Estimating the Carbon Footprint of Serverless Functions on a Public Cloud Platform

Hanan Awwad, Changyuan Lin, Rabab Ward, Mohammad Shahrad
{hanan,chlin,rababw,mshahrad}@ece.ubc.ca
University of British Columbia
Vancouver, BC, Canada

Abstract

As the carbon footprint of cloud data centers grows rapidly, sustainability has become an increasing concern for practitioners. Understanding the carbon emissions of cloud workloads and identifying strategies to reduce them is critical. In this paper, we model and extensively analyze the carbon emissions of functions executed on a public serverless platform using available telemetry, offering new insights into the relationship between carbon emissions and traditional metrics of cost and performance. We explore various factors affecting carbon emissions, including host region, architecture, cold starts, application resource composition, and input-sensitivity. Based on our findings, we propose future optimization opportunities and research directions. Our work aims to empower developers to make more sustainable decisions when configuring or optimizing their applications.

CCS Concepts

• Social and professional topics → Sustainability; **• Software and its engineering → Cloud computing**.

Keywords

Sustainability, Serverless Computing, Carbon Modeling

1 Introduction

Cloud data centers have emerged as significant contributors to global greenhouse gas (GHG) emissions within the Information and Communication Technology (ICT) sector [22]. The serverless computing model, which has gained significant traction over the past few years, offers potential environmental benefits through dynamic resource allocation, enabled by autoscaling of function or application sandboxes (e.g., containers, pods). Minimizing idle resource waste helps reduce carbon emissions. This capability is particularly critical given that most real-world applications experience fluctuating traffic patterns. However, the environmental efficacy of this paradigm remains contingent upon the efficiency of the building blocks of scaling, i.e., how each sandbox is configured. Prior research has demonstrated that suboptimal configuration of serverless functions can incur substantial operational costs and performance degradation [34, 49, 55], suggesting a probable correlation with increased carbon footprints. Consequently, the sustainability benefits of serverless computing hinge on understanding and optimizing for efficient sandbox configurations.

Understanding the key factors driving carbon emissions in serverless computing is critical for empowering developers to prioritize sustainability and optimize their applications. While cloud providers have introduced initiatives to reduce data center emissions, these efforts often remain disconnected from the configuration decisions developers must make. This raises a pivotal question: Are developers equipped with the necessary insights to make informed, carbon-aware decisions? Three primary barriers currently hinder their ability to do so. First, within virtualized serverless sandboxes, developers lack access to granular infrastructure-level energy metrics provided by hardware interfaces such as Intel RAPL. This leaves them blind to the direct environmental impact of their code. Second, without visibility into co-located workloads on shared servers, they cannot accurately attribute static power consumption in multi-tenant environments [50, 65]. Finally, the emissions data provided to developers by providers arrives via coarse-grained reports at the end of billing cycles [19, 27, 28, 61]—far too late to inform real-time optimizations. These limitations underscore a gap between sustainability goals and actionable developer tools, stifling progress toward greener serverless architectures.

The goal of this study is to investigate how fine-grained telemetry accessible to developers combined with published power and carbon models can reveal opportunities to improve the carbon efficiency of serverless functions through configuration adjustments. Departing from prior methodologies that rely on controlled local hardware environments [65] or bare-metal instances with fixed resource profiles [58], we instead explore what actionable insights can be derived solely from existing serverless logs and metrics available to developers. By narrowing our carbon estimation scope to the execution phase of serverless functions—the portion developers are directly charged for and can influence through optimizations—we examine how well the existing pricing models incentivize emission reduction efforts by developers. While this approach cannot eliminate the need for providers to implement high-quality, real-time carbon APIs, it nevertheless establishes a framework to harmonize current cost-driven optimization practices with sustainability objectives. With this goal, this paper makes the following contributions:

- We build operational and embodied carbon models for functions executed on AWS Lambda, a popular serverless platform.
- We show which readily-available metrics from AWS CloudWatch Lambda Insights can be used to feed these carbon models.
- We characterize various sources of emissions and compare carbon emissions to classic metrics such as performance and cost.

SESAME' 25, March 30-April 3 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1557-0/2025/03
https://doi.org/10.1145/3721465.3721863

- Building on our characterization results, we highlight several challenges and propose future research directions.

2 Carbon Model

This section explains the fundamental concepts of carbon emissions in cloud systems and outlines our approach to modeling its various components for serverless functions.

2.1 Basics

The Greenhouse Gas Protocol (GHG protocol) [9] is a widely adopted framework for carbon footprint assessment. Under the GHG protocol, the carbon emission of cloud data centers stems from direct emission (Scope 1), purchased energy (Scope 2), and carbon embodied in hardware and infrastructure (Scope 3) [51]. Scopes 1 and 2 generally involve the operational carbon of data centers, while the direct emission (e.g., on-site power generators and employees) from the data center is usually negligible [3]. The Scope 2 carbon can be modeled based on the energy consumption, power usage effectiveness (PUE) of the data center, and the carbon intensity of the underlying power grid. Scope 3 emission mainly includes the embodied carbon associated with hardware and infrastructure manufacturing, transportation, maintenance, and replacement, as well as the hardware life cycle policy.

The total carbon footprint of serverless functions consists of operational (C_{op}) and embodied carbon (C_{em}).

$$C_{total} = C_{op} + C_{em} \tag{1}$$

Operational carbon refers to the carbon emissions generated from the energy consumed during the execution of serverless functions. This includes the energy required to power the resources allocated to the function during its execution. Embodied carbon is determined by the emissions associated with the hardware on which an application runs. As discussed in §2.3, it accounts for the carbon emissions from manufacturing, transportation, installation, maintenance, and disposal of the resources. This evaluation considers life cycle analysis (LCA) from the production to disposal of devices used in cloud infrastructure. Serverless functions contribute to embodied carbon proportional to the hardware capacity they use. Consideration of embodied carbon as one of the primary differentiating factors for carbon vs. energy optimization. Prior work has demonstrated that optimizing for carbon is different from optimizing for energy [68, 76].

In many serverless systems, developers control function configurations (to various degrees). For example, in AWS Lambda, developers set memory configurations of functions and CPU is allocated proportionally [7]. Since we study AWS Lambda in this paper, we adhere to the same resource allocation model and account for carbon emissions from both allocated and used resources. Throughout the rest of the paper, we denote the function's memory configuration as m. At 1,769 MB of memory, there is exactly one vCPU allocated to the Lambda function [7].

