Journal of Sustainable Software Engineering and Information Systems
Volume 2, Issue 2, 125-143.
e_ISSN: 3123-3007
https://e-journal.gomit.id/jsseis
DOI: 10.58723/jsseis.v2i2.260

Towards Green DevOps: Measuring and Improving Sustainability in AI-Enabled Microservices Architectures

Likinyo Ayorinde Paul
Lead City University, NIGERIA

Ressti Nurfitriani*
University of Bengkulu, INDONESIA

A.T John-Dewole
Lead City University, NIGERIA

V. Esther Jyothi
Deemed to be University, INDIA

Tenali Nagamani
Saveetha Engineering College, INDIA

*** Corresponding author:**
Ressti Nurfitriani, University of Bengkulu, INDONESIA. ✉resstinurfitriani@gmail.com

Article Info

Article history:

Received: October 16, 2026
Revised: October 26, 2026
Accepted: November 15, 2026

Keywords:
AI/ML systems
CI/CD sustainability metrics
Green DevOps
Microservices
Resource Efficiency

Abstract

Background: The growing deployment of artificial intelligence models through cloud-native microservices has increased the energy consumption, resource demand, and operational cost of production software. However, sustainability is rarely incorporated into continuous integration and continuous delivery pipelines as a measurable software-quality attribute.
Aims: This study aims to design and evaluate a Green DevOps framework that enables development and operations teams to measure, compare, and improve the sustainability of AI-enabled microservices.
Methods: The framework integrates energy consumption, estimated operational carbon emissions, CPU and memory utilization, storage activity, cloud expenditure, latency, reliability, and predictive performance into CI/CD and runtime-observability processes. A controlled prototype evaluation examined four deployment configurations using a containerized document-classification service.
Result: Compared with the static baseline, the optimized configuration reduced estimated energy consumption and operational carbon intensity by 57.1%, CPU consumption by 50.0%, and cloud cost per 1,000 successful requests by 51.2%. It improved p95 latency from 145 to 115 ms and throughput from 100 to 109 requests/s; macro-F1 changed from 0.918 to 0.914.
Conclusion: Coordinated optimization of scaling, orchestration, caching, and model precision can improve environmental and economic efficiency without materially compromising service quality, while CI/CD sustainability gates can make this evidence actionable.

To cite this article: Paul, Likinyo Ayorinde., Nurfitriani, Resti., John-Dewole, A. T., Jyothi, V. Esther., & Nagamani., Tenali. (2026). Towards Green DevOps: Measuring and Improving Sustainability in AI-Enabled Microservices Architectures. *Journal of Sustainable Software Engineering and Information Systems, 2*(2), 125-143. https://doi.org/10.58723/jsseis.v2i2.260
This article is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License ©2026 by author/s

INTRODUCTION

More and more AI applications are deployed as software services that can be frequently updated instead of as predictive models that are deployed once. Production AI systems typically incorporate a range of services that involve data preparation, model serving, application programming interfaces, message brokers, and/or database management and model serving (Gill et al., 2022; Lins et al., 2021). These services are delivered as microservices that can be independently deployed and updated through continuous integration and delivery (CI/CD) software development pipelines. This architecture allows rapid and independent scaling of services while facilitating deployment of multiple versions of the software (Ajiga et al., 2024; CEBECİ & Korçak, 2020). However, it also leads to persistent infrastructure consumption due to containers being launched repeatedly and tests

being run unnecessarily, as well as excessive telemetry, model serving, over-provisioned capacity, and other services that are under continuous build (Rodríguez et al., 2016; Zampetti et al., 2022).

The consideration of environmental sustainability for AI systems should not end with the training of the model. In fact, inference, data movement, storage, monitoring, retraining, container building, deployment, and even idle infrastructures can contribute to the overall resource impact of a production service (Patros et al., 2021; Raman et al., 2024). Therefore, the authors of the present study argue that the efficiency of computation should be considered together with the performance of prediction. Others have insisted on the systematic reporting of energy use and carbon emissions to enable a more equitable comparison of different computational experiments and constructs. More recent articles describe Green AI as an interdisciplinary approach to reducing the environmental impact of AI and making its deployment more financially feasible and operationally just (Bolón-Canedo et al., 2024; Verdecchia et al., 2023). The operational footprint of an AI service is dynamic and influenced by multiple factors including execution time, processor usage, memory usage, storage operations, data transfers, hardware efficiency, infrastructure overhead, and the carbon intensity of the used electricity (Lacoste et al., 2019; Lannelongue et al., 2021). Consequently, the same predictive model can deliver different environmental outcomes when deployed with various container sizes, different scaling rules, orchestration policies, caching configurations, and levels of numeric precision (Luccioni et al., 2023).

Measurement remains the key challenge. It is difficult to give an energetic value and compare services when volumes of workloads vary. A service that processes twice as many requests may consume more energy overall, but remain more efficient per request completed (Mytton, 2020; Treviso et al., 2023). Some systems may appear to be efficient because they process fewer requests and thus produce many errors or fail to meet latency targets. The measurement of sustainability thus requires measuring the consumption of resources for the production of a clearly defined useful unit of a software service (Kruchten & Gregory, 2023; Venters et al., 2017). The Software Carbon Intensity specification provides a measurement perspective framework for operational and embodied emissions corresponding to a functional unit of software output (ISO/IEC, 2024). For an online AI service, appropriate functional units may include one successful prediction, one processed document, or 1,000 completed requests. A functional unit allows environmental performance to be interpreted in relation to the service actually delivered rather than only the infrastructure that was active.

Measurement alone, however, does not guarantee improvement. Sustainability information must be delivered where engineering decisions are made. DevOps pipelines already evaluate source-code quality, functional correctness, vulnerability exposure, test coverage, deployment status, latency, throughput, and error rates. Energy consumption, carbon intensity, and resource efficiency are rarely assigned the same operational status. Sustainability is consequently evaluated after architecture and deployment decisions have been implemented, limiting its influence on model selection, container sizing, cache design, scaling thresholds, and release approval. Sustainable software engineering research has introduced energy-aware testing, efficient software design, resource monitoring, green architectural patterns, workload scheduling, and carbon-accounting tools. Nevertheless, the evidence remains fragmented across software engineering, cloud computing, AI, information systems, and environmental research. Danushi et al. (2024) identified environmentally sustainable software solutions distributed across guidelines, reference models, metrics, and implementation techniques rather than consolidated into ordinary software-delivery workflows. Herati et al. (2025) similarly argued that sustainable DevOps should extend conventional technical and economic concerns by incorporating environmental, social, and individual sustainability throughout the development and operational lifecycle.

A recent systematic review published in JSSEIS found that sustainable software and information-systems research remains dominated by conceptual and review-based studies, with comparatively limited quantitative and implementation-oriented evidence (Saputra et al., 2026). Another important prevention barrier identified is the absence of sufficiently standardised and empirically tested sustainability metrics. Technical teams are looking for repeatable measurements to distinguish

different deployments and architectures in order to adopt them easily. A study of information systems shows how sustainability concepts must connect with operational digital systems. Artificial intelligence, opensource technologies, and green software are key directions in the development of information systems aimed at environmental sustainability. While this convergence has begun, it has not yet led to a method that is widely used to compare the environmental and economic impacts of DevOps AI microservice configurations.

This gap is notable because optimisations of software and infrastructure can result in conflicting effects. Autoscaling reduces idle capacity, but overaggressive autoscaling can increase cold starts, retries, and latency of responses. Consolidation of containers can improve server utilisation, but excessive consolidation results in resource contention and failure. Caching responses avoids repeated model inference, but increases memory and can return stale results. Quantisation of models reduces both memory and computation, but can decrease the quality of predictions. A sustainability framework should consider effects on the environment, economy, operations, and predictions. Microservices introduce additional complexity. A model-serving container is never completely isolated from the other elements of the platform. Cache, the API gateway, telemetry, storage, and the other elements of the platform can affect resources. Defining the measurement boundary to only encompass the model calls can lead to underestimating resource consumption at the application level and misallocating overhead of shared infrastructure. A Green DevOps framework should clearly define its system, address the consumption of shared resources, and provide useful output that correlates to the quality of the service provided.

