = 0.0 (p = 0.010), indicating complete distributional separation. No overlap occurred between the measurement ranges of the two conditions, with the optimized

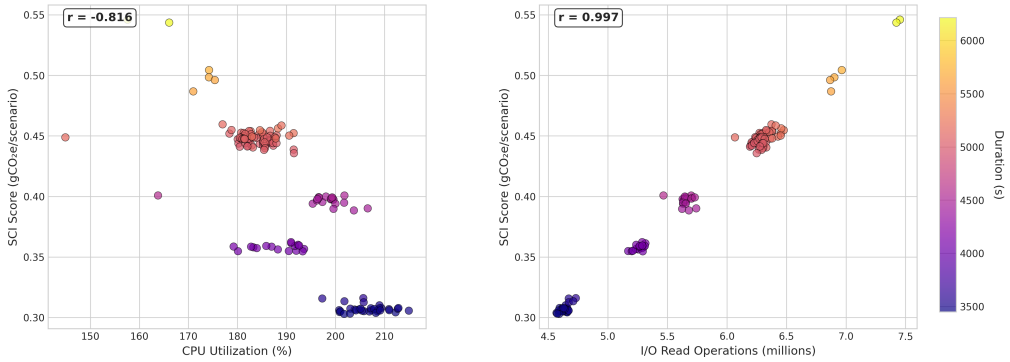

Figure 8: Resource utilization patterns vs SCI scores showing correlation relationships: CPU utilization (r = -0.82, p < 0.001) and disk I/O read operations (r = 0.997, p < 0.001) colored by test duration

configuration's maximum value (0.362 gCO_2eq/scenario) falling below the baseline minimum value (0.442 gCO_2eq/scenario).

7.3.3 Statistical Validation and Effect Size. The carbon-aware configuration demonstrates strong statistical evidence with a Holm-corrected p-value of 0.010 and achieved statistical power of 1.0. Cohen's d effect size of 4.10 indicates considerable practical significance, with the bootstrap confidence interval [3.52, 7.55] for the effect size. The mean difference between baseline (0.457) and optimized (0.340) configurations yields an absolute SCI reduction of 0.117 gCO_2eq/scenario. A 95% confidence interval for this mean difference [0.092, 0.142] gCO_2eq/scenario indicates that the actual improvement lies between 20.1% and 31.1% reduction with 95% probability.

7.3.4 Hypothesis Testing Results for RQ3. Statistical evidence supports rejecting H_0^2 in favor of H_a^2. The carbon-optimized strategy maintains functional reliability while achieving substantial improvements in SCI score and performance. The effect size (Cohen's d = 4.10) indicates considerable practical significance.

8 DISCUSSION

This study demonstrates that application-level configuration optimization achieved substantial SCI score reductions in AFAS SB, with empirical evidence revealing distinct effectiveness patterns across configuration categories.

8.1 Performance Drivers and Configuration Effectiveness

Correlation analysis reveals that disk I/O read operations tend to be a significant factor associated with carbon emissions (r = 0.997), with CPU utilization being negatively correlated (r = -0.82), indicating that disk I/O activity tends to be associated with carbon efficiency across configuration categories. Garbage collection optimization appears most effective because it may address this significant associated factor: inefficient garbage collection tends to create allocation patterns that may increase disk I/O read operations for object access, heap traversal, and memory management operations. The garbage collection configurations tested (`MemoryConservation`, `DynamicAdaptation`, `ThreadEfficient`, and `ReducedFootprint`) all target different aspects of heap management that may reduce these disk access patterns. Garbage collection optimizations tend to reduce disk I/O read operations and simultaneously improve both execution time (29.0–30.3% reduction) and carbon efficiency. The consistency of substantial improvements across four distinct garbage collection strategies aligns with existing research, which shows that application performance improvements can be achieved through various garbage collection optimization techniques, such as memory management optimization [13], heap management tuning [7], and collector strategy selection [63]. While the improvements in this study primarily derive from performance gains that reduce total execution time, research also demonstrates that garbage collector selection can achieve energy reductions of up to 47% [71]. This confirms that garbage collection optimization offers multiple pathways for carbon reduction in managed runtime environments.