2.2 Modeling Operational Carbon

The operational carbon is influenced by 1) the energy efficiency of the data center, a.k.a. power usage effectiveness (PUE), 2) the carbon intensity of the electrical grid (I_{grid}) [70], and 3) the energy consumed by the resources used during execution. The previous work [67, 72] indicates that the main contributors to the system's energy are the memory, CPU, network, and storage resources.

CPU Vendor, Arch., and Freq.	CPU Model (Inferred)	Occurrence Frequency	Idle Power (W)*	Active Power (W)**	Ref.
Intel Haswell 2.50 GHz	Xeon E5-2680 v3	83.17%	44	125	[1, 10]
Intel Haswell 2.90 GHz	Xeon E5-2666 v3[1]	5.84%	32[2]	135	[1, 14]
Intel Haswell 3.00 GHz	Xeon E5-1660 v3	10.40%	34	140	[1, 10]
AMD EPYC 2.65 GHz	EPYC 7R13[1]	0.59%	82[3]	225	[13, 66]

*The C1E-state power specification is used for the idle power of Intel CPUs. **TDP power is used.
[1]The server CPU is the OEM version with limited datasheet information. The box version of the CPU (i.e., Intel Xeon E5-2667 v3[2] and AMD EPYC 7643[3]) with a comparable number of cores, base frequency, and TDP is used to estimate idle power.

Table 1: The CPU models for serving function requests, prevalence in percentages, and their energy metrics.

$$C_{op} = (E_{mem} + E_{cpu} + E_{network} + E_{storage}) \times PUE \times I_{grid} \tag{2}$$

2.2.1 Memory. The energy usage of memory is influenced by both its active power (P_{mem}^{high}) and the idle power (P_{mem}^{low}) consumption throughout the execution period.

$$P_{mem} = P_{mem}^{high} \times m_{used} + P_{mem}^{low} \times (m_{alloc} - m_{used}) \tag{3}$$

We take the average of the memory power consumption figures collected for idle and active memory in AWS [35]; 3.26e-4 kW/GB and 8.38e-4 kW/GB for P_{mem}^{low} and P_{mem}^{high}, respectively. Multiplying the power by the duration of function execution results in the total energy consumed.

2.2.2 CPU. The CPU energy consumption, denoted as E_{cpu}, is determined by the number of allocated CPU cores n_{cpu}, average per-core CPU power P_{cpu}, and function execution duration d:

$$E_{cpu} = P_{cpu} \times n_{cpu} \times d, \tag{4}$$

where P_{cpu} can be derived from a linear utilization-based power model [31] that is formulated as

$$P_{cpu} = P_{cpu}^{idle} + u \times \left(P_{cpu}^{act} - P_{cpu}^{idle}\right) \tag{5}$$

Here, u is the average CPU utilization rate throughout function execution (i.e., the ratio of the CPU time consumed by the function to the product of d and the total number of CPU cores), P_{cpu}^{idle} is the idle (baseline) CPU power, and P_{cpu}^{act} is the active CPU power under full utilization. In order to obtain the CPU power metrics (i.e., idle and active power), we read the CPU information from `/proc/cpuinfo`. We inferred the CPU model based on the CPU vendor, microarchitecture, and frequency reported by `cpuinfo`, AWS documentation [4, 5], and CPU specifications [10] and gathered power metrics from CPU datasheets and research report [1, 66]. Table 1 presents the reported CPU information, inferred CPU models, frequency of occurrence (out of 2,020 function invocations), and their corresponding power metrics.

2.2.3 Network. The energy needed for data transmission to and from Lambda functions can be estimated by considering the amount of data being transferred (S). There is a great deal of uncertainty in the existing network energy models [20, 23, 52]. We use the E_{trans} of 0.001 kWh/GB in this paper, which appears to be on the lower end of estimates for 2024 [37]. The energy consumption associated with the transmission of S gigabytes of data is calculated as

$$E_{network} = E_{trans} \times S \tag{6}$$

2.2.4 Storage. The `/tmp` file system in AWS Lambda offers 512 MB of ephemeral storage attached to each function, by default (extensible up to 10 GB). Solid state drives (SSDs) are more commonly used in cloud system data centers than traditional hard disk drives (HDDs) [48, 71]. We use 1.2e-3 W/GB as the unit power consumption for SSD servers [70]. Multiplying the baseline power (P_{ssd}) for SSDs by the amount of data stored (D) over execution time d yields the total energy consumption attributed to ephemeral data storage.

$$E_{storage} = P_{ssd} \times d \times D \tag{7}$$

We exclude the carbon impact of external storage (e.g., attached volumes and S3 buckets) since their cost and emissions are separately measured and reported by the respective storage services (e.g., Amazon S3 and EBS).

2.2.5 PUE. We use a power usage effectiveness (PUE) of 1.11, which represents the average value within the range of 1.07 to 1.15 as reported by AWS [18].

2.2.6 Carbon Intensity. Carbon intensity can vary over time, daily or seasonally. In this paper, we use the average carbon intensity values from 2024 reported from electric grids hosting four public AWS regions in North America. We use historical datasets provided by the Electricity Maps [53] for this purpose. The selected regions demonstrate a spectrum of carbon intensity levels. The *us-east-1* region had the highest annual average is with 392 gCO2e/kWh, while *ca-central-1* records the lowest at 35 gCO2e/kWh. Additionally, *us-west-1* and *us-west-2* report carbon intensities at 272 gCO2eq/kWh and 195 gCO2e/kWh, respectively. The reader should note that the actual carbon intensity of data centers may vary, as data centers may have energy storage [15], thermal energy harvesting [77], etc. These do not affect the trends reported in this work, however.

2.3 Modeling Embodied Carbon

The embodied carbon attributed to the function execution is mainly determined by the allocated computing resources (e.g., vCPUs and memory size) and the embodied carbon of the hardware providing the resource [50]. For a given serverless function f with a set of allocated resources R, we can formulate the embodied carbon as

$$C_{em} = \sum_{r \in R} c_{em}(r) \times d \times ALC\,(f, r), \tag{8}$$

where $c_{em}(r)$ is the per-unit-and-duration lifetime embodied carbon of the hardware associated with the resource r, d is the function execution duration, and $ALC(f, r)$ is the allocated size of resource r to f.