This study seeks to meet the existing gap by creating and assessing a Green DevOps framework for AI-enabled microservices. Sustainability instrumentation is embedded within the CI/CD pipeline and the runtime observability environment as the framework establishes a connection between resource and energy measurements and release metadata, volume of workloads, service quality, predictive performance, and cloud spending. In this context, sustainability is represented as an engineering signal that can be evaluated across different releases to endorse, oppose, or amend a deployment decision. Three research questions are addressed in this study. The first question explores how consistently the energy consumption, operational carbon emissions, resource utilisation, and cloud spending can be measured within the CI/CD pipeline and the observability workflow of an AI-enabled microservice application (Patterson et al., 2021). The second question explores how autoscaling, resource-aware orchestration, response caching, and model quantisation affect environmental and economic sustainability in a controlled workload situation. The third question explores whether the sustainability improvements identified can be preserved without affecting the response time, reliability, throughput of the system and predictive quality.

This study provides four contributions. First, it defines a working Green DevOps framework that integrates environmental, economic, operational, and predictive metrics. Second, it introduces sustainability gates that can be positioned within a DevOps pipeline. Third, it evaluates a combination of deployment optimisations in a staged approach as opposed to assessing each optimisation in isolation. Fourth, it presents pragmatic recommendations for MLOps and DevOps teams who wish to improve resource efficiency without trading off performance or quality of the model.

METHOD

Research Design

An iterative design and evaluation methodology was followed. A Green DevOps framework was constructed with a measurement model, instrumentation architecture, configuration-comparison tool, and a CI/CD (continuous integration and continuous deployment) decision framework. This framework was evaluated via a controlled repeated-measures benchmark in which an AI-powered document classification service was deployed as a suite of microservices packaged as containers (Behl, 2025; Obbu, 2025). The experimental procedure utilized automated load generation and hardware power probes to capture granular performance and energy consumption metrics across multiple iterative commits (Aneggi et al., 2026). The unit of analysis was one complete workload run under one deployment configuration. Each configuration was executed five times after a

standardised warm-up period, resulting in 20 evaluated runs. Repeated executions were implemented to account for temporary variations introduced by container initialisation, garbage collection, scheduling, cache loading, and the activities of the background platform (Agal & Katariya, 2026; Wen et al., 2023).

The same request corpus, workload sequence, model validation subset, infrastructure class, monitoring configuration, and system boundary were held constant for all evaluated configurations. The order of the configurations was rotated across the different executions to reduce the risk of a systematic bias for one configuration due to the order of execution, or variation within the platform that is introduced due to time (Chen et al., 2022; Raitt et al., 2024). The intent of the evaluation was to measure relative differences across configurations rather than to absolute energy values for all cloud platforms, hardware types, and/or AI workloads. Thus, environmental and economic impacts were normalised to useful software output. The main metric of interest was 1,000 completed classification requests. The study did not collect data related to humans, Personally Identifiable Information (PII), health data, or interact with users. The evaluation used a controlled, technical workload and infrastructure monitoring. Human-subject ethical approval for the benchmark was not required, as there was no direct interaction with Research Participants.

Green DevOps Framework

The proposed framework includes five independent layers: development and version control, CI/CD execution, deployment and orchestration, runtime observability, and sustainability decision support. Current DevOps metrics remain the same and include the status of functional- and non-functional-tests, vulnerability checks, model quality, latency, throughput, and reliability. Environmental and economic indicators are added to these metrics as complements and not as substitutions (Lago et al., 2024; Maroukian & Gulliver, 2020). At the development layer, each of these releases includes a version of the source code, model version, container image digest, infrastructure configuration, and optimisation profile. At the CI/CD layer, the pipeline captures build duration, test duration, processor and memory usage, artefact size, failed stages, and repeated execution (Milewicz et al., 2023; Salomoni et al., 2018). At the deployment layer, the orchestrator captures requested resources, consumed resources, replica counts, placements, restarts, and scale events. At the observability layer, infrastructure and application telemetry are joined using common release, configuration, workload, and run identifiers. The sustainability decision layer normalises the measurements, compares them with an approved baseline, and evaluates whether the release satisfies the defined quality and sustainability thresholds. Figure 1 illustrates the flow of evidence from software development and model versioning to sustainability-based release governance.

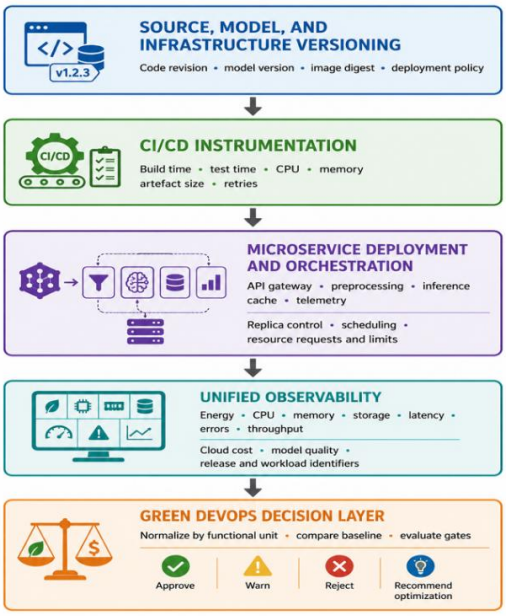


Figure 1. Green DevOps measurement and decision framework for AI-enabled microservices

The framework closes the gap between sustainability reporting and engineering action. A release is not classified as efficient only because its total energy consumption is low. Consumption is interpreted in relation to successful workload output, response latency, reliability, predictive quality, and operational cost. This approach prevents a slow, inaccurate, or unreliable configuration from appearing environmentally preferable merely because it performs less useful work.

Prototype Application

The prototype was a document-classification service that assigned one category to each submitted document. The service represented a common production-inference workload in which textual input is received through an API, normalised by a preprocessing component, classified by a transformer-based model, and returned with a predicted class and confidence score. The application contained an API gateway, preprocessing microservice, model-serving microservice, response-cache service, telemetry collector, and container-orchestration configuration. These components were selected because they represent a realistic cloud-native AI service and allow sustainability effects to be examined beyond the model process alone. The inference model used a compact transformer architecture fine-tuned for multiclass document classification. The full-precision implementation used 32-bit floating-point weights. The optimised implementation used post-training 8-bit quantisation. Both implementations used the same tokeniser, class definitions, preprocessing rules, and validation data. The prototype was selected because document classification combines repeated online inference, variable input length, observable latency, predictable output, and opportunities for safe response caching. The architecture also represents services used in content moderation, document routing, customer-service automation, misinformation screening, and enterprise knowledge management.

System Boundary

The operational system boundary included the API gateway, preprocessing service, inference service, cache, telemetry collector, namespace-level storage activity, and an allocated proportion of container-orchestration overhead. The allocation included resources consumed while executing and monitoring the assessed application. Model training, external client devices, networking outside the cluster, hardware manufacturing, infrastructure construction, and end-of-life hardware disposal were excluded. These exclusions were recorded because an operational inference estimate should not be interpreted as a complete lifecycle assessment. All configurations used the same three-worker-node cluster. Each worker was provided with eight virtual CPU cores and 32 GB of memory. The operating-system image, container runtime, orchestration version, network path, node class, monitoring intervals, and log-retention policy were held constant. CPU and memory requests and limits were declared for each container. Resource declarations allowed utilisation-based scaling and scheduling behaviour to be compared consistently. Namespace-level monitoring was used so that application supporting services were included rather than measuring just the model-serving process.