Parallelism configurations optimize computational throughput, but AFAS SB was constrained by disk I/O operations, rather than CPU capacity, which explains its limited effectiveness. Other configurations similarly failed to address the system's primary bottlenecks, resulting in negligible changes in their resulting SCI scores. The consistent correlation between execution time reduction and SCI score improvement across effective configurations indicates that faster execution was achieved without proportional increases in power consumption, demonstrating efficiency gains per functional unit rather than mere speed optimization. This pattern aligns with research showing that performance improvements can correspond with genuine efficiency gains: Chan-Jong-Chu et al. found that better-performing mobile applications consumed less energy overall [11], while Nahrstedt et al. showed that superior data processing libraries reduced total energy consumption for equivalent tasks [57]. The garbage collection optimizations therefore achieved genuine efficiency improvements, delivering both faster execution and improved resource utilization per functional unit.

8.2 Integration Strategy Results

The carbon-optimized configuration achieved 0.340 gCO_2eq/scenario (25.7% improvement) but performed worse than individual garbage

collection configurations (0.317-0.322), indicating optimization interference rather than additive benefits. This suggests that additional optimizations can disrupt the effectiveness of garbage collection, potentially through altered resource allocation or conflicting bottlenecks.

The integrated configuration maintained the pattern observed in individual garbage collection optimizations: simultaneous carbon reduction and execution time improvement (25.2%). This demonstrates that in the AFAS SB case study, the environmental and performance benefits consistently occur together across both separate and combined optimization approaches. The achievement of substantial carbon reduction while maintaining 100% functional correctness indicates that garbage collection optimization delivers environmental improvements without compromising application functionality within the tested scenarios.

8.3 Production Environment Implications

The baseline assessment reveals that VM Scale Sets contribute 74.2% of total carbon emissions in the AFAS SB production deployment, representing the infrastructure components that host the application instances optimized in this study. This finding provides context for interpreting the controlled experiment results: the application-level configuration optimizations achieved SCI score reductions of up to 30.6% for the AFAS SB application software that runs on these dominant emission sources. While the experimental environment differs from production complexity, the results indicate optimization potential for the software deployed on the infrastructure components responsible for the majority of production carbon emissions.

The carbon optimization approach demonstrated in this study operates within existing virtual machine infrastructure, focusing on software configuration parameters. This complements infrastructure-level optimization strategies, such as those shown in carbon-aware VM management frameworks, which achieve carbon reductions through rightsizing underutilized instances (e.g., 48% savings by downsizing VM specifications) and regional migration to areas with lower grid carbon intensity [60]. For AFAS SB's production deployment, such infrastructure-level analysis could identify instances with suboptimal resource allocation, while the validated application-level optimizations address software efficiency within the allocated resources.

When presented with the findings of this study, AFAS expressed interest in implementing garbage collection optimizations, viewing them as easily implementable with potential benefits. Broader application of energy-aware optimization practices would require additional organizational knowledge development beyond the immediate scope of these specific configurations.

8.4 Interpretation Limitations

The controlled experimental environment differs significantly from production complexity, which limits confidence in the observed effectiveness patterns. Bootstrap confidence intervals reflect genuine experimental variability, with bimodal performance patterns in some garbage collection configurations producing wide confidence bounds while maintaining statistical significance. Garbage collection optimization showed consistent benefits under standardized conditions. However, production environments involve workload variability and resource contention that may alter the relative effectiveness of different configuration categories.

The architectural specificity of the findings constrains the applicability of the effectiveness hierarchy observed across configuration categories. The identification of garbage collection optimization as the dominant approach reflects AFAS SB's specific system characteristics, and may not generalize to applications with different performance bottlenecks or architectural constraints.

8.5 Statistical Rigor and Multiple Comparison Considerations

The experimental design involved 56 statistical comparisons across configuration categories, substantially increasing Type I error probability without multiple comparison correction. The application of the Holm correction for multiple comparisons provides conservative statistical control but may obscure genuine configuration effects with smaller effect sizes. The achievement of statistical significance for garbage collection configurations under this conservative approach strengthens confidence in these findings.

The low statistical power achieved for non-garbage collection categories (0.24–0.69) indicates insufficient sample sizes to detect smaller effect sizes reliably. This limitation suggests that configurations showing modest improvements may require larger sample sizes or more sensitive measurement approaches to establish statistically significant evidence of their effectiveness.

8.6 Implications for Carbon Optimization Research

Correlation analysis effectively identified system factors driving emissions, with I/O operations showing a strong correlation ($r = 0.997$). This suggests that resource utilization analysis can guide optimization prioritization and indicates that researchers should investigate opportunities for simultaneous performance and carbon improvements, rather than assuming inherent trade-offs.