We consider a server lifespan of six years, as reported by AWS in February 2024 [6]. We leverage Datavizta [8], a publicly available tool for assessing ICT/digital environmental impacts, to obtain the embodied carbon of CPUs and memory. The per-vCPU embodied carbon of the four CPU models listed from top to bottom in Table 1 are 825 gCO2eq, 860 gCO2eq, 1115.63 gCO2eq, and 312.5 gCO2eq, respectively. The per-GB embodied carbon of memory is 1796.88 gCO2eq. We adopt the per-GB embodied carbon of 160 gCO2eq for storage [69]. For example, the storage embodied carbon of a function with 1 GB of storage and execution duration of 1 s can be calculated as

$$C_{em}^{storage} = \frac{160^{gCO2eq}/_{GB}}{6 \times 365 \times 86400s} \times 1s \times 1GB = 8.46 \times 10^{-7} gCO2eq$$

To the best of our knowledge, there is no available data on the embodied carbon associated with network data transfer. So, we do not consider the embodied carbon of the network in our analysis.

The embodied carbon model we used mainly focuses on the manufacturing carbon associated with main components such as CPU, memory, and storage, with the scope limited to serverless functions. These components are externally measurable. Our model excludes broader embodied carbon factors, such as manufacturing carbon for other devices (e.g., motherboards and power supply units), transportation of components, building construction, and alike. Therefore, the embodied carbon considered in this work is essentially a lower-bound.

3 Characterization Results

3.1 Methodology

All experiments were conducted on AWS Lambda, one of the most popular public serverless platforms. The *us-west-1* (California) region was primarily used in our studies. The host region can affect our results in two ways: 1) the mix of the underlying hardware, which we control for as described in §3.4.2, and 2) the electric grid's carbon intensity, which we use average regional statistics in §2.2.6.

3.1.1 Metric collection. We use metrics reported by the AWS CloudWatch Lambda Insights [11], a dedicated service for monitoring serverless applications, to feed our carbon models presented in §2. Specifically, we use the following metrics: `duration` for function execution time, `cpu_total_time` as the sum of time spent in user and kernel modes, `used_memory_max` to track maximum memory utilized, `total_network` to capture data transmitted, and `tmp_used` to account for how much of the temporary file system was used. For functions that did not involve network-intensive operations, we still observed some data transfer values from AWS Lambda Insights, which can be attributed to network calls made by the Lambda runtime [11]. To address this variability, we calculated the average of the collected network data. We had to trace the CPU information from the function side by accessing `/proc/cpuinfo`. We distinguished between cold starts and warm starts by reading the field `cold_start` from AWS Lambda Insights logs.

3.1.2 Benchmarks. We analyze invocation logs from five different benchmarks developed in Python, JavaScript, and Java, which are the primary languages used by AWS Lambda users [30]. PyAES [45], a Python-based AWS Lambda function utilizing the AES (Advanced Encryption Standard) algorithm to secure data, and Markdown-to-HTML [62], a Python script that transforms Markdown into HTML. These benchmarks need a reasonable level of computing power and do not involve any external communication. To accommodate various energy sources, such as those derived from CPU or I/O operations, we have chosen specific benchmarks: Video-Processing [29], a Python script used for adding watermarks to videos and converting them to GIFs, and Java-S3 [21], a Java application designed to retrieve, compress, and store images in an S3 bucket. To analyze the

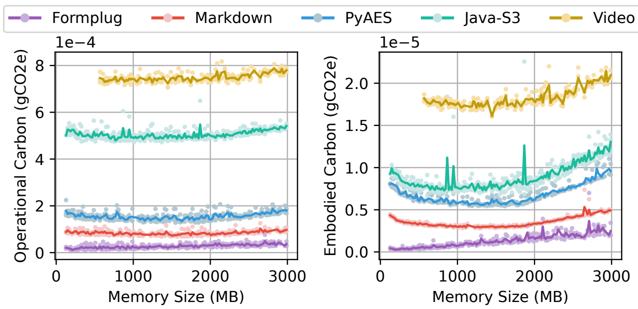

Figure 1: Operational and embodied carbon of five benchmark functions under different memory sizes. The lines represent the average carbon emissions.

behavior of functions under very low memory allocations, we selected Formplug [42], an HTML form forwarding service developed in JavaScript.

We initiated the benchmarks using specific input values and subsequently altered the inputs for each benchmark to test input sensitivity in §3.4.3. Both the Video-Processing and Java-S3 functions processed 1 MB of video and image from an S3 bucket, respectively. PyAES encrypted a message of 256 characters 100 times, while the Markdown-to-Html benchmark generated an HTML web form from a Markdown text of 165k characters.

3.1.3 Sampling. The logs were collected at a sampling rate of 20 MB, spanning memory sizes from 128 MB to 3,008 MB. This sampling rate was implemented across all benchmarks except for Video-Processing, which demonstrated better performance at 500 MB of memory. Each memory configuration was sampled three times, excluding cold starts. Cold starts are analyzed separately in §3.4.1.

3.2 Carbon Attribution

In this section, we explore the attribution of carbon emissions with various criteria. We base our analysis on the execution logs with the Intel Haswell 2.50 GHz CPU, since it is the predominant CPU model as presented in Table 1. Also, we further discuss the impact of different CPU models in §3.4.2.

3.2.1 Operational and Embodied Carbon. Figure 1 illustrates the operational (left) and embodied (right) carbon emissions of five benchmark functions deployed in the *us-west-1* region. As the figure shows, carbon emissions vary with memory configurations for these benchmarks. Emissions do not necessarily increase with higher configurations; for most benchmarks, the emission functions are convex rather than monotonically increasing. This occurs because, up to a certain point, increasing memory—and subsequently CPU allocation—can reduce execution time when a function is resource-bottlenecked. Beyond the threshold where all necessary resources are provided, extra resources only become wasteful.

The other observation is that across benchmarks, operational and embodied emissions do not change with the same ratio. For instance, Java-S3 consistently has more than twice the operational emissions of PyAES, while their embodied emissions are relatively close. This has to do with the varying mix of resources used by different functions, something we characterize in §3.2.2.

The operational carbon is influenced by the carbon intensity of the grid powering the host data center (§2.2.6). Execution on identical hardware in other regions would linearly scale the emissions shown in Figure 1-Left, proportional to the ratio of the region's carbon intensity to that of *us-west-1*. This relationship highlights how regional variations in energy sourcing—not just workload configuration—impact sustainability outcomes; something recently explored by researchers to reduce emissions of serverless workloads [37, 60].

3.2.2 Contribution by Resources. Figure 2 shows the breakdown of operational emissions by resource. For three functions, most of the carbon emissions come from the CPU. For Java-S3 and Video-Processing benchmarks, the network energy is the dominant contributor to the emissions. As noted earlier in §1, this paper specifically focuses on emissions generated during the active execution phase of serverless functions—a scope aligned with serverless billing models. Keeping sandboxes alive to mitigate cold starts [58, 63] extends memory emissions shown in this figure, but that carbon footprint falls under the provider's operational responsibility.