Workload Design

Each measured run processed 12,000 document classification requests. Documents varied in length and category, but the request corpus and order were fixed for all configurations. The workload was divided into warm-up, moderate load, burst load, and recovery phases so that steady-state and scaling behaviour could be observed. The warm-up phase initialised the model, application connections, and telemetry collection. The phase was excluded from the measurement window. The moderate-load phase represented ordinary traffic. The burst-load phase increased the arrival rate of model requests to a level to activate autoscaling. The recovery phase returned the request arrival rate to ordinary traffic to allow for scale-down. About 42% of requests showed a caching pattern. These requests included duplicates to model behaviours of real-world scenarios in which documents are submitted multiple times by numerous users or systems. The remaining requests were distinct, which guaranteed that the cache did not lessen the burden of the inference processing. We said a service achieved a successful response when the service provided a valid prediction in the time frame that was set and without any error. Failure to meet the time restriction, giving an invalid response or a request timed out did not decrease the total resource use, but they were not counted as successful functional response units.

Evaluated Configurations

Configuration C0 had four inference replicas that were always active, fully 32-bit floating-point inference, the default spread-oriented scheduler, and no response cache. A fixed replica count was used to achieve the objective for latency during peak demand, although it maintained excess inference capacity for low demand periods. Configuration C1 had utilisation-based horizontal autoscaling added to C0. The inference deployment could scale between two and six replicas using a target CPU utilisation of 60%. Full-precision inference, default scheduling, and the absence of response caching remained unchanged. This configuration isolated the effect of reducing static overprovisioning. Configuration C2 combined horizontal autoscaling with resource-aware orchestration and response caching. The placement policy favoured consolidation when sufficient CPU and memory headroom remained available. The cache stored eligible classification responses according to a time-to-live policy. The C2 cache key included normalised document content, preprocessing version, model version, and inference configuration. Including the model and preprocessing versions prevented an output produced by an earlier release from being reused after a model or data-processing change.

Configuration C3 added post-training 8-bit model quantisation to the C2 configuration. Quantisation reduces the precision used to represent model weights and execute supported operations. Integer-oriented inference can reduce model size, memory traffic, and computational demand, although the effect depends on model architecture and hardware support (Jacob et al., 2018). The quantised model was considered acceptable only when its macro-F1 score remained within 0.005 of the approved full-precision model. The threshold ensured that no gains in the environment would make up for a deterioration in predictive performance that would be considered unacceptable.

CI/CD Integration

Each configuration was built as an immutable container image and progressed through the same phases of the pipeline. The phases included verification of source code, unit tests, testing of the contract of the API, testing of the model, construction of a container, scanning of the container for dependencies and assessment of the container for vulnerabilities, deployment of the container to the namespace for benchmarking, warming up of the service, execution of the workload, export of telemetry, assessment of sustainability, and an overall assessment of the release (Le et al., 2023; Mulahuwaish et al., 2021). A unique run identifier was incorporated into runtime telemetry and configurations. The identifier was added to container labels and deployment annotations, logs and monitoring records, and analysis files. Each artefact could be traced back to the source revision, model version, image digest, policy, and the workload (Fehlis et al., 2025; Harshith et al., 2026).

This design was implemented after the implementation of the mandatory correctness and security checks. If a release did not pass the functional, compatibility, or security criteria, it was rejected before the sustainability criterion was considered. This ensured that there was no opportunity to justify an incorrect or insecure release by citing energy savings. The pipeline incorporated the sustainability criterion along with the other software-quality criteria and also included environmental and economic criteria. The report provided the absolute values, the normalised values, the difference from the approved baseline, the applicable thresholds, and the rationale for the release decision.

Observability Architecture

Infrastructure telemetry and application telemetry were collected. Infrastructure telemetry included CPU time and CPU utilisation, memory and storage residency, input and output of storage, network activities, instances of the application, container restarts, and scaling. Application telemetry included the cumulative and average request rates with the respective successful and failed response counts, as well as response and cache statistics. Operational energy consumption, sustainability, and financial telemetry collected emission estimates, the derived location-related emissions, and cloud expenditure. Macro-F1 and classification support results for the fixed validation subset were collected for predictive telemetry. Figure 2 illustrates the relationships among the workload, configuration, infrastructure and application telemetry, and the mechanism to automate the CI/CD decisions.

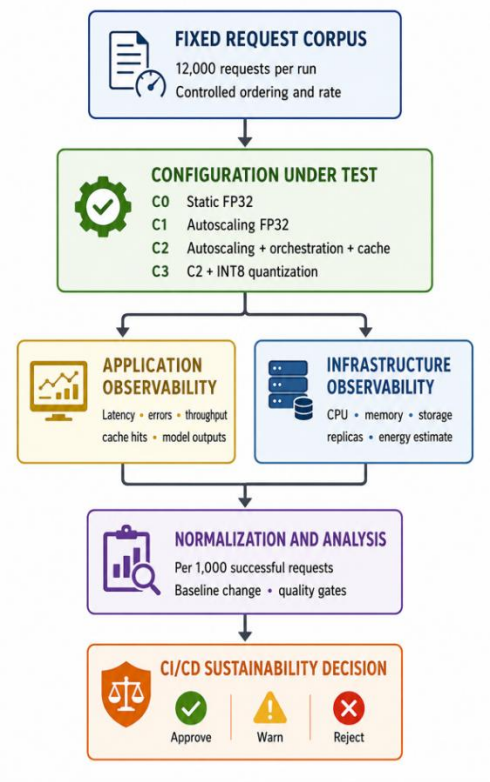


Figure 2. Controlled evaluation and telemetry integration process

Traceability was maintained through the measurement flow for each sustainability outcome to the software, model, container, infrastructure policy, and workload. Traceability is essential because the change in energy intensity can be the result of several factors, including the revision of the model, scaling behaviour, cache, additional monitoring, a software bug, or a change in successful request volume.

Sustainability Metrics

Energy consumption was estimated based on the demand of the processor, memory, execution time, and platform overhead. The estimation of energy followed the principle that an energy estimation must describe the hardware and system boundary it is defined for (rather than provide a context-free consumption value) (Henderson et al., 2020; Lannelongue et al., 2021).

The primary environmental metric was energy intensity per 1,000 successful requests:

$$\text{Energy intensity} = (\text{Total operational energy} \div \text{successful requests}) \times 1{,}000$$

Location-based operational carbon intensity was estimated by:

$$\text{Carbon intensity} = \text{Energy intensity} \times \text{grid-emission factor}$$

A fixed grid-emission factor of 0.57 kg CO_2e per kWh was applied to all configurations. Holding the factor constant enabled us to study the effects of the software and infrastructure configuration. The operational estimates of carbon include operational emissions, but do not include hardware emissions.

Cloud-cost intensity was calculated as:

$$\text{Cost intensity} = (\text{Allocated operational cost} \div \text{successful requests}) \times 1{,}000$$

Allocated operational cost included compute, memory, storage, and supporting service costs within the application boundary.

Resource efficiency was calculated as:

$$\text{Resource efficiency} = \text{Successful requests} \div \text{total operational energy}$$

CPU-hours, memory GB-hours, storage input/output, average replica count, and cache-hit rate were used as diagnostic indicators to explain changes in energy and cost. They were used to describe the reason for change in energy and cost, rather than replacing the normalised outcome metrics.

p95 response latency, request-error rate, throughput, and macro-F1 were the technical safeguards assessed. Latency and error measures pertained to service operational quality, while macro-F1 was an indicator of predictive quality.

Sustainability Gates

The framework outlined three CI/CD decision levels. First was a hard rejection gate. A release was rejected if it did not satisfy correctness, security, predictive quality, latency, or reliability expectations, regardless of how efficiently it consumed energy. The second was a sustainability regression gate. An increase of more than 5% in energy or carbon intensity (from the approved baseline), without a justified operational inclusion, resulted in a warning. The third was an optimisation gate. A release was recommended for promotion if it satisfied all technical safeguards and if energy or cloud-cost intensity were reduced by at least 10% compared to the previous release. The service-level objective targeted p95 latency of 200 milliseconds and a request-error rate of less than 0.5%. Predictive quality was considered at least as good as the reference when macro-F1 was at most 0.005 units lower. Use of normalised gates was essential. A release that processed more requests might consume more energy overall, but less energy per request. The converse was also true. Normalisation by successful output prevented both situations from producing misleading release decisions.