8.7 Practical Experience and Recommendations for SCI Implementation

8.7.1 Primary Implementation Challenge: Telemetry-to-Energy Conversion. The most significant challenge encountered was converting cloud telemetry data to energy consumption estimates. Azure does not provide direct energy metrics, requiring estimation approaches using CPU utilization, memory usage, and other resource metrics. Each infrastructure component type presents different available metrics. VM Scale Sets provide CPU and memory utilization percentages, while SQL Elastic Pools report percentage-based metrics that require conversion to absolute values. Standalone VMs, on the other hand, offer direct memory usage in bytes. This heterogeneity necessitates component-specific processing pipelines, which prevents the use of standardized energy calculation approaches across the entire infrastructure.

8.7.2 Methodological Gaps in Current Standards. Organizations implementing SCI should anticipate substantial preprocessing efforts to make cloud telemetry data suitable for energy calculation. Current SCI guidance offers a mathematical framework but lacks

practical advice on handling inconsistencies and variations in cloud provider data, as well as metric availability. The absence of standardized telemetry-to-energy conversion methodologies forces practitioners to develop custom solutions, reducing comparability across different implementations and cloud providers.

8.7.3 Recommendations for Standard Enhancement. The Green Software Foundation and ISO/IEC should develop standardized conversion protocols for the telemetry formats of major cloud providers. These protocols should address variations in metric availability across service types and provide validated power modeling coefficients for different instance families. Additionally, guidance on handling missing or incomplete telemetry data would improve implementation reliability, as cloud monitoring systems may not provide consistent metric coverage across all infrastructure components.

8.7.4 Implementation Recommendations for Organizations. Organizations should begin by cataloging available telemetry metrics across their infrastructure components before attempting SCI implementation. Different cloud services provide varying metric granularity and formats, requiring custom preprocessing approaches. Practitioners should also plan for substantial effort in data pipeline development, as cloud monitoring APIs require translation layers to produce SCI-compatible input formats. Finally, organizations should recognize that estimation-based approaches introduce uncertainty that should be acknowledged when communicating carbon assessment results to stakeholders.

8.7.5 Common Implementation Pitfalls. The primary pitfall is underestimating the complexity of telemetry data preprocessing. Cloud providers' monitoring systems are designed for operational purposes rather than energy assessment, requiring significant transformation effort. Additionally, organizations should avoid assuming that cloud resource utilization metrics directly correlate with energy consumption patterns, as the relationship depends on hardware characteristics and power management features not visible through standard monitoring interfaces.

9 THREATS TO VALIDITY

This section discusses the threats to validity that may affect the interpretation of the results obtained during the experiments.

9.1 Internal Validity

The virtualized experimental environment introduces measurement overhead, as captured in energy calculations, but this overhead is not directly attributable to AFAS SB. Although VM configurations remain constant across treatments and complete system resets eliminate carry-over effects, the `Proxmox` hypervisor's resource consumption may affect measurement precision. This threat is controlled through identical virtualization settings across all experiments.

9.2 External Validity

Results specifically apply to AFAS SB's architecture and CQRS implementation. The controlled experimental setup differs substantially from contemporary cloud deployments in resource allocation models and operational complexity [28]. Additionally, the `Playwright` test scenarios represent typical usage patterns but constitute a subset of full application functionality, potentially missing business processes that exhibit different energy consumption characteristics. The experimental measurement protocol captures energy consumption during continuous test execution rather than variable production workloads that include idle periods, potentially overestimating the relative impact of computational optimizations compared to baseline power consumption in real deployment scenarios. These limitations restrict generalizability to other enterprise applications, programming languages, or deployment scenarios.

9.3 Construct Validity

The baseline calculation methodology relies on estimation models for CPU and memory power consumption, as direct energy measurements are not currently available in the Azure production environment. The CPU energy model used by the `Impact Framework` [31] employs power curve interpolation calibrated for AWS EC2 instances [18], rather than Azure instances, which may introduce systematic estimation errors. More critically, the linear memory power model from the `Cloud Carbon Footprint` methodology [16] oversimplifies non-linear DDR4 memory power characteristics [36]. However, similar TDP-based estimation approaches have been successfully applied to Azure workloads in official Green Software Foundation use cases, including production graph databases [47] and web applications [49], demonstrating that carbon measurement using CPU utilization and TDP modeling can yield meaningful SCI scores on Azure infrastructure [20]. This simplification can significantly affect the validity of the baseline calculations.