3.3 Carbon vs. Classic Metrics

3.3.1 Carbon vs. Cost. To explore the relationship between carbon emissions and cost, we calculate the cost of each invocation using the AWS pricing model [17] (excluding the free-tier discounts). We then compare the costs with the corresponding carbon emissions. Figure 3 shows the relationship between carbon emissions and cost across all benchmarks. The figure illustrates a "<"-shaped trend as memory configurations increase. The sharp cost rise beyond the Pareto-optimal point stems from this cloud provider charging for allocated—not utilized—resources. For all benchmarks, optimizing for cost leads to carbon optimization. As a result, rightsizing serverless functions will kill two birds with one stone.

3.3.2 Carbon vs. Performance. As Figure 4 shows, optimizing for performance can reduce carbon emissions until the resources required by the function are satisfied. Beyond that point (the knee of the L-shaped curves), increasing the memory configuration merely raises carbon emissions due to the underutilization of resources without improving performance. This underscores the importance of rightsizing serverless functions to optimize performance and minimize carbon emissions resulting from resource over-allocation.

3.4 Sources of Variance

3.4.1 Cold Starts vs. Warm Starts. Cold start executions have higher emissions than warm starts, as quantized in Figure 5. This is due to added resource usage and allocated resources prior to the function execution. The relative increase in emissions is amplified when: 1) the function execution is short (Formplug), and 2) the runtime is slow and resource-intensive, which is the case for JVM (for Java-S3) compared to Python and Node.js runtimes.

3.4.2 Host Processor. Our logs reveal that various CPU models are used to execute functions, with the selection being made by the provider and beyond the developer's control. Using the data derived from varying power levels for both active and idle states, as

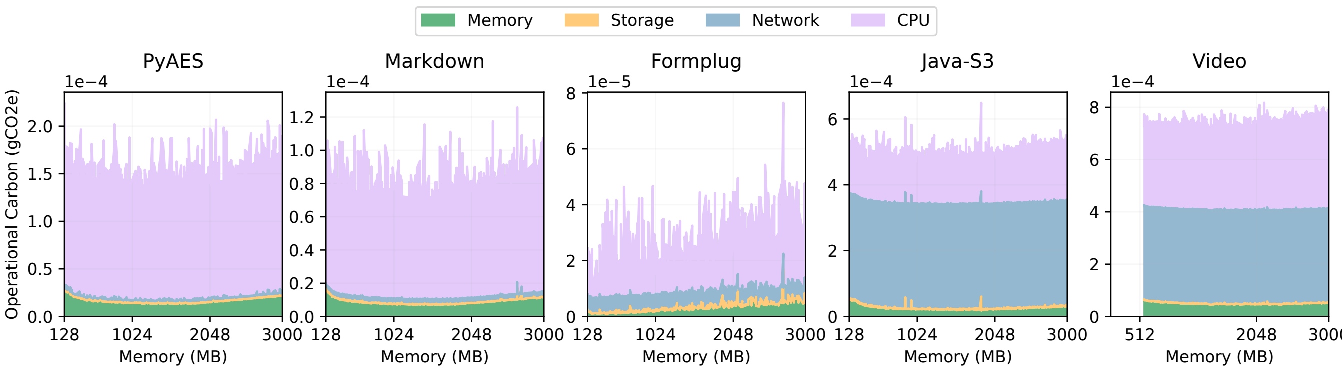

Figure 2: The carbon contribution of resources can vary significantly across functions and configurations.

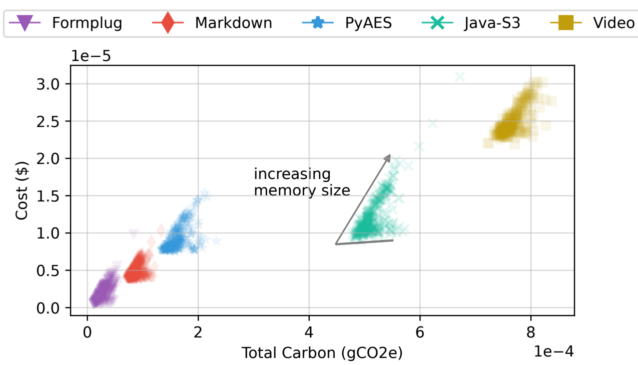

Figure 3: Carbon and cost optimality align well, making function rightsizing essential to address both.

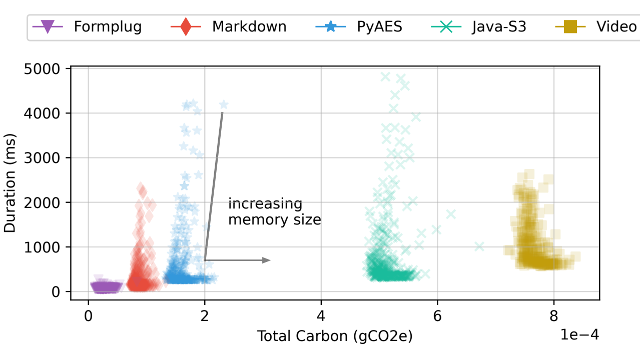

Figure 4: Increasing memory size leads to reduced carbon and better performance (shorter execution duration) until resource needs are satisfied, after which excessive memory merely incurs more carbon without performance gains.

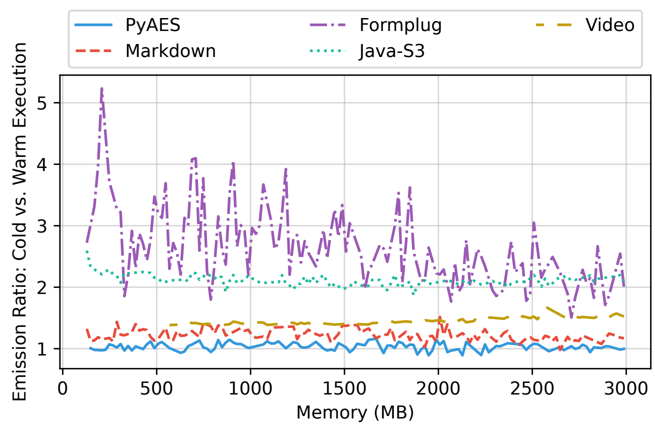

Figure 5: Cold starts can incur significant carbon emissions.

