Accountable Carbon Footprints and Energy Profiling For Serverless Functions

Prateek Sharma
Indiana University Bloomington
prateeks@iu.edu

Alexander Fuerst
Indiana University Bloomington
alfuerst@iu.edu

ABSTRACT

Cloud computing is a significant and growing cause of carbon emissions. Understanding the energy consumption and carbon footprints of cloud applications is a fundamental prerequisite to raising awareness, designing sustainability metrics, and creating targeted system optimizations. In this paper, we address the challenges of providing accurate and full-system (not just CPU) carbon footprints for serverless (FaaS) functions. To the best of our knowledge, this is the first work which develops an energy and carbon metrology framework for FaaS.

Carbon footprints require a new approach to energy profiling. We use FaaS workload properties such as locality to develop a simple and practical online statistical disaggregation approach. Our fine-grained per-invocation carbon footprints also include shared hardware and software emissions, and use insights from Shapley values to fairly account for both operational and embodied emissions. Owing to the growing importance of carbon measurement, we develop a new rigorous marginal energy based validation methodology which results in accountable, complete, and fair footprints. Over a wide range of FaaS workloads and hardware platforms, our energy footprints have an accuracy of > 99%.

CCS CONCEPTS

• **Software and its engineering → Operating systems**; • **Hardware → Enterprise level and data centers power issues**; • **Computer systems organization → Cloud computing**.

SoCC '24, November 20–22, 2024, Redmond, WA, USA

ACM ISBN 979-8-4007-1286-9/24/11…$15.00
https://doi.org/10.1145/3698038.3698531

KEYWORDS

Cloud computing, Functions as a Service, Energy measurement, Carbon footprint, Sustainable computing

ACM Reference Format:
Prateek Sharma and Alexander Fuerst. 2024. Accountable Carbon Footprints and Energy Profiling For Serverless Functions. In *ACM Symposium on Cloud Computing (SoCC '24), November 20–22, 2024, Redmond, WA, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3698038.3698531

1 INTRODUCTION

Energy and carbon are becoming key resources and optimization targets in large scale distributed computing. Cloud platforms consume a significant (> 1%) amount of global energy [52, 100], and reducing it is a vital step in IT decarbonization efforts [13, 16, 18, 98]. With a growing awareness and need for environmental sustainability, the energy and carbon footprint of cloud applications is increasingly serving as the primary accounting and optimization metric [101], complementing the traditional metrics such as monetary cost and resource utilization (e.g., CPU). This requires energy and carbon accounting throughout the cloud computing stack—hardware, resource allocation software, applications, etc. However, cloud abstractions and applications are continuously evolving—making carbon observability and optimization a moving target.

Serverless computing, or Functions as a Service (FaaS), has emerged as a key cloud abstraction which is enabling rapid application development and deployment [14, 25, 92]. Cloud functions are small, self-contained programs, whose entire execution is managed by the cloud provider. They have low cost, auto-scaling, and a "serverless" model where users don't have to worry about explicit resource management. Serverless computing is a major and growing cloud workload [96], and serves as the resource abstraction for a wide range of event-driven applications (such as web and API services, IoT, and ML inference), workflows [27, 77], and even throughput-intensive parallel workloads [23, 46, 47, 113]. FaaS is also an increasingly useful abstraction for harnessing computational accelerators [40, 81, 115] and edge computing resources [107]. While the *performance* of serverless functions has received significant attention, in this paper we take

the first step towards quantifying their energy and carbon utilization and footprints.

Metrics for the carbon footprint of applications such as the Software Carbon Intensity [48] have recently been adopted as international standards [49]. These footprints consist of operational and embodied emissions, which are in turn computed using the application's energy consumption and hardware utilization. In multiplexed resource sharing environments, application-level energy consumption is estimated using power profilers [67] such as Scaphandre [64] and PowerAPI [41]. While application-level power measurement has a long history, these tools have fundamental shortcomings when used to estimate carbon footprints of applications in general and serverless functions in particular. They rely on CPU power measurements (e.g., RAPL) and do not consider system-wide energy; do not account for shared resources; do not scale well to large number of concurrent functions and CPU cores; and are not sufficiently validated against an external ground truth—rendering them inaccurate and fragile. Given the rising importance and ubiquity of carbon footprints for evaluating and optimizing cloud software sustainability, we believe that a new approach to address the above shortcomings is required.