9.4 Conclusion Validity

The experimental design involves 56 statistical comparisons across configuration categories, which substantially increases the Type I error probability. While the Holm correction addresses multiple comparison issues, the conservative approach may obscure real configuration effects. The achieved statistical power for non-garbage collection categories (0.24-0.69) increases Type II error risk, where genuine effects with smaller magnitudes may remain undetected due to insufficient sample sizes. The use of 10 repetitions per configuration, rather than the conventional 30, was necessitated by experimental feasibility: 15 configurations would require 450 total runs at 30 repetitions each, with an estimated execution time of 570 hours, which exceeded the practical constraints of this controlled laboratory study.

10 CONCLUSION

This research demonstrates that application-level configuration optimization achieved substantial SCI score reductions in AFAS SB while maintaining operational performance and functional correctness. Garbage collection optimization emerged as the most effective approach, with garbage collection parameter tuning achieving statistically significant reductions of 29.5–30.6% under conservative multiple comparison correction. The carbon-optimized configuration demonstrated a 25.7% SCI score reduction while maintaining 100% functional correctness across all tested scenarios.

The correlation analysis identified I/O operations as a significant factor associated with carbon emissions ($r = 0.997$, $p < 0.001$) and

revealed CPU utilization constraints ($r = -0.82$, $p < 0.001$), providing clear evidence for optimization prioritization. The experimental approach established large effect sizes (Cohen's $d = 4.75–5.25$) for memory management optimizations while demonstrating minimal effects from other configuration categories. The baseline assessment confirmed that VM Scale Sets, which host the optimized application instances, contribute 74.2% of production carbon emissions, indicating substantial potential for system-wide improvements through application of the validated configuration strategies. The results suggest that carbon optimization for enterprise software applications can be achieved through evidence-based configuration assessment, providing a foundation for further research into application-level carbon reduction approaches.

REFERENCES

[1] 2024. CloudPrice.net – Cloud Instance Pricing and Specifications. https://cloudprice.net Accessed: 2025-02-24.

[2] AFAS Software B.V. 2025. AFAS Software: ERP and Business Automation Solutions. https://www.afas.nl Dutch family-owned company founded in 1996, Leusden, Netherlands.

[3] Lars Andringa, Brian Setz, and Vasilios Andrikopoulos. 2025. Understanding the Energy Consumption of Cloud-native Software Systems. In *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE '25)*. ACM, London, United Kingdom, 309–319.

[4] Noman Bashir, Varun Gohil, Anagha Belavadi, Mohammad Shahrad, David Irwin, Elsa Olivetti, and Christina Delimitrou. 2024. The Sunk Carbon Fallacy: Rethinking Carbon Footprint Metrics for Effective Carbon-Aware Scheduling. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '24)*. ACM, Redmond, WA, USA, 542–551.

[5] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 1994. The Goal Question Metric Approach. *Encyclopedia of Software Engineering* (1994), 528–532.

[6] Victor R. Basili, Forrest Shull, and Filippo Lanubile. 1999. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering* 25, 4 (1999), 456–473. https://doi.org/10.1109/32.799939

[7] Tim Brecht, Eshrat Arjomandi, Chang Li, and Hang Pham. 2001. Controlling Garbage Collection and Heap Growth to Reduce the Execution Time of Java Applications. In *Proceedings of the 16th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '01)*. ACM, Tampa Bay, FL, USA, 353–366.

[8] Andreas Brunnert and Ferdinand Gutzy. 2024. Extending the OpenTelemetry Java Auto-Instrumentation Agent to Publish Green Software Metrics. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24)*. ACM, London, United Kingdom, 1–3. HM Munich University of Applied Sciences.

[9] Yuan Cai, Marcus T. Schmitz, Alireza Ejlali, Bashir M. Al-Hashimi, and Sudhakar M. Reddy. 2005. Cache Size Selection for Performance, Energy and Reliability of Time-Constrained Systems. In *Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '05)*. IEEE, 923–926. https://doi.org/10.1109/ICCAD.2005.1560205

[10] Mohak Chadha, Thandayuthapani Subramanian, Eishi Arima, Michael Gerndt, Martin Schulz, and Osama Abboud. 2023. GreenCourier: Carbon-Aware Scheduling for Serverless Functions. In *Proceedings of the 9th International Workshop on Serverless Computing (WoSC '23)*. ACM, Koblenz, Germany, 1–7.