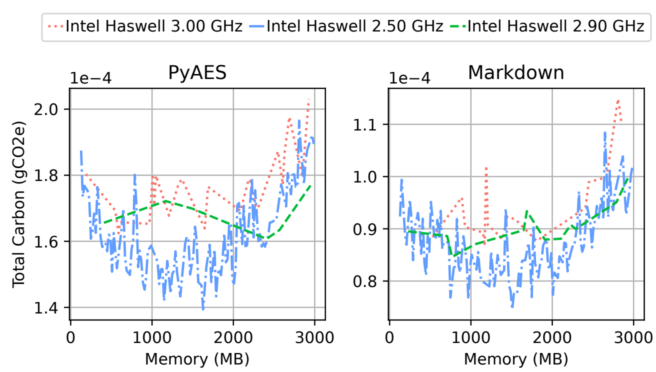

Figure 6: Host processors on the studied cloud platform do not have a major impact on carbon emissions.

outlined in §2.2, we observe that the underlying CPU model does not significantly affect carbon emissions (Figure 6).

3.4.3 Input Sensitivity. To assess the impact of input on carbon footprint, we executed the PyAES and Java-S3 benchmarks using various workloads. Figure 7 shows the carbon footprint of these functions with different inputs. Black markers in the figure indicate median values. Using different inputs can change resource consumption as well as execution time, leading to different carbon emissions. The increase in emissions is not necessarily linear, as there is a baseline emission incurred even without any work performed on the input for calling the function handler and returning the response.

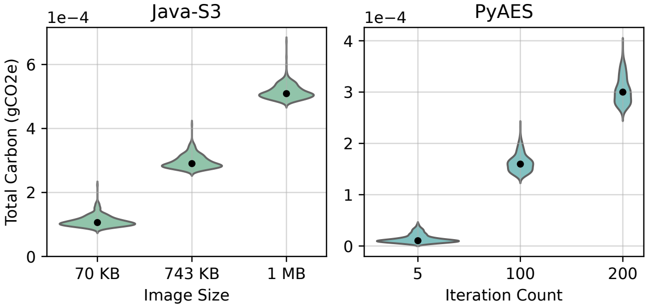

Figure 7: Changing the input can significantly change the carbon footprint of serverless functions.

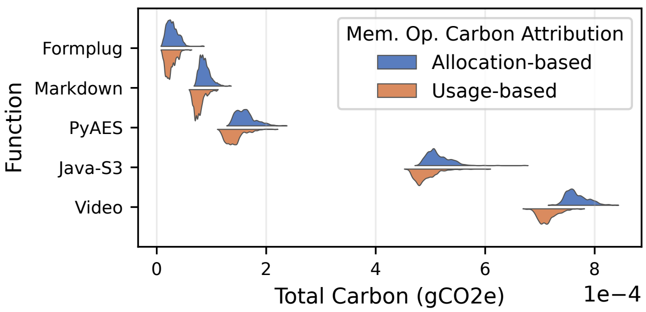

Figure 8: Whether to include unused allocated memory's operational carbon marginally impacts the total footprint.

3.4.4 Memory Allocation Model. Most serverless providers, including AWS Lambda, charge based on configured memory, while others, like Azure Functions, charge only for memory used [54]. The operational carbon model in Equation (3) accounts for unused allocated memory. Figure 8 shows the impact on carbon emissions under a usage-based model (i.e., $P_{mem} = P_{mem}^{high} \times m_{used}$). Distributions cover all samples across memory configurations. In reality, memory overcommitment—which cannot be externally measured—means the true emissions likely fall between the usage-based and allocation-based results presented in the figure.

4 Challenges and Avenues for Future Research

Based on our above emission characterization results, we identify several challenges that necessitate further research.

Cloud carbon transparency. We faced challenges of the opaque cloud infrastructure and the lack of carbon-related metrics from the provider when modeling the operational and embodied carbon of AWS Lambda functions. To estimate the CPU power and embodied carbon, we referred to the CPU datasheet and research report of inferred CPU models (Table 1). We also assumed that the data center leveraged the local power grid and relied on Electricity Maps to obtain carbon intensity, while the actual energy sources and intensities may vary. While the analyses of relative trends, proportions, and correlations are less affected, the absolute carbon values may be less accurate due to the lack of essential carbon metrics of cloud data centers (e.g., power grid carbon intensity and operational and embodied carbon of hardware). To address this challenge, public cloud providers can increase transparency by exposing more reliable carbon metrics, carbon proxies, and embodied carbon data [2, 33, 56, 74]. At the same time, providing fine-grained, real-time carbon emission logs for cloud services can significantly enhance cloud carbon transparency, enabling developers to make informed decisions about workload rightsizing, shifting, and optimization based on carbon emissions. This approach allows providers to factor in emissions from internal system components (e.g., for container keep-alive [62] and distributed caches [57]).

The need for dynamic and flexible resource allocation. Prior work has identified the cost and resource inefficiencies of fixed resource allocation in serverless [24, 75]. We go beyond that, quantifying how the status quo leads to significant missed opportunities for emission reduction. Firstly, static resource allocation is carbon-inefficient, especially for input-sensitive applications where configurations should accommodate peak resource demands [55]. Over-allocation of resources incurs emissions with no gains on performance (Figure 4). Secondly, proportional CPU-memory allocation simplifies scheduling but causes unnecessary emissions unless a function perfectly matches the assigned resource ratio.

Better network energy models. In this study, we faced the lack of effective methods to model the energy consumption of data transfer. This limitation has been brought up by prior work too; e.g., Lyu et al. cite *"no public data on NICs"* [51]. However, the impact can be more substantial in serverless settings, where typical data transfer amounts combined with very short execution times [44, 63] result in a high data-to-compute ratio. Even with the lower than typical transfer energy of 0.001 kWh/GB (§2.2.3), the share of network in operational carbon exceeded 50% for network-bound benchmarks (Figure 2). There is a pressing need for research to develop advanced network energy models and comprehensive profiling methodologies to collect relevant system-level information.

5 Related Work

Modeling carbon emissions of serverless systems. While there has been a large body of work to model carbon emissions of cloud systems [19, 33, 41, 67], there are only a limited number of works focused on building specialized carbon models for serverless systems [26, 50, 58, 64, 65]. Sharma [64] measured the energy footprint of a specific serverless function using the laptop battery interface. Chadha et al. [26] leveraged software carbon intensity (SCI) specification [12], assuming 50% CPU utilization, to model serverless function emissions in terms of CPU and memory. We use a different carbon model that accounts for varying resource usage and emissions from storage and network. Lin et al. [50] proposed a per-request carbon model for serverless functions. We adopted a similar embodied carbon model but employed a different operational carbon model with alternate power models (linear utilization and network energy) since dynamic power metering is not feasible on AWS Lambda. Sharma and Fuerst [65] developed a more advanced energy consumption quantification method by applying statistical disaggregation and fair attribution among functions in a multi-tenant environment. However, this methodology is not externally applicable due to the unknown mix of co-tenants to developers. Basu Roy et al. [58] collected energy consumption estimates by reading MSR registers via the RAPL interface on a c5 bare-metal EC2 instance, enabling them to profile the energy usage of functions in cloud environments—a technique that is not applicable to managed serverless platforms like AWS Lambda.