In this paper, we develop the techniques for *in-situ* accountable carbon (and energy) footprints of serverless functions across four major contributions. First, we leverage the workload locality prevalent in FaaS to develop a statistical disaggregation technique which provides accurate estimates of fine-grained *full-system* energy consumption of functions running on a server. Our energy profiler seeks to provide a *complete* footprint, including the energy consumed by shared hardware and software resources such as the FaaS control plane (such as OpenWhisk [7]). We develop a simple and practical energy profiler capable of using a mix of stock hardware and CPU power instrumentation capabilities (such as plug-level power meters and RAPL [118]), and works across a range of edge and server devices. By adapting a Kalman filter inspired approach, we are able to provide *online* energy estimates without offline pre-training, which allows it to be used in diverse heterogeneous cloud environments.

Our second insight is that for serverless functions, and other distributed applications, the energy and carbon footprint is diffused across many local and remote software and hardware components. For instance, the FaaS control plane which performs all the containerization and orchestration of function invocations can itself be a significant energy consumer. We thus develop *fair* energy and carbon attribution methods for estimating the fair-share of a function's contribution to the energy consumption of shared software and hardware. We provide the operational and embodied carbon footprint of functions using the conceptual framework of Shapley values [39, 110]. This approach provides a practical and theoretically grounded carbon footprint metric, and also leads to many open questions about fairness in embodied carbon accounting for the broader research community.

Our third major observation is that the "ground truth" for application level power is crucial for validating energy and carbon footprints. However, prior work on power profiling has typically relied on proxy metrics such as the total hardware power consumption and measuring application power in isolation (without multi-tenancy). To address this fundamental gap in energy metrology, we develop a new approach for empirical validation which uses the *marginal* energy contribution of different functions. The marginal contribution is computed by replaying the workload trace, and measuring the server-level power after removing certain invocations. This provides the ground truth for the function's in-situ power energy consumption, and we will release all our marginal energy ground truth data for facilitating development and testing of accurate energy profilers.

Finally, our fourth contribution is the integration of energy into the FaaS control plane, for which we use Iluvatar [50]. Our system is written in about 6,000 lines of Rust and Python and is open sourced along with more than 100 workload traces, all profiling features, power measurements, and ground truth data[1]. To the best of our knowledge, this is the first work to provide carbon and energy accounting for serverless functions, and we make the following contributions:

(1) Our energy profiling combines direct and model-based disaggregation to provide *accurate and complete* energy footprints for functions. We provide a simple and practical technique for online and full-system energy estimates.

(2) We provide fair attribution of the operational and embodied carbon emissions for functions, by leveraging Shapley values.

(3) We have extensively evaluated the internal and external validity of the energy footprints on multiple FaaS workloads on three different hardware platforms. Our energy footprints are accurate to within 99% of the marginal energy ground truth.

2 BACKGROUND

2.1 Functions as a Service (FaaS)

FaaS allows users to register small snippets of function code that get executed in response to some trigger (such as an HTTP request, message queue event, etc.) [8–10, 92]. Functions are executed inside virtual execution environments such as lightweight hardware virtual machines [15] or OS containers. Cloud functions are "pay for what you use", and their cost is a combination of their maximum memory allocation and their execution duration [8]. The goal of our work is to additionally enable *energy and carbon based* pricing.

[1] https://github.com/COS-IN/faasmeter-socc-artifact

FaaS control planes (such as OpenWhisk [7]) handle all aspects of function execution. They manage the cluster of servers to run functions on, and implement function scheduling, load-balancing, resource monitoring, function status tracking, storing function results, logging, etc. Similar to operating systems, they are an important *shared resource* for functions. Current research and production FaaS control planes are energy-oblivious, and do not incorporate any energy management functionality.

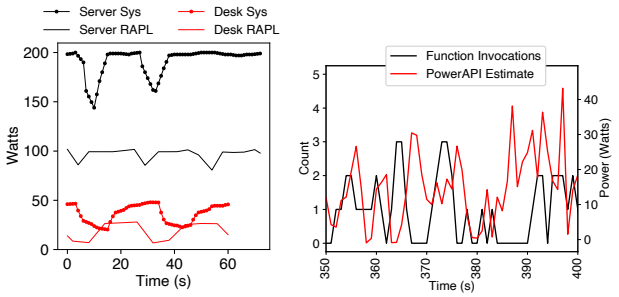
Workloads. Functions are a common abstraction for accessing cloud resources, and are being used for diverse applications such as web-services, ML inference and training, data analysis, parallel and scientific computing, etc [14, 25, 27, 77, 92]. This results in high workload diversity in all dimensions: the CPU, memory requirements, and inter-arrival-times in public clouds such as Azure are heavy tailed [96]. For example, the inter-arrival times of functions in Azure can range from 0.01 s–1 day, and their execution times can range from 0.1 s to 100 s. This also translates to diverse function energy footprints. Functions are also popular in edge computing [2], resulting in heterogeneous execution environments.