[11] Kwame Chan-Jong-Chu, Tanjina Islam, Miguel Morales Exposito, Sanjay Sheombar, Christian Valladares, Olivier Philippot, Eoin Martino Grua, and Ivano Malavolta. 2020. Investigating the Correlation between Performance Scores and Energy Consumption of Mobile Web Apps. In *Proceedings of the Evaluation and Assessment in Software Engineering (EASE '20)*. ACM, Trondheim, Norway, 190–199. https://doi.org/10.1145/3383219.3383239

[12] G. Chen, R. Shetty, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and M. Wolczko. 2002. Tuning Garbage Collection in an Embedded Java Environment. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA '02)*. IEEE. https://doi.org/10.1109/HPCA.2002.995696

[13] Wen-ke Chen, Sanjay Bhansali, Trishul Chilimbi, Xiaofeng Gao, and Weihaw Chuang. 2006. Profile-guided Proactive Garbage Collection for Locality Optimization. In *Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '06)*. ACM, Ottawa, Ontario, Canada.

[14] Shaiful Chowdhury, Silvia Di Nardo, Abram Hindle, and Zhen Ming (Jack) Jiang. 2018. An Exploratory Study on Assessing the Energy Impact of Logging on Android Applications. *Empirical Software Engineering* 23, 3 (2018), 1422–1461. https://doi.org/10.1007/s10664-017-9545-x

[15] Cloud Carbon Footprint. 2024. Cloud Carbon Footprint Coefficients. https://github.com/cloud-carbon-footprint/cloud-carbon-coefficients Accessed: 2025-02-24.

[16] Cloud Carbon Footprint. 2024. Methodology. https://www.cloudcarbonfootprint.org/docs/methodology/

[17] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Lawrence Erlbaum Associates.

[18] Benjamin Davy. 2021. Building an AWS EC2 Carbon Emissions Dataset. https://medium.com/teads-engineering/building-an-aws-ec2-carbon-emissions-dataset-3f0fd76c98ac

[19] Rui Ding, Hucheng Zhou, Jian-Guang Lou, Hongyu Zhang, Qingwei Lin, Qiang Fu, Dongmei Zhang, and Tao Xie. 2015. Log2: A Cost-Aware Logging Mechanism for Performance Diagnosis. In *Proceedings of the 2015 USENIX Annual Technical Conference (USENIX ATC '15)*. USENIX Association, Santa Clara, CA, USA, 139–150.

[20] Jesse Dodge, Taylor Prewitt, Remi Tachet Des Combes, Erika Odmark, Roy Schwartz, Emma Strubell, Alexandra Sasha Luccioni, Noah A. Smith, Nicole DeCario, and Will Buchanan. 2022. Measuring the Carbon Intensity of AI in Cloud Instances. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT '22)*. ACM, Seoul, Republic of Korea, 1877–1894.

[21] Bradley Efron and Robert Tibshirani. 1986. Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy. *Statist. Sci.* 1, 1 (1986), 54–75. https://doi.org/10.1214/ss/1177013815

[22] Elastic N.V. 2025. Elasticsearch: The Official Distributed Search and Analytics Engine. https://www.elastic.co/elasticsearch/. Accessed: 2025-06-28.

[23] Elastic N.V. 2025. Kibana: Your Window into the Elastic Stack. https://www.elastic.co/kibana/. Accessed: 2025-06-28.

[24] ElectricityMaps. 2025. ElectricityMaps: Real-time and Historical Electricity Data. https://www.electricitymaps.com/ Accessed: 2025-05-28.

[25] Brad Everman, Trevor Villwock, Dayuan Chen, Noe Soto, Oliver Zhang, and Ziliang Zong. 2023. Evaluating the Carbon Impact of Large Language Models at the Inference Stage. In *Proceedings of the IEEE International Conference on Performance Computing and Communications (IPCCC '23)*. IEEE. https://doi.org/10.1109/IPCCC59175.2023.10253886

[26] Ronald A. Fisher. 1925. *Statistical Methods for Research Workers* (1st ed.). Oliver & Boyd, Edinburgh, Scotland. https://archive.org/details/statisticalmeth00fish Foundational text introducing the 0.05 significance level criterion in statistical testing.

[27] Martin Fowler. 2011. CQRS. https://martinfowler.com/bliki/CQRS.html Technical article on Command Query Responsibility Segregation pattern.