External characterization of emissions of computing systems. Despite limited access to cloud infrastructure internals, externally characterizing carbon emissions is a crucial first step toward optimizing resource usage, identifying new research opportunities, and promoting sustainability for practitioners. Other researchers have estimated the carbon footprint of computing systems. A number of studies target carbon characterization of AI infrastructure in the cloud [25, 32, 59]. Li et al. [47] have conducted a comprehensive analysis of the carbon footprint of high performance computing (HPC) systems. This work is similar in nature but focuses on serverless functions in a public platform.

Broader efforts by the community to improve the sustainability of cloud systems. The climate urgency along with the sudden increase in emissions of cloud data centers have fueled many research endeavors in this space over the past few years. On the provider side, the community has investigated avenues such as carbon-aware scheduling [26, 40, 43], auto-scaling [39], load balancing [60], incentive design [36], edge offloading [46], resource pooling [38], and even hardware design [73]. From the developer side, there is middleware that allows developers can use for workload shifting [37] to reduce emissions without support from providers. There are also specialized tools and libraries they can use for energy and carbon estimation, such as Kepler [16]. Our characterization work is orthogonal to these efforts, aiming to provide insights for practitioners and motivate them to consider the role of configuration in the sustainability of serverless workloads.

6 Conclusions

We characterize the carbon footprint of serverless workloads with various inputs and configurations running on a widely adopted cloud computing platform, AWS Lambda, across different regions and hardware. In doing so, we use already-available telemetry and publicly available information. Our characterization results shed light on the overall alignment of cost and carbon, but highlight the need for developers to optimize configurations of serverless functions. In the future, effective dynamic resource management can remove this burden. Additionally, there is a need for more research on fine-grained real-time carbon emission reporting and modeling the carbon emissions of networks.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), through research grants RGPIN-2021-03714 and DGECR-2021-00462, and a Canada Graduate Scholarship (CGS D). Resources allocated to us by the Digital Research Alliance of Canada facilitated this work.

References

[1] 2015. Intel(R) Xeon(R) Processor E5-1600, E5-2600, and E5-4600 v3 Product Families, Volume 1 of 2, Electrical Datasheet. https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e5-v3-datasheet-vol-1.pdf. Accessed: Feb 06 2025.
[2] 2021. Carbon proxies: Measuring the greenness of your application - Sustainable Software. https://devblogs.microsoft.com/sustainable-software/carbon-proxies-measuring-the-greenness-of-your-application/. Accessed: Feb 06 2025.
[3] 2022. Microsoft 2022 Environmental Sustainability Report. https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RW14sJN. Accessed: Feb 06 2025.
[4] 2023. AWS Lambda: Resilience under-the-hood | AWS Compute Blog. https://aws.amazon.com/blogs/compute/aws-lambda-resilience-under-the-hood/. Accessed: Feb 06 2025.
[5] 2025. Amazon EC2 instance type specifications - Amazon EC2. https://docs.aws.amazon.com/ec2/latest/instancetypes/ec2-instance-type-specifications.html. Accessed: Feb 06 2025.
[6] 2025. The Cloud - Amazon Sustainability. https://sustainability.aboutamazon.com/products-services/the-cloud. Accessed: Feb 06 2025.
[7] 2025. Configure Lambda function memory - AWS Lambda. https://docs.aws.amazon.com/lambda/latest/dg/configuration-memory.html. Accessed: Feb 06 2025.
[8] 2025. Datavizta. https://dataviz.boavizta.org/serversimpact. Accessed: Feb 06 2025.
[9] 2025. GHG Protocol. https://ghgprotocol.org/. Accessed: Feb 06 2025.
[10] 2025. Intel(R) Xeon(R) Processor E5 v3 Family. https://ark.intel.com/content/www/us/en/ark/products/series/78583/intel-xeon-processor-e5-v3-family.html. Accessed: Feb 06 2025.
[11] 2025. Metrics collected by Lambda Insights - Amazon CloudWatch. https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Lambda-Insights-metrics.html. Accessed: Feb 06 2025.
[12] 2025. Software Carbon Intensity (SCI) Specification. https://sci.greensoftware.foundation/. Accessed: Feb 06 2025.
[13] 2025. Specifications for Amazon EC2 general purpose instances - Amazon EC2. https://docs.aws.amazon.com/ec2/latest/instancetypes/gp.html. Accessed: Feb 06 2025.
[14] 2025. Specifications for Amazon EC2 previous generation instances - Amazon EC2. https://docs.aws.amazon.com/ec2/latest/instancetypes/pg.html. Accessed: Feb 06 2025.
[15] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Aditya Sundarrajan, Kiwan Maeng, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. 2023. Carbon Dependencies in Datacenter Design and Management. *SIGENERGY Energy Inform. Rev.* 3, 3 (Oct. 2023), 21–26.
[16] Marcelo Amaral, Huamin Chen, Tatsuhiro Chiba, Rina Nakazawa, Sunyanan Choochotkaew, Eun Kyung Lee, and Tamar Eilam. 2023. Kepler: A framework to calculate the energy consumption of containerized applications. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE, 69–71.
[17] Amazon Web Services. 2023. AWS Lambda – Pricing. https://aws.amazon.com/lambda/pricing/ Accessed: Feb 06 2025.
[18] Amazon Web Services. 2023. Four Trends Driving Global Utility Digitization. https://aws.amazon.com/blogs/industries/four-trends-driving-global-utility-digitization/ Accessed: Feb 06 2025.
[19] Rohan Arora, Umamaheswari Devi, Tamar Eilam, Aanchal Goyal, Chandra Narayanaswami, and Pritish Parida. 2023. Towards carbon footprint management in hybrid multicloud. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*. 1–7.
[20] Joshua Aslan, Kieren Mayers, Jonathan G Koomey, and Chris France. 2018. Electricity intensity of internet data transmission: Untangling the estimates. *Journal of industrial ecology* 22, 4 (2018), 785–798.
[21] AWS. 2023. S3 image resizer (Java). Retrieved 2025-02-07 from https://github.com/awsdocs/aws-lambda-developer-guide/tree/main/sample-apps/s3-java
[22] Seth Ayers, Sara Ballan, Vanessa Gray, and Rosie McDonald. 2023. Measuring the Emissions and Energy Footprint of the ICT Sector: Implications for Climate Action. (2023).
[23] Jonathan Barnsley, Jhénelle A Williams, Simon Chin-Yee, Anthony Costello, Mark Maslin, Jacqueline McGlade, Richard Taylor, Matthew Winning, and Priti Parikh. 2023. Location location location: a carbon footprint calculator for transparent travel to the UN Climate Conference 2022. *UCL Open Environment* 5 (2023).
[24] Muhammad Bilal, Marco Canini, Rodrigo Fonseca, and Rodrigo Rodrigues. 2023. With Great Freedom Comes Great Opportunity: Rethinking Resource Allocation for Serverless Functions. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. ACM, 381–397.
[25] Lucía Bouza, Aurélie Bugeau, and Loïc Lannelongue. 2023. How to estimate carbon footprint when training deep learning models? A guide and review. *Environmental Research Communications* 5, 11 (2023), 115014.
[26] 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*. 18–23.
[27] Google Cloud. 2025. Viewing Carbon Footprint Data. https://cloud.google.com/carbon-footprint/docs/view-carbon-data Accessed: Feb 06 2025.