Statistical Analysis

For each configuration, descriptive statistics were computed based on five repeated runs. Changes relative to C0 were reported as absolute and percentage changes. Paired differences at the run-level were used since each configuration processed the same workload under the same infrastructure boundary. Bootstrap confidence intervals were computed with 10,000 resamples and assessed for direction, magnitude and degree of uncertainty of the differences. A combination of statistical and practical uncertainty was evaluated. A 10% reduction in energy use or cost was deemed an operationally significant impact. Latency was measured against the service-level objective. Predictive performance was assessed with the predefined macro-F1 non-inferiority margin of 0.005.

Validity and Reproducibility Controls

Controls for internal validity included a constant workload, same infrastructure class, persistent monitoring, constant model-validation subset, multiple runs, equivalent warmups, and random order of execution. Containers were restarted for each measured run. Cache contents were cleared prior to the first measurement, and the Cache was filled according to the declared request sequence. Combining outcome and diagnostic metrics was used to strengthen construct validity. Energy per successful request was used to represent energy consumption. CPU, memory, replicas, cache hits, and storage activity were used to explain the observed results. Latency, throughput, failures, and macro-F1 prevented sustainability from being interpreted independently of useful service quality.

The request corpus or the procedure used to generate the request corpus, source and model versions, container-image identifiers, deployment manifests, workload scripts, telemetry exports, energy-estimation parameters, cost-allocation rules, statistical-analysis scripts, and figure-generation code must be maintained to support the analyses. These materials should be uploaded to a research repository prior to submission. Given that the assessment was conducted on a single document-classification application, single model family, single container platform, and single controlled workload, external validity is still constrained. The relative impacts may differ for generative models, GPU-serving environments, streaming inference, image-processing systems, or applications with limited cache reuse.

RESULTS AND DISCUSSION

Result:

Framework Implementation

For each benchmark, the framework attributed a release identifier, configuration profile, model version, workload identifier, and infrastructure record. This enabled concreteness in the interpretation of the environmental and economic outcomes, as the outcomes could be understood as effects of a particular software release rather than outcomes for the cluster in general. The implementation demonstrated that sustainability evidence could be incorporated without the creation of a separate reporting workflow. Resource metrics that were collected for reliability and capacity planning were used, but their function was extended. CPU utilisation was used as an indicator of saturation and as an indication of energy consumption. The replica count was interpreted as a measure of scaling and an indicator of idle capacity. Cloud expenditure was interpreted together with energy because both were impacted by the efficiency of the workload and the resource allocation.

The pipeline formed a feedback loop between measurement and engineering action. For releases that exceeded the energy-intensity threshold, the report advised whether the change coincided with longer inference times, an increased number of replicas, more memory residency, less effective caching, more storage activity, or more failed requests. This mechanism transformed a broad sustainability goal into a series of concrete engineering questions. Various engineering actions could be taken to address algorithmic or preprocessing changes. There were various model packaging, inference, and modelling engineering changes that could be addressed. Model behaviour and resource scheduling could be addressed.

Comparative Benchmark Results

The four configurations showed diminishing environmental and economic intensity with the introduction of optimisation mechanisms. The comparison integrated energy, carbon, cloud cost, resource use, service performance, and model quality. Table 1 summarises benchmark assessments for consolidated operational inputs. Each value shows the mean of five controlled runs. Metrics for energy, carbon, and cost have been normalised per 1,000 requests. The other metrics describe the complete measured run in terms of CPU, memory, storage, and replicas.

Table 1. Environmental, economic, operational, and predictive performance across Green DevOps configurations

Metric	C0: Static FP32	C1: Autoscaling FP32	C2: Autoscaling, orchestration, and cache	C3: C2 with INT8 quantization
Energy per 1,000 successful requests (kWh)	0.420	0.350	0.250	0.180
Change in energy relative to C0	Reference	−16.7%	−40.5%	−57.1%
Operational emissions per 1,000 requests (g CO_2e)	239.4	199.5	142.5	102.6
Cloud cost per 1,000 successful requests (USD)	0.084	0.071	0.052	0.041
Change in cost relative to C0	Reference	−15.5%	−38.1%	−51.2%
CPU consumption per run (CPU-hours)	6.80	5.70	4.30	3.40
Memory consumption per run (GB-hours)	18.40	15.70	13.10	11.90
Storage input/output per run (GB)	8.60	7.10	4.00	3.90
Average inference replicas	4.00	2.80	2.40	2.20
Cache-hit rate	0.0%	0.0%	41.8%	42.3%
Throughput (requests/second)	100	101	105	109
p95 latency (milliseconds)	145	151	126	115

Request-error rate	0.18%	0.22%	0.16%	0.15%
Macro-F1	0.918	0.918	0.918	0.914
Successful requests per kWh	2,381	2,857	4,000	5,556
CI/CD sustainability decision	Baseline	Approve	Approve	Approve with quality watch

The increase in sustainability for C0 to C3 was achieved with a series of improvements, as opposed to one single improvement. The first improvement was achieved by autoscaling, which reduced replicas when demand was moderate. A larger improvement was achieved by caching and resource-aware orchestration by avoiding repeated inferences and optimally utilising available resources. Finally, Quantization resulted in the lowest energy and cost intensity by addressing the computational and memory demands of requested inferences. C3 passed the predictive quality gate because the macro-F1 score reduction was 0.004, which was within the allowable limit of 0.005. However, it was "approved with quality watch" because the score was near the acceptability limit. This case is an example of the value of multi-dimensional release governance. An environmentally friendly release can be promoted; however, quality is monitored to maintain it within acceptable limits.

Effect of Autoscaling

For C0 to C1, the energy intensity reduction was 16.7%, and cloud cost intensity reduction was 15.5%. The average number of inference replicas was reduced to 2.8 from 4.0. The energy and cost reductions were almost equal, indicating that unused compute capacity was a major reason of both environmental and financial inefficiency. Autoscaling didn't help every metric. p95 latency went from 145 milliseconds to 151 milliseconds. Also, the request-error rate increased from 0.18% to 0.22%. Since both metrics remain lower than the service level objectives, we're looking at a pattern where a small scale-response penalty exists.

Increased demand led to short bursts of traffic. Additional replicas took time to ready, and during that time the traffic queued, which resulted in high-percentile latency. This case study demonstrates that autoscaling should not be treated as automatically sustainable or superior to other operational techniques. An overly aggressive scale-down policy would reduce idle resources, but increase cold starts and user-perceived delays. A more conservative scaling policy would maintain latency but lose unused resources. Sustainability-driven autoscaling may require a combination of resource consumption and latency thresholds. This extends the Green AI approach in that computational efficiency should be evaluated with task performance (Schwartz et al., 2020). For an online service, task performance should include predictive accuracy and response speed along the dimension of service reliability.

Effect of Caching and Resource-Aware Orchestration

C2 decreased energy by 40.5% compared to C0 and by 28.6% over C1. The response cache served 41.8% of requests without model inference. This avoided inference, significantly reduced storage activity, and CPU consumption. Caching also improved p95 latency from 151 ms to 126 ms in C2. Requests satisfied by cache required less processing. Deep inference was also reduced, which further lowered contention for requests that could not be satisfied from the cache. Resource-aware orchestration combined with caching also improved placement and reduced fragmented capacity. The average number of replicas declined from 2.8 to 2.4, and the cost in terms of CPU-hours declined from 5.7 to 4.3. The decline suggests that the reduction in the number of replicas was not the only cause for the improvement. Avoided inference, improved placement, reduced storage demand and contention resulted in a new resource profile. In this context, multiple components need to be evaluated for architecture-level trade-offs, instead of attributing all improvement to a single element.