[28] Sukhpal Singh Gill, Adarsh Kumar, Harvinder Singh, Manmeet Singh, Kamalpreet Kaur, Muhammad Usman, and Rajkumar Buyya. 2022. Sustainable Cloud Computing: Foundations and Future Directions. *Comput. Surveys* 55, 9 (2022), 1–37. https://doi.org/10.1145/3505244

[29] L. K. Gohar and K. P. Shine. 2007. Equivalent CO2 and its use in understanding the climate effects of increased greenhouse gas concentrations. *Weather* 62, 11 (2007), 307–311. https://doi.org/10.1002/wea.103

[30] Green Software Foundation. 2024. *Impact Framework: A framework to Model, Measure, siMulate and Monitor the environmental impacts of software.* Green Software Foundation. https://if.greensoftware.foundation/ Accessed: 2025-05-15.

[31] Green Software Foundation. 2024. Software Carbon Intensity (SCI) Pipeline Tutorial. https://if.greensoftware.foundation/pipelines/sci/

[32] Green Software Foundation Standards Working Group. 2024. *Software Carbon Intensity (SCI) Specification.* Green Software Foundation. https://sci.greensoftware.foundation Version 1.1.0.

[33] Geerd-Dietger Hoffmann and Verena Majuntke. 2024. Improving Carbon Emissions of Federated Large Language Model Inference through Classification of Task-Specificity. In *Proceedings of the 3rd Workshop on Sustainable Computer Systems (HotCarbon '24)*. ACM, Santa Cruz, CA, USA, 1–7.

[34] Sture Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics* 6, 2 (1979), 65–70.

[35] International Organization for Standardization. 2024. *ISO/IEC 21031:2024 Information technology – Sustainability – Software Carbon Intensity.* International Standard ISO/IEC 21031:2024. ISO, Geneva, Switzerland. https://www.iso.org/standard/86612.html

[36] JEDEC Solid State Technology Association. 2015. *DDR4 SDRAM Standard.* Technical Standard JESD79-4C. JEDEC, Arlington, VA. Latest revision of DDR4 specification including power consumption characteristics.

[37] Yichao Jin, Yonggang Wen, and Qinghua Chen. 2012. Energy Efficiency and Server Virtualization in Data Centers: An Empirical Investigation. In *Proceedings of the 1st IEEE INFOCOM Workshop on Communications and Control for Sustainable Energy Systems: Green Networking and Smart Grids (Green-Nets '12)*. IEEE, 133–138. https://doi.org/10.1109/INFCOMW.2012.6193512

[38] Joon Jung, Daniel Cicalese, Amogh Dhamdhere, Matthew Luckie, Jay Kroll, David Clark, and kc Claffy. 2019. Cache Me If You Can: Effects of DNS Time-to-Live. In *Proceedings of the 2019 Internet Measurement Conference (IMC '19)*. ACM, Amsterdam, Netherlands, 313–326. https://doi.org/10.1145/3355369.3355570

[39] Natalia Juristo and Ana M. Moreno. 2012. *Basics of Software Engineering Experimentation.* Springer.

[40] Jung-Sing Jwo, Jing-Yu Wang, Chun-Hao Huang, Shyh-Jon Two, and Hsu-Cheng Hsu. 2011. An Energy Consumption Model for Enterprise Applications. In *Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications (GreenCom '11)*. IEEE, 216–219. https://doi.org/10.1109/GreenCom.2011.42

[41] Gopal Kakivaya, Lu Xun, Richard Hasha, Shegufta Bakht Ahsan, Todd Pfleiger, Rishi Sinha, Anurag Gupta, Mihail Tarta, Mark Fussell, Vipul Modi, Mansoor Mohsin, Ray Kong, Anmol Ahuja, Oana Platon, Alex Wun, Matthew Snider, Chacko Daniel, Dan Mastrian, Yang Li, Aprameya Rao, Vaishnav Kidambi, Randy Wang, Abhishek Ram, Sumukh Shivaprakash, Rajeet Nair, Alan Warwick, Bharat S. Narasimman, Meng Lin, Jeffrey Chen, Abhay Balkrishna Mhatre, Preetha Subbarayalu, Mert Coskun, and Indranil Gupta. 2018. Service Fabric: A Distributed Platform for Building Microservices in the Cloud. In *Proceedings of the Thirteenth EuroSys Conference 2018* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3190508.3190546

[42] Sven Köhler, Lukas Wenzel, Andreas Polze, Benedict Herzog, Henriette Hofmeier, Manuel Vögele, and Timo Hönig. 2023. Carbon-Aware Memory Placement. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems (HotCarbon '23)*. ACM, Boston, MA, USA. https://doi.org/10.1145/3604930.3605714

[43] Rutger Kool. 2025. Empirical Assessment of Carbon Reduction Strategies in Enterprise Software Applications – Implementation and Data. https://github.com/rutgerkool/msc-thesis-sci-2025 GitHub Repository.