[28] IBM Cloud. 2025. IBM Cloud Carbon Calculator Methodology (Version 3). https://cloud.ibm.com/media/docs/downloads/account/carbon-calc-method-v3.pdf Accessed: Feb 06 2025.

[29] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. 2021. SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing. In *Proceedings of the 22nd International Middleware Conference (Middleware '21)*. ACM, 64–78.

[30] Datadog. 2024. State of Serverless. https://www.datadoghq.com/state-of-serverless/. Accessed: Feb 06 2025.

[31] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. 2015. Data center energy consumption modeling: a survey. *IEEE Communications Surveys & Tutorials* 18, 1 (2015), 732–794.

[32] 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* (Seoul, Republic of Korea) *(FAccT '22)*. ACM, 1877–1894.

[33] Tamar Eilam. 2021. Towards transparent and trustworthy cloud carbon accounting. In *Proceedings of the 22nd International Middleware Conference: Extended Abstracts*. 1–5.

[34] Simon Eismann, Long Bui, Johannes Grohmann, Cristina Abad, Nikolas Herbst, and Samuel Kounev. 2021. Sizeless: predicting the optimal size of serverless functions. In *Proceedings of the 22nd International Middleware Conference* (Québec city, Canada) *(Middleware '21)*. ACM, 248–259.

[35] Teads Engineering. 2020. Estimating AWS EC2 Instances' Power Consumption. https://medium.com/teads-engineering/estimating-aws-ec2-instances-power-consumption-c9745e347959 Accessed: Feb 06 2025.

[36] Anshul Gandhi, Dongyoon Lee, Zhenhua Liu, Shuai Mu, Erez Zadok, Kanad Ghose, Kartik Gopalan, Yu David Liu, Syed Rafiul Hussain, and Patrick Mcdaniel. 2023. Metrics for sustainability in data centers. *ACM SIGENERGY Energy Informatics Review* 3, 3 (2023), 40–46.

[37] Viktor Urban Gsteiger, Pin Hong (Daniel) Long, Yiran (Jerry) Sun, Parshan Javanrood, and Mohammad Shahrad. 2024. Caribou: Fine-Grained Geospatial Shifting of Serverless Applications for Sustainability *(SOSP '24)*. ACM, 403–420.

[38] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. 2022. Clio: A hardware-software co-designed disaggregated memory system. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 417–433.

[39] Walid A Hanafy, Qianlin Liang, Noman Bashir, David Irwin, and Prashant Shenoy. 2024. CarbonScaler: leveraging cloud workload elasticity for optimizing carbon-efficiency. *ACM SIGMETRICS Performance Evaluation Review* 52, 1 (2024), 49–50.

[40] Walid A Hanafy, Qianlin Liang, Noman Bashir, Abel Souza, David Irwin, and Prashant Shenoy. 2024. Going Green for Less Green: Optimizing the Cost of Reducing Cloud Carbon Emissions. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 479–496.

[41] Hongyu Hè, Michal Friedman, and Theodoros Rekatsinas. 2023. EnergAt: Fine-Grained Energy Attribution for Multi-Tenancy. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*. 1–8.

[42] Daniel Ireson. 2021. Formplug. Retrieved 2025-02-07 from https://github.com/danielireson/formplug/

[43] Yankai Jiang, Rohan Basu Roy, Baolin Li, and Devesh Tiwari. 2024. EcoLife: Carbon-Aware Serverless Function Scheduling for Sustainable Computing. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[44] Artjom Joosen, Ahmed Hassan, Martin Asenov, Rajkarn Singh, Luke Darlow, Jianfeng Wang, and Adam Barker. 2023. How Does It Function? Characterizing Long-term Trends in Production Serverless Workloads. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '23)*. ACM, 443–458.

[45] Jeongchul Kim and Kyungyong Lee. 2019. FunctionBench: A suite of workloads for serverless cloud function service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 502–504.

[46] Dragi Kimovski, Roland Mathá, Josef Hammer, Narges Mehran, Hermann Hellwagner, and Radu Prodan. 2021. Cloud, fog, or edge: Where to compute? *IEEE Internet Computing* 25, 4 (2021), 30–36.

[47] Baolin Li, Rohan Basu Roy, Daniel Wang, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Toward Sustainable HPC: Carbon Footprint Estimation and Environmental Implications of HPC Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. ACM, Article 19, 15 pages.

[48] Shuwen Liang, Zhi Qiao, Jacob Hochstetler, Song Huang, Song Fu, Weisong Shi, Devesh Tiwari, Hsing-Bung Chen, Bradley Settlemyer, and David Montoya. 2018. Reliability Characterization of Solid State Drives in a Scalable Production Datacenter. In *2018 IEEE International Conference on Big Data (Big Data)*. 3341–3349. https://doi.org/10.1109/BigData.2018.8622643

[49] Changyuan Lin, Nima Mahmoudi, Caixiang Fan, and Hamzeh Khazaei. 2023. Fine-Grained Performance and Cost Modeling and Optimization for FaaS Applications. *IEEE Transactions on Parallel and Distributed Systems* 34, 1 (2023), 180–194.