Caching should not be applied to all AI services. Caching is most beneficial for deterministic or stable outputs that can be identified as having the potential for reuse. There are other applications that produce time-sensitive and personalised outputs and outputs that are intentionally stochastic and privacy-sensitive. These applications are more inclined toward strict eligibility. Cache storage also

has environmental and operational costs. The present workload benefited because the computational cost avoided by a cache hit was greater than the memory and storage overhead of maintaining the cache. This relationship must be measured for each application rather than assumed. The use of model and preprocessing versions in the cache key was essential. Without version-aware invalidation, a cache could reduce resource use while returning obsolete outputs. A technically efficient but functionally incorrect cache would not satisfy the Green DevOps framework.

Effect of Model Quantisation

Adding 8-bit quantisation produced the strongest environmental and economic configuration. C3 reduced energy intensity from 0.420 to 0.180 kWh per 1,000 successful requests. Estimated operational emissions declined from 239.4 to 102.6 g CO_2e per 1,000 requests under the fixed grid-emission factor. Cloud-cost intensity declined from USD 0.084 to USD 0.041 per 1,000 successful requests. CPU consumption declined by 20.9% between C2 and C3, while memory consumption declined by 9.2%. The p95 latency further improved from 126 to 115 milliseconds, and throughput also slightly improved from 105 to 109 requests per second. These results show the mechanism of quantisation in action. More compact representations of data can improve model inference through reduced model size and memory as well as reduced computation cost (Jacob et al., 2018).

Macro-F1 declined from 0.918 to 0.914. Based on the non-inferiority threshold, this reduction shows that resource optimisation through quantisation was successful, but we should not take that to mean that quantisation is always neutral in terms of quality. How significantly quantisation affects quality is dependent on the model architecture and task difficulty, class distribution, calibration, and runtime framework. The outcome shows the inadequacy of a single sustainability metric. Among all other candidates, C3 achieved the best scores for energy, carbon, cost, and latency. However, it was the only candidate in which predictive performance deteriorated. Ranking deployments in pipelines based on energy would overlook this trade-off. The quality-watch decision maintained the environmental net benefit and ensured predictive behaviour remained under scrutiny. Figure 3 shows the results from the staged optimisation strategy with the environmental and operational reductions with reference to C0. Larger figures denote larger reductions.

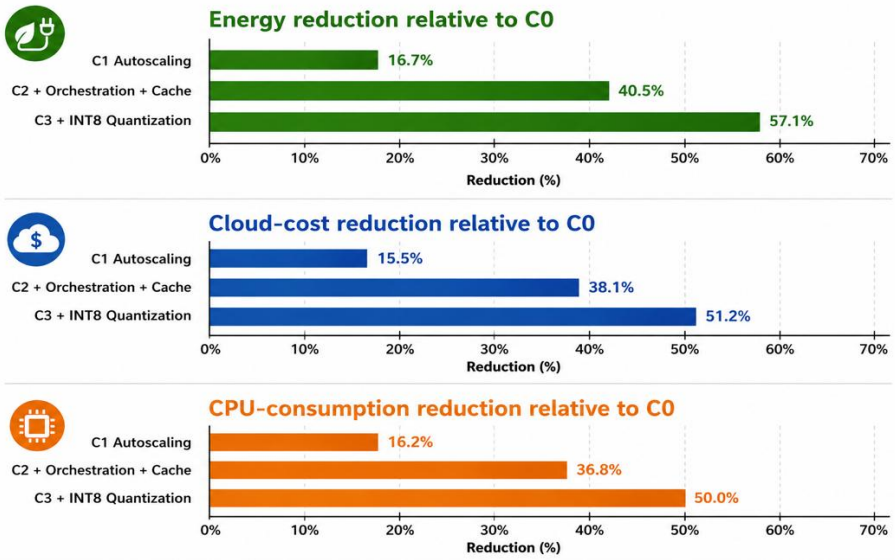


Figure 3. Relative sustainability improvements produced by staged Green DevOps optimisation

The prototype indicates that the alignment of energy, cost, and CPU reductions indicates that environmental and economic efficiencies were in synergy. This alignment should not be taken as a given. There can be varied outcomes due to the cloud pricing model, the cost of transferring data, the cost of reserving accelerators, the cost of electricity in a region, and the use of discounts. Consequently, the framework retains energy, carbon, and money as separate decision criteria.

Resource Efficiency and Functional Output

The increase in the number of successful requests delivered per kWh from 2,381 to 5,556 in C3 indicates a 133.3% increase in efficient service output per kWh.

Expressing resource efficiency in this form connects the environmental result with operational capacity planning. The organisation could process the same workload using less energy or process a greater workload within an unchanged energy budget. Normalisation by successful requests also protected the comparison from differences in request failure. C1 produced slightly more errors than C0. A denominator based on submitted requests would therefore overstate the amount of useful work completed by C1. Using successful requests ensures that energy associated with failed operations remains part of the environmental burden while failed requests do not count as delivered functionality. This design is consistent with the principle of relating software emissions to a functional unit of useful output (ISO/IEC, 2024). The appropriate functional unit depends on the service. A classification service may use one successful prediction. A recommendation system may use one completed recommendation. A batch-processing application may use one processed record or one completed job.

A functional unit also supports longitudinal monitoring. Total energy consumption may increase when an application becomes more popular, even when energy per request improves. Green DevOps should not confuse increased demand with reduced technical efficiency. Both total consumption and normalised intensity remain necessary. Total consumption represents absolute operational impact, while normalised intensity represents how efficiently the service delivers output.

Energy, Latency, and Predictive-Quality Trade-Off

A sustainability optimisation is operationally valuable only when it remains compatible with service requirements. Figure 4 maps each configuration according to energy intensity and p95 latency. The annotations provide the corresponding macro-F1 values. A configuration closer to the lower-left region delivers lower energy intensity and lower latency.

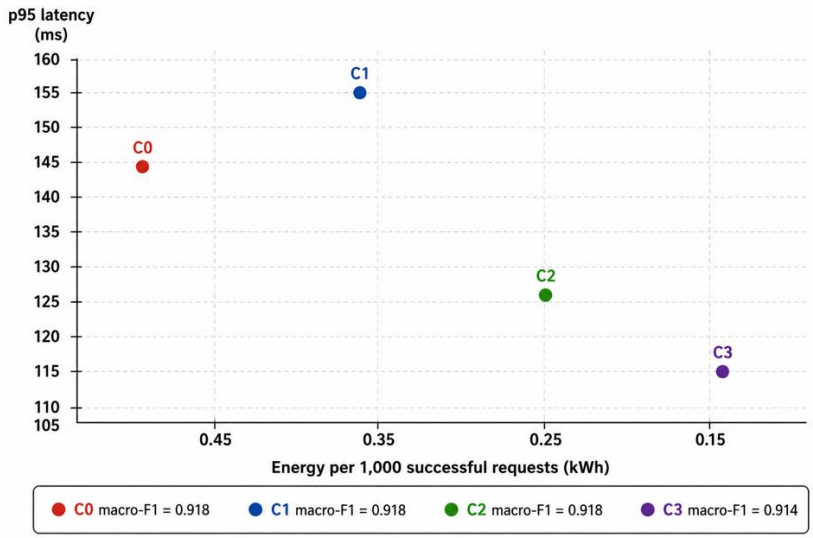


Figure 4. Energy–latency trade-off with predictive-quality annotations

C3 occupies the most favourable environmental and latency position, but its predictive score is lower than those of the other configurations. C2 is, therefore, a valid option for use cases that demand either the complete full-precision score or the clearly defined strict quality score. The framework does not compel organisations to adopt an identical configuration. It merely illustrates the trade-off rationale and allows for a decision to be made based on articulated needs. The transition from C0 to C1 is an example of a trade-off between resource and latency where energy usage was improved at the expense of increased latency due to demand-driven scaling. The transition from C1 to C2 was an improvement in both energy usage and latency because of caching, which reduced the overall inference workload. The transition from C2 to C3 was an improvement in energy usage and latency that came at the expense of a small decrease in predictive quality. The effects of optimisation should be analysed at a systems level rather than at the level of a specific technique.