[44] Ram Lakshmanan. 2024. Java Performance Tuning: Adjusting GC Threads for Optimal Results. https://dzone.com/articles/java-performance-tuning-adjusting-gc-threads-for-optimal-results

[45] Ivano Malavolta, Vincenzo Stoico, and Patricia Lago. 2024. Ten Years of Teaching Empirical Software Engineering in the Context of Energy-Efficient Software. In *Handbook on Teaching Empirical Software Engineering*, Daniel Mendez, Davide Fucci, Marcos Kalinowski, Michael Felderer, Dietmar Pfahl, Rodrigo Spinola, Alessandra Bagnato, Valentina Lenarduzzi, and Davide Taibi (Eds.). Springer, 209–253.

[46] Henry B. Mann and Donald R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50–60. https://doi.org/10.1214/aoms/1177730491

[47] Tammy McClellan, Srinivasan Rakhunathan, Eric Friedeberg, Brandon Smith, and Samantha Beauchamp. 2024. *Graph Database Carbon Measurement Use Case.* SCI Implementation Use Case. Green Software Foundation. https://github.com/Green-Software-Foundation/sci-guide/blob/dev/use-case-submissions/dow-msft-Graph-DB.md Official GSF use case submission demonstrating SCI implementation on Azure VMs with Neo4J and TigerGraph databases.

[48] Hemant Kumar Mehta, Paul Harvey, Omer Rana, Rajkumar Buyya, and Blesson Varghese. 2020. WattsApp: Power-Aware Container Scheduling. In *Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. IEEE, 79–89. https://doi.org/10.1109/UCC48980.2020.00027

[49] Microsoft Corporation. 2024. *eShoppen Application SCI Implementation.* SCI Implementation Use Case. Green Software Foundation. https://github.com/Green-Software-Foundation/sci-guide/blob/dev/use-case-submissions/msft-eShoppen.md Official Microsoft GSF use case submission for web application carbon measurement on Azure App Service and SQL Database.

[50] Microsoft Corporation. 2024. *Playwright: Framework for Web Testing and Automation.* Microsoft Corporation. https://playwright.dev/ Accessed: 2025-05-10.

[51] Microsoft Corporation. 2025. Azure Service Fabric: A Distributed Systems Platform. https://azure.microsoft.com/en-us/products/service-fabric/. Accessed: 2025-06-28.

[52] Microsoft Corporation. 2025. Microsoft Azure: Cloud Computing Platform and Services. https://azure.microsoft.com/. Accessed: 2025-06-28.

[53] Microsoft Corporation. 2025. PowerShell: A Cross-platform Task Automation Solution. https://docs.microsoft.com/en-us/powershell/. Accessed: 2025-06-28.

[54] Microsoft Corporation. 2025. SQL Server: Relational Database Management System. https://www.microsoft.com/en-us/sql-server/. Accessed: 2025-06-28.

[55] Microsoft Corporation. 2025. Windows Server: The Platform for Building an Infrastructure of Connected Applications, Networks, and Web Services. https://www.microsoft.com/en-us/windows-server. Accessed: 2025-06-28.

[56] Aleksandar Milenković, Armen Dzhagaryan, and Martin Burtscher. 2013. Performance and Energy Consumption of Lossless Compression/Decompression Utilities on Mobile Computing Platforms. In *Proceedings of the 2013 IEEE 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '13)*. IEEE, San Francisco, CA, USA, 254–263. https://doi.org/10.1109/MASCOTS.2013.33

[57] Felix Nahrstedt, Mehdi Karmouche, Karolina Bargieł, Pouyeh Banijamali, Apoorva Nalini Pradeep Kumar, and Ivano Malavolta. 2024. An Empirical Study on the Energy Usage and Performance of Pandas and Polars Data Analysis Python Libraries. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024)*. ACM, Salerno, Italy, 58–68. https://doi.org/10.1145/3661167.3661203

[58] Adel Noureddine. 2022. PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. (June 2022), 1–4. https://doi.org/10.1109/IE54923.2022.9826760

[59] Stephen L. Olivier, Allan K. Porterfield, Sridutt Bhalachandra, and Jan F. Prins. 2013. Power Measurement and Concurrency Throttling for Energy Reduction in OpenMP Programs. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW '13)*. IEEE, Boston, MA, USA, 884–891. https://doi.org/10.1109/IPDPSW.2013.15