[50] Changyuan Lin and Mohammad Shahrad. 2024. Bridging the Sustainability Gap in Serverless through Observability and Carbon-Aware Pricing. In *Proceedings of the 3rd Workshop on Sustainable Computer Systems*.

[51] Jialun Lyu, Jaylen Wang, Kali Frost, Chaojie Zhang, Celine Irvene, Esha Choukse, Rodrigo Fonseca, Ricardo Bianchini, Fiodar Kazhamiaka, and Daniel S Berger. 2023. Myths and misconceptions around reducing carbon embedded in cloud platforms. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*.

[52] Jens Malmodin, Nina Lövehagen, Pernilla Bergmark, and Dag Lundén. 2024. ICT sector electricity consumption and greenhouse gas emissions – 2020 outcome. *Telecommunications Policy* 48, 3 (2024), 102701. https://www.sciencedirect.com/science/article/pii/S0308596123002124

[53] Electricity Maps. 2024. Electricity Maps Datasets. https://portal.electricitymaps.com/datasets Accessed: Feb 06 2025.

[54] Microsoft. 2025. Azure Functions Pricing. Microsoft Azure. https://azure.microsoft.com/en-us/pricing/details/functions/#overview Accessed: Mar 03 2025.

[55] Arshia Moghimi, Joe Hattori, Alexander Li, Mehdi Ben Chikha, and Mohammad Shahrad. 2023. Parrotfish: Parametric regression for optimizing serverless functions. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*. 177–192.

[56] Pratyush Patel, Theo Gregersen, and Thomas Anderson. 2023. An agile pathway towards carbon-aware clouds. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*. 1–8.

[57] Francisco Romero, Gohar Irfan Chaudhry, Íñigo Goiri, Pragna Gopa, Paul Batum, Neeraja J. Yadwadkar, Rodrigo Fonseca, Christos Kozyrakis, and Ricardo Bianchini. 2021. Faa$T: A Transparent Auto-Scaling Cache for Serverless Applications. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. ACM, 122–137.

[58] Rohan Basu Roy, Raghavendra Kanakagiri, Yankai Jiang, and Devesh Tiwari. 2024. The Hidden Carbon Footprint of Serverless Computing. In *Proceedings of the 2024 ACM Symposium on Cloud Computing* (Redmond, WA, USA) *(SoCC '24)*. ACM, 570–579.

[59] Stefano Savazzi, Vittorio Rampa, Sanaz Kianoush, and Mehdi Bennis. 2023. An Energy and Carbon Footprint Analysis of Distributed and Federated Learning. *IEEE Transactions on Green Communications and Networking* 7, 1 (2023), 248–264.

[60] Jayden Serenari, Sreekanth Sreekumar, Kaiwen Zhao, Saurabh Sarkar, and Stephen Lee. 2024. GreenWhisk: Emission-Aware Computing for Serverless Platform. In *2024 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 44–54.

[61] Amazon Web Services. 2025. Customer Carbon Footprint Tool Overview. https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/what-is-ccft.html#ccft-gettingstarted Accessed Feb 06 2025.

[62] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. 2019. Architectural implications of function-as-a-service computing. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*. 1063–1075.

[63] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 205–218.

[64] Prateek Sharma. 2023. Challenges and opportunities in sustainable serverless computing. *ACM SIGENERGY Energy Informatics Review* 3, 3 (2023), 53–58.

[65] Prateek Sharma and Alexander Fuerst. 2024. Accountable Carbon Footprints and Energy Profiling For Serverless Functions. In *Proceedings of the 2024 ACM Symposium on Cloud Computing* (Redmond, WA, USA) *(SoCC '24)*. ACM, 522–541.

[66] Emanuele Simili, Gordon Stewart, Samuel Skipsey, Dwayne Spiteri, and David Britton. 2023. Power Efficiency in HEP (x86 vs. ARM). *Power (W)* 350, 400 (2023), 450.

[67] Thibault Simon, David Ekchajzer, Adrien Berthelot, Eric Fourboul, Samuel Rince, and Romain Rouvoy. 2024. BoaviztAPI: a bottom-up model to assess the environmental impacts of cloud services. In *HotCarbon'24*.

[68] Abel Souza, Noman Bashir, Jorge Murillo, Walid Hanafy, Qianlin Liang, David Irwin, and Prashant Shenoy. 2023. Ecovisor: A Virtual Energy System for Carbon-Efficient Applications. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. ACM, 252–265.

[69] Swamit Tannu and Prashant J Nair. 2023. The dirty secret of SSDs: Embodied carbon. *ACM SIGENERGY Energy Informatics Review* 3, 3 (2023), 4–9.

[70] Thoughtworks. 2023. Cloud Carbon Footprint Methodology. https://www.cloudcarbonfootprint.org/docs/methodology. Accessed: Feb 06 2025.

[71] Erica Tomes and Nihat Altiparmak. 2017. A comparative study of HDD and SSD RAIDs' impact on server energy consumption. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 625–626.

[72] B. M. Tudor and Y. M. Teo. 2013. On understanding the energy consumption of ARM-based multicore servers. *SIGMETRICS Performance Evaluation Review* 41, 1 (Jun 2013), 267–278.

[73] Jaylen Wang, Daniel S. Berger, Fiodar Kazhamiaka, Celine Irvene, Chaojie Zhang, Esha Choukse, Kali Frost, Rodrigo Fonseca, Brijesh Warrier, Chetan Bansal,

Jonathan Stern, Ricardo Bianchini, and Akshitha Sriraman. 2024. Designing Cloud Servers for Lower Carbon. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 452–470.

[74] Jaylen Wang, Udit Gupta, and Akshitha Sriraman. 2023. Peeling back the carbon curtain: Carbon optimization challenges in cloud computing. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*. 1–7.

[75] Hanfei Yu, Hao Wang, Jian Li, Xu Yuan, and Seung-Jong Park. 2022. Accelerating Serverless Computing by Harvesting Idle Resources. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. ACM, 1741–1751.

[76] Zhi Zhou, Fangming Liu, Ruolan Zou, Jiangchuan Liu, Hong Xu, and Hai Jin. 2016. Carbon-Aware Online Control of Geo-Distributed Cloud Services. *IEEE Transactions on Parallel and Distributed Systems* 27, 9 (2016), 2506–2519.

[77] Xinhui Zhu, Weixiang Jiang, Fangming Liu, Qixia Zhang, Li Pan, Qiong Chen, and Ziyang Jia. 2020. Heat to Power: Thermal Energy Harvesting and Recycling for Warm Water-Cooled Datacenters. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 405–418.