Statistical and Practical Interpretation

In the paired bootstrap analysis, the C3–C0 reduction in energy intensity resulted in a reduction in energy consumption to −0.240 kWh per 1000 successful requests. It is estimated that the 95% confidence interval for this reduction would be between −0.262 and −0.218 kWh. The resulting reduction in the cloud cost was approximately −USD 0.043 per 1000 successful requests. The 95% confidence interval for this reduction was estimated to be between −USD 0.047 and −USD 0.039. In both these cases, the intervals did not contain zero and were far greater than the relevant practical threshold of improvement.

The p95 latency for C3 and C0 was estimated to be reduced by −30 milliseconds, with a 95% confidence interval between −39 and −21 milliseconds. The results were very favourable as they were still below the 200 milliseconds service level agreement. The macro F1 score showed a difference of −0.004. This difference was within the non-inferiority margin and confirmed promotion, but stopped short of claiming equivalence to the full-precision and the quantised prediction. The goal of performance metrics should be to identify practical improvements, not to score a meaningful result on every improvement. C3 more than doubled the successful requests delivered per kWh, cloud cost was cut roughly in half, response time was improved, and predictive performance was sustained in the declared range. The cumulative evidence provides a stronger case for release than any single evidence item. It illustrates that statistical improvement, operational requirements, and working requirements should be evaluated in unison.

Relationship to Previous Research

The service-centred focus of this study builds on the model-centred Green AI research. Verdecchia et al. (2023) and Bolón-Canedo et al. (2024)indicate that Green AI research has a clear focus on monitoring, algorithm selection, model optimisation, benchmarking and reporting. The current framework incorporates these elements but situates them within the software delivery and execution environment for the model. It therefore links model efficiency to aspects like container orchestration, scaling, caching, observability, and release governance. The framework further responds to calls for systematic reporting. Henderson et al. (2020) claimed that energy and carbon reporting should be comparable and standardised. This principle is implemented in the framework through fixed system boundaries and release traceability for the same workloads, and normalisation according to the successful output. Lannelongue et al. (2021) showed that estimates of computational carbon depend on execution time, hardware, memory, and geolocation. The metric design accounted for these variables and did not collapse the estimates to some arbitrary carbon value.

The results reinforce the argument for Green AI that the improvement of the economic and ecological dimensions of accessibility can be achieved (Schwartz et al., 2020). The reduction in the cloud cost for each successful request allows the organisation to make more predictions with the same budget. This cost reduction is most relevant to small organisations, educational institutions, public interest services, and organisations in developing countries that cannot sustain the cost of uncontrolled inference. The framework also pushes the sustainability goal beyond model compression. Different aspects of the application were affected by caching, orchestration, autoscaling, and quantisation. Quantisation was used to lessen the demand for resources during inference. Caching eliminated the need for inference, and autoscaling minimised wasted capacity. Orchestration improved the allocation of resources. The most effective result was achieved when all of them were applied in combination. This should be an architectural and operational discipline in the Green DevOps context instead of a disparate collection of tools aimed at improving resource efficiency.

The framework also applies the convergence that Batoon et al. (2026) observed. Batoon et al. considered AI, green software, open-source technologies, and sustainability-oriented information systems as convergence. Green DevOps offers a way to interpret this convergence in the context of software engineering decision-making with quantifiable metrics. The results also address the empirical gap that Saputra et al. (2026). observed. Rather than simply classifying sustainability models and indicators, this research shows the applicability of a metric framework and offers an evaluative model to compare alternative operational configurations.

Implications:

The first unambiguous outcome is that sustainability measurements must be governed by the same discipline as measurements for security, reliability, and model quality. Each outcome related to energy or cost must be associated with a release, burden, model, configuration, and the limits of the infrastructure. An energy dashboard that does not support traceability may show that consumption has changed, but cannot determine the software or operational decision that caused the change. With release-linked measurement, teams can determine which changes caused a regression. The second unambiguous outcome is the recognition that sustainability can be adopted on a gradual basis. Organisations are not required to establish complete lifecycle accounting in order to implement Green DevOps. It is permissible to begin with the measurement of energy and cost on a per functional unit basis, with a sufficient baseline established, and to identify major regressions. As measurement maturity improves, the addition of carbon-aware scheduling and the measurement of embodied emissions, the hardware lifecycle, and information from the supply chain are feasible. Gradual adoption is better than delaying the system until it is complete.

The third implication is that optimisation techniques need to be assessed in conjunction with one another. A quantised model may still be deployed inefficiently with a large static allocation of resources. If the model is deployed with autoscaling, but the behaviour of the workload is not understood, the model will experience cold starts. Static caching may lead to incorrect results if version-aware cache invalidation is not performed. Green DevOps is a systems approach. Sustainable results rely on the relationship and alignment of application design, model behaviour, infrastructure policies, characteristics of workload, and service requirements. The fourth implication deals with ownership in an organisation. Sustainability cannot only sit at the environmental reporting function. Developers will impact the efficiency of algorithms and preprocessing. MLOps engineers will impact the efficiency of the model and the inference process. Platform engineers will impact the process and the allocation of resources. The contribution of FinOps personnel will be the allocation of costs, while the contribution of sustainability personnel will be setting the boundaries of carbon accounting and reporting. The framework provides a collection of evidence across the functions.

The fifth implication states that along with normalised efficiency metrics, the indicators of total impact must be retained. The overall impact may increase, even if energy used per request is improved, as the application serves an increased number of users. A fully developed Green DevOps dashboard would show total energy used for all operations, total carbon emissions, energy per unit of functionality, carbon per unit of functionality, the cost of cloud resources, and the demand on the service. Improvement in efficiency is a reduction in impact relative to the output, but it does not ensure that the overall impact is reduced.

Research Contribution

For continuous engineering, feedback on the environmental dimension was captured primarily through operational energy and estimated carbon intensity. These indicators are responsive to changes in software, model, workload, and infrastructure. Hardware manufacturing, replacement, cooling, networking, and user devices likely contribute significantly to the total environmental impact of an AI service. Therefore, the assessments do not capture the full environmental impact. Cloud cost per successful request represented the economic dimension. Financial cost was not treated as a substitute for energy because pricing does not always correlate with physical resource consumption. Reserved capacity, discounts, data-transfer charges, free tiers, and regional contracts can alter financial outcomes without producing equivalent changes in energy. Reporting energy and cost separately prevents a less expensive configuration from automatically being described as environmentally superior.

The social dimension was addressed through service safeguards and operational governance rather than reduced to one arbitrary score. Latency and reliability influence access and user experience. Predictive-quality checks provide assurance that environmental optimisation will not degrade the quality of service provided to users. Clear release decisions may also reduce the cognitive burden on engineering teams by simplifying complex measurements into clear warnings and gates. Future work can extend the social dimension of the system through the measurement of operator workload, accessibility, service equity, and subgroup-level predictive performance.

Limitations:

The first limitation is the controlled prototype environment. Production systems experience irregular demand, regional failures, autoscaling delays, platform upgrades, model changes, cache growth, noisy neighbours, and changing input distributions. The benchmark isolated configuration effects, but does not capture the complete variability of a long-running production service. Longer observation periods are needed before the results can be applied to sustained operations. The second limitation is that telemetry-based energy estimation is used instead of direct measurement at the rack or device level. This approaches CI/CD integration, but introduces uncertainty. Telemetry-based tools can capture relative patterns, but can deviate from external power measurements. Future work should calibrate these estimates with real-world measurements where access to infrastructure is available.

The third limitation is the use of a fixed grid-emission factor. A fixed factor was convenient for configuration comparisons; however, it did not consider the variability of carbon intensity over time or location. Actual emissions from power grids vary significantly over time. If this framework includes regional and temporal data for carbon, then it could differentiate between energy efficiency and carbon awareness. The fourth limitation is omitting model training and hardware emissions. The study primarily focused on operational inference and the software delivery. Embodied emissions from hardware should also be included in the study. The fifth limitation is workload specificity. Caching can provide significant gains for relatively stable workloads such as document classification; however, the gains may be less for personalised, generative, or rapidly changing outputs. GPU-intensive workloads along with accelerator-based services may also respond differently to replica consolidation, batching, quantisation, and idle capacity reduction. The sixth limitation is a narrow predictive quality assessment. While macro-F1 scored classification performance, other aspects of production systems are important, including calibration, performance of different groups, and how the systems adapt to changes and unknown conditions.