[60] Priyavanshi Pathania, Rohit Mehra, Samarth Sikand, Vibhu Saujanya Sharma, Vikrant Kaulgud, Nikhil Bamby, Sanjay Podder, and Adam P. Burden. 2024. Towards a Framework for Carbon-aware Virtual Machine Management. In *Proceedings of the 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C '24)*. IEEE, 250–251. https://doi.org/10.1109/ICSA-C61636.2024.00052

[61] Karl Pearson. 1895. Note on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London* 58, 347-352 (1895), 240–242.

[62] Performance Optimisation and Productivity Centre of Excellence. 2025. Chunk/-task grain-size trade-off (parallelism/overhead). https://co-design.pop-coe.eu/best-practices/chunk-task-grain-size-trade-off

[63] Filip Pizlo, Erez Petrank, and Bjarne Steensgaard. 2008. A Study of Concurrent Real-Time Garbage Collectors. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '08)*. ACM, Tucson, AZ, USA, 33–44.

[64] Giuseppe Procaccianti, Patricia Lago, and Stefano Bevini. 2015. A Systematic Literature Review on Energy Efficiency in Cloud Software Architectures. *Sustainable Computing: Informatics and Systems* 7 (2015), 2–10. https://doi.org/10.1016/j.suscom.2015.01.004

[65] Proxmox Server Solutions GmbH. 2024. *Proxmox Virtual Environment: Open-source server virtualization platform.* Proxmox Server Solutions GmbH. https://www.proxmox.com/en/proxmox-virtual-environment Accessed: 2025-04-20.

[66] Giampaolo Rodola. 2024. *psutil: Cross-platform lib for process and system monitoring in Python.* https://github.com/giampaolo/psutil Accessed: 2025-05-12.

[67] S2-Group. 2023. Experiment Runner: A Generic Framework to Automatically Execute Measurement-Based Experiments. https://link.springer.com/chapter/10.1007/978-3-031-71769-7_8 Originally available at: https://github.com/S2-group/experiment-runner.

[68] Md. Faiyaz Abdullah Sayeedi, Jannatul Ferdous Deepti, Anas Mohammad Ishfaqul Muktadir Osmani, Taimur Rahman, Safrin Sanzida Islam, and Md. Motaharul Islam. 2024. A Comparative Analysis for Optimizing Machine Learning Model Deployment in IoT Devices. *Applied Sciences* 14, 13 (2024), 5459. https://doi.org/10.3390/app14135459

[69] Andreas Schmidt, Gregory Stock, Robin Ohs, Luis Gerhorst, Benedict Herzog, and Timo Hönig. 2024. carbond: An Operating-System Daemon for Carbon Awareness. *ACM SIGENERGY Energy Informatics Review* 4, 3 (2024), 52–57. https://doi.org/10.1145/3698355.3698364

[70] Samuel Sanford Shapiro and Martin B. Wilk. 1965. An Analysis of Variance Test for Normality (Complete Samples). *Biometrika* 52, 3/4 (1965), 591–611. https://doi.org/10.1093/biomet/52.3-4.591

[71] Marina Shimchenko, Mihail Popov, and Tobias Wrigstad. 2022. Analysing and Predicting Energy Consumption of Garbage Collectors in OpenJDK. In *Proceedings of the 19th International Conference on Managed Programming Languages and Runtimes (MPLR '22)*. ACM, Brussels, Belgium, 3–15. https://doi.org/10.1145/3546918.3546925

[72] Oleh V. Talaver and Tetiana A. Vakaliuk. 2024. Dynamic System Analysis Using Telemetry. In *Proceedings of CS&SE@SW 2023 (CEUR Workshop Proceedings, Vol. 193)*. CEUR-WS.org. https://ceur-ws.org/Vol-193/

[73] WattTime. 2023. WattTime API: Real-Time Marginal Emissions Data. https://www.watttime.org/. Accessed: 2025-06-28.

[74] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering* (1st ed.). Springer.

[75] Fan Yang, Dhruva Gupta, Prashant Soman, Indranil Gupta, and Chandra Krintz. 2001. Automatic Heap Sizing: Taking Real Memory Into Account. In *Proceedings of the 2001 International Symposium on High-Performance Computer Architecture (HPCA '01)*. IEEE, 155–166. https://doi.org/10.1109/HPCA.2001.903257