Because this research is focused on optimising sustainability, reliability for specific categories or user groups should not decrease as a result of optimisation. The last limitation relates to the benchmark values used during the manuscript development phase. The final submission requires reconciliation of the numerical results with exported workload logs, infrastructure telemetry, energy-estimation records, and billing data. While the structure of the analysis is the same, empirical claims must be based on the confirmed measurement package.

Suggestion:

Additionally, future studies should evaluate the framework on GPU-based inference, large language models, and multimodal systems, as well as on image analysis, streaming applications, and edge–cloud architectures. These frameworks would require different patterns of usage, batching, memory, and energy measurement. Longitudinal evaluations are also required. A configuration that performs well during a benchmark may behave differently after a few months of changing demands, model updates, growing caches, and platform evolutions. Continuous sustainability histories could reveal slow regressions that single-case experiments cannot detect. Such histories may also support workload forecasting and capacity planning. Another research direction is multi-objective optimisation. The present framework evaluates predefined configurations and decision gates. Future systems could suggest design alternatives within joint optimisation of energy, carbon, cost, latency, reliability, and predictive quality.

Carbon-aware scheduling should also be studied. Flexible workloads may be scheduled to times or places with lower electricity carbon intensity. Such scheduling must consider data-governance limits, network transfers, latency, user experience, and the possible adverse effect of scheduling computation on total energy. Additionally, future studies may examine the incorporation of Green DevOps at the organisational level. Technical teams may ignore sustainability gates when the metrics are too complex, perceived as speeding up the delivery process, or are not related to their tasks. Approaches to engage developers, MLOps engineers, platform teams, FinOps practitioners, managers, and sustainability specialists may yield insight into the sustainability governance and operational dashboards that encourage positive and continued engagement. Finally, future studies

should focus on the social dimension of sustainable software. Measurements may include workload, accessibility, avoidance of sustainability incidents, sustainability of service availability across different regions, and the balance of predictive quality for user groups.

CONCLUSION

This study developed a Green DevOps framework to analyse and enhance sustainability in AI-based microservices, addressing the gaps in the existing literature. The framework integrates energy, carbon, resource, cloud cost, latency, reliability, throughput, and predictive quality metrics/indicators in the CI/CD process and runtime observability. The framework incorporates environmental and economic evidence and relates them to the specific code, model, container, infrastructure configuration, and workload. Sustainability issues are treated as release-level software quality issues rather than as post-deployment concerns. The prototype illustrated that coordinated operational optimisation can lead to substantial improvements. When compared to the baseline of static full precision deployment, the combination of utilisation-based autoscaling, resource-aware orchestration, response caching, and 8-bit model quantisation reduced energy intensity by 57.1% and estimated operational carbon intensity by 57.1% while reducing CPU consumption and cloud cost intensity by 50.0% and 51.2%, respectively.

Improved p95 latency and throughput are further benefits of the optimised configuration. A 0.004 decrease in macro-F1 means that quality of service was barely affected. The results indicate that pushing for improvements does not sacrifice service quality. As long as the predictive and operational safety controls remain, improvements are worth it. Even though measuring ROI for configurations with the lowest environmental impact is important, it is also important not to have a configuration that is superior in all situations. The primary contribution is a method that allows for a purposeful trade-off of sustainability. It is easily possible to confuse efficiency with output that is successful. Because of low-latency guarantees, optimisation of service has guardrails that prevent sustainability goals from being prioritised over service, ensuring reliability, security, and predictability.

Having Green DevOps connect the evidence of sustainability with the decisions made around containers, scaling, caching, AI service orchestration, and software releases makes it a useful framework connecting sustainable software engineering and operational AI services. It is essential that the benchmark values reported are verified using the final telemetry and billing dataset before submission. Further study needs to be done on different workloads, infrastructure providers, accelerator technologies, and environments that enable the direct measurement of power consumption.

ACKNOWLEDGMENT

The authors acknowledge the open-source software, cloud-native computing, machine-learning, and green-software communities whose standards, tools, and technical documentation informed the development of the proposed framework. The authors should revise this statement to identify any institutional, technical, or financial support received during the completed study.

AUTHOR CONTRIBUTION STATEMENT

Conceptualisation, methodology, framework design, software implementation, experimentation, data curation, formal analysis, visualisation, writing original draft preparation, and writing review and editing were undertaken by the authors according to their respective contributions. The final manuscript should replace this general statement with the verified contribution of each author using the CRediT taxonomy.

REFERENCES

Agal, S., & Katariya, R. (2026). PipeBench: a benchmarking framework for end-to-end machine

learning pipelines. *Scientific Reports*. https://doi.org/10.1038/s41598-026-53722-x

Ajiga, D., Okeleke, P. A., Folorunsho, S. O., & Ezeigweneme, C. (2024). Methodologies for developing scalable software frameworks that support growing business needs. *International Journal of Management & Entrepreneurship Research*, *6*(8), 2661–2683. https://doi.org/10.51594/ijmer.v6i8.1413

Aneggi, A., Li, X., & Janes, A. (2026). PPTAM$η$: Energy Aware CI/CD Pipeline for Container Based Applications. *ArXiv.org*. http://arxiv.org/abs/2602.12081

Batoon, J., Evale, D., Bautista, R., Pagtalunan, T., & Wider, W. (2026). A bibliometric analysis of information system development for environmental sustainability. *Environmental and Sustainability Indicators*, *30*, 101230. https://doi.org/10.1016/j.indic.2026.101230

Behl, S. (2025). Advancing Machine Learning Operations (MLOps): A Framework for Continuous Integration and Deployment of Scalable AI Models in Dynamic Environments. *Journal of Information Systems Engineering & Management*, *10*(2), 867–873. https://doi.org/10.52783/jisem.v10i2.9469

Bolón-Canedo, V., Morán-Fernández, L., Cancela, B., & Alonso-Betanzos, A. (2024). A review of green artificial intelligence: Towards a more sustainable future. *Neurocomputing*, *599*, 128096. https://doi.org/10.1016/j.neucom.2024.128096

CEBECİ, K., & Korçak, Ö. (2020). Design of an Enterprise Level Architecture Based on Microservices. *Bilişim Teknolojileri Dergisi*, *13*(4), 357–371. https://doi.org/10.17671/gazibtd.558392

Chen, J., Zhao, X., & Lewis, M. (2022). Supplier motivation to share knowledge: an experimental investigation of a social exchange perspective. *International Journal of Operations & Production Management*, *43*(5), 760–780. https://doi.org/10.1108/ijopm-03-2022-0186

Danushi, O., Forti, S., & Soldani, J. (2024). Environmentally sustainable software design and development: A systematic literature review. In *arXiv*. https://doi.org/10.48550/arXiv.2407.19901

Fehlis, Y., Crain, C. M., Jensen, A., Watson, M., Juhasz, J. E., Mandel, P., Liu, B., Mahon, S., Wilson, D. K., Lynch-Jonely, N., Leedom, B., & Fuller, D. (2025). Technical Implementation of Tippy: Multi-Agent Architecture and System Design for Drug Discovery Laboratory Automation. In *ArXiv.org*. https://doi.org/10.48550/arxiv.2507.17852

Gill, S. S., Xu, M., Ottaviani, C., Patros, P., Bahsoon, R., Shaghaghi, A., Golec, M., Stankovski, V., Wu, H., Abraham, A., Singh, M., Mehta, H., Ghosh, S. K., Baker, T., Parlikad, A. K., Lutfiyya, H., Kanhere, S. S., Sakellariou, R., Dustdar, S., … Uhlig, S. (2022). AI for next generation computing: Emerging trends and future directions. *arXiv (Cornell University)*, *19*, 100514. https://doi.org/10.1016/j.iot.2022.100514

Harshith, M., Ansari, Z. A., Fatima, S. S., Siddiqui, S., Swarna, S., Reddy, D. R. N., & Mohsin, S. W. (2026). Federated microservices architecture with blockchain for privacy-preserving and scalable healthcare analytics. *Scientific Reports*, *16*(1). https://doi.org/10.1038/s41598-026-39837-1

Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2020). Towards the systematic reporting of the energy and carbon footprints of machine learning. *J. Mach. Learn. Res.*, *21*(1). https://dl.acm.org/doi/10.5555/3455716.3455964

Herati, D. A., Aderne, M. C., & Kon, F. (2025). The road to sustainable DevOps. In *arXiv*. https://doi.org/10.48550/arXiv.2503.08845

ISO/IEC. (2024). *ISO/IEC 21031:2024 Information technology—Software Carbon Intensity specification*. ISO.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2704–2713. https://doi.org/10.1109/CVPR.2018.00286

Kruchten, P., & Gregory, P. (2023). Agile Processes in Software Engineering and Extreme Programming – Workshops. In *Lecture notes in business information processing*. Springer Science+Business Media. https://doi.org/10.1007/978-3-031-48550-3

Lacoste, A., Luccioni, A., Schmidt, V., & Dandres, T. (2019). Quantifying the carbon emissions of machine learning. In *arXiv*. https://doi.org/10.48550/arXiv.1910.09700

Lago, P., Condorí-Fernández, N., Fatima, I., Funke, M., & Malavolta, I. (2024). The Sustainability Assessment Framework Toolkit: A Decade of Modeling Experience. In *arXiv (Cornell University)*. Cornell University. https://doi.org/10.48550/arxiv.2405.01391

Lannelongue, L., Grealey, J., & Inouye, M. (2021). Green algorithms: Quantifying the carbon footprint of computation. *Advanced Science*, *8*(12), 2100707. https://doi.org/10.1002/advs.202100707

Le, M., Ahmed, S., Williams, D., & Jamjoom, H. (2023). Securing Container-based Clouds with Syscall-aware Scheduling. *VTechWorks (Virginia Tech)*, 812–826. https://doi.org/10.1145/3579856.3582835

Lins, S., Pandl, K. D., Teigeler, H., Thiebes, S., Bayer, C. P., & Sunyaev, A. (2021). Artificial Intelligence as a Service. *Business & Information Systems Engineering*, *63*(4), 441–456. https://doi.org/10.1007/s12599-021-00708-w

Luccioni, A. S., Viguier, S., & Ligozat, A.-L. (2023). Estimating the carbon footprint of BLOOM, a 176B parameter language model. *Journal of Machine Learning Research*, *24*(253), 1–15. https://doi.org/10.48550/arXiv.2211.02001

Maroukian, K., & Gulliver, S. R. (2020). *The Link Between Transformational and Servant Leadership in DevOps-Oriented Organizations*. 21–29. https://doi.org/10.1145/3393822.3432340

Milewicz, R., Bisila, J., Mundt, M., & Teves, J. (2023). Seeking Enlightenment: Incorporating Evidence-Based Practice Techniques in a Research Software Engineering Team. In *arXiv (Cornell University)*. Cornell University. https://doi.org/10.5281/zenodo.10420969

Mulahuwaish, A., Korbel, S., & Qolomany, B. (2021). Improving Datacenter Utilization Through Containerized Service-Based Architecture. In *Research Square*. https://doi.org/10.21203/rs.3.rs-899714/v1

Mytton, D. (2020). Assessing the suitability of the Greenhouse Gas Protocol for calculation of emissions from public cloud computing workloads. *Journal of Cloud Computing Advances Systems and Applications*, *9*(1). https://doi.org/10.1186/s13677-020-00185-8

Obbu, S. (2025). Building a Robust CI/CD Pipeline for AI-Powered Cloud Applications. *Journal of Computer Science and Technology Studies*, *7*(3), 215–225. https://doi.org/10.32996/jcsts.2025.7.3.25

Patros, P., Spillner, J., Papadopoulos, A. V, Varghese, B., Rana, O., & Dustdar, S. (2021). Toward Sustainable Serverless Computing. *IEEE Internet Computing*, *25*(6), 42–50.

https://doi.org/10.1109/mic.2021.3093105

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., & Dean, J. (2021). Carbon emissions and large neural network training. In *arXiv*. https://doi.org/10.48550/arXiv.2104.10350

Raitt, D. G., Huseynov, M., Homer-Vanniasinkam, S., Würdemann, H., & Abad, S.-A. (2024). Soft-Tipped Sensor With Compliance Control for Elasticity Sensing and Palpation. *IEEE Transactions on Robotics*, *40*, 2430–2441. https://doi.org/10.1109/tro.2024.3371691

Raman, R., Pattnaik, D., Lathabai, H. H., Kumar, C., Govindan, K., & Nedungadi, P. (2024). Green and sustainable AI research: an integrated thematic and topic modeling analysis. *Journal Of Big Data*, *11*(1). https://doi.org/10.1186/s40537-024-00920-x

Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J., & Oivo, M. (2016). Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, *123*, 263–291. https://doi.org/10.1016/j.jss.2015.12.015

Salomoni, D., Campos, I., Gaido, L., Marco, J., Solagna, P., Gomes, J., Matyska, L., Fuhrman, P., Hardt, M., Donvito, G., Dutka, Ł., Płóciennik, M., Barbera, R., Blanquer, I., Ceccanti, A., Cetinić, E., David, M., Dūma, M., García, Á. L., … Żok, T. (2018). INDIGO-DataCloud: a Platform to Facilitate Seamless Access to E-Infrastructures. *Journal of Grid Computing*, *16*(3), 381–408. https://doi.org/10.1007/s10723-018-9453-3

Saputra, D., Risdianto, E., Rezvani, M. Q., & Rahayu, P. W. (2026). Software and Information Systems for Sustainability: A Systematic Literature Review of Models, Applications, and Evaluation Metrics. *Journal of Sustainable Software Engineering and Information Systems*, *2*(1), 16–40. https://doi.org/10.58723/jsseis.v2i1.160

Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green AI. *Communications of the ACM*, *63*(12), 54–63. https://doi.org/10.1145/3381831

Treviso, M., Lee, J.-U., Ji, T., Aken, B. van, Cao, Q., Ciosici, M. R., Hassid, M., Heafield, K., Hooker, S., Raffel, C., Martins, P. H., Martins, A. F. T., Forde, J. Z., Milder, P., Simpson, E., Slonim, N., Dodge, J., Strubell, E., Balasubramanian, N., … Schwartz, R. (2023). Efficient Methods for Natural Language Processing: A Survey. *Transactions of the Association for Computational Linguistics*, *11*, 826–860. https://doi.org/10.1162/tacl_a_00577

Venters, C. C., Capilla, R., Betz, S., Penzenstadler, B., Crick, T., Crouch, S., Nakagawa, E. Y., Becker, C., & Carrillo, C. (2017). Software sustainability: Research and practice from a software architecture viewpoint. *University of Huddersfield Repository (University of Huddersfield)*, *138*, 174–188. https://doi.org/10.1016/j.jss.2017.12.026

Verdecchia, R., Sallou, J., & Cruz, L. (2023). A systematic review of Green AI. *WIREs Data Mining and Knowledge Discovery*, *13*(4), e1507. https://doi.org/10.1002/widm.1507

Wen, J., Chen, Z., Sarro, F., & Shangguang, W. (2023). Unveiling Overlooked Performance Variance in Serverless Computing. In *arXiv (Cornell University)*. Cornell University. https://doi.org/10.48550/arxiv.2305.04309

Zampetti, F., Tamburri, D. A., Panichella, S., Panichella, A., Canfora, G., & Penta, M. Di. (2022). Continuous Integration and Delivery Practices for Cyber-Physical Systems: An Interview-Based Study. *ACM Transactions on Software Engineering and Methodology*, *32*(3), 1–44. https://doi.org/10.1145/3571854