2.2 Energy in Cloud Computing

Energy as a first-class resource for operating systems is a long-standing problem and vision [21, 43, 44, 116, 117]. Power virtualization entails accurate process or application level energy measurement [99], and fair attribution of shared energy consumers such as the OS [54, 55, 58].

Measurement and observability into energy usage of applications is the first step towards power virtualization, and is increasingly important for environmentally sustainable cloud systems design and implementation. Major public clouds are now offering carbon footprint tools for certain cloud applications [4, 5, 11]. Given this trend, fine-grained energy footprints of serverless applications will be essential for developing energy-aware cloud applications.

Resource multiplexing is the main challenge for accurate measurement of the power/energy footprint of applications. Energy is a shared, global resource, and can often only be measured at a coarse granularity both in space and time. Hardware capabilities such as RAPL [71, 118], can provide CPU energy (and in some cases, DRAM [36]). "Software power meters" such as `powertop` and others [29, 37, 41, 42, 72, 84, 94, 106, 120], use statistical models to attribute total CPU power to processes based on resource use (such as CPU performance counters). Modern hardware still only has rudimentary support for power measurement. Component-level power (such as for network cards) is usually unavailable. Full-system power can be obtained using server BMCs (baseboard management controllers), battery controllers in mobile devices, external plug-level power meters, or special server hardware [53, 75]. Power measurement and modeling thus continues to be dominated by CPU-power, but even the accuracy and fidelity of CPU power monitoring remains low, with large jitter and temporal errors [17, 67, 71, 86].



(a) System and CPU power is noisy and coarse-grained. **(b) Predicted function footprint with PowerAPI [84] does not correlate with use (i.e., function invocations).**

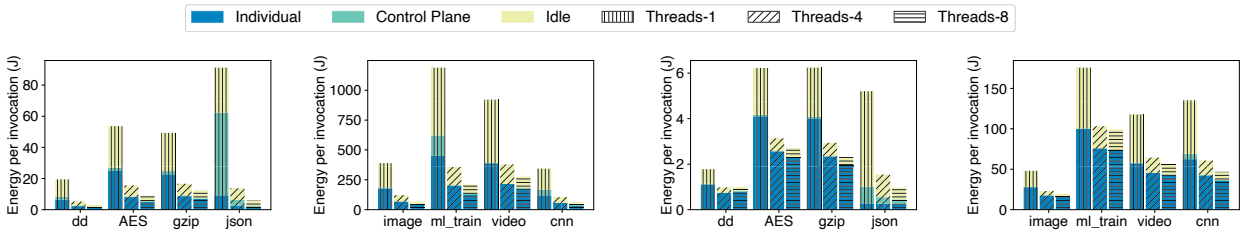
Figure 1: Function power signatures cannot be captured reliably by existing power profiling methods.

2.3 Carbon Footprints

Owing to their significant and growing contribution, the carbon emissions of cloud platforms and energy intensive applications such as AI has been in the spotlight in recent research [100, 112] and elsewhere [31]. The carbon footprint of an application is the total emissions associated with its execution, and these footprints can help quantify, highlight, and optimize the sustainability of computing applications under different execution scenarios.

The *complete* carbon footprint of an application, such as the recent ISO standard SCI (Software Carbon Intensity) comprises of two parts, the operational (O) and the embodied emissions (M), and is defined as: SCI = $\frac{O+M}{R}$, where R is the functional unit of work (such as a function invocation).

The operational emissions is the net energy consumption multiplied by the carbon intensity of the energy source (typically the grid) expressed in grams CO2 per kWh. The grid carbon intensity depends on the mix of energy sources and has significant spatio-temporal dynamics [80]. The second part of the footprint, embodied emissions, reflects the environmental cost of the computing hardware itself, and comprises the emissions due to manufacturing, transportation, and in some cases recycling and disposal. Recent work has highlighted the importance and significance of embodied emissions, which may be as large as 20% of the total lifetime emissions for servers and more than 80% for mobile phones [59].



(a) Server-Small (b) Server-Large (c) Desktop-Small (d) Desktop-Large

Figure 2: Function energy per invocation, measured in isolation. Server load and concurrency levels significantly impact the footprints, making this an unreliable method for energy measurement.

3 CHALLENGES AND GOALS

The high-level goal is to infuse energy and carbon accounting and control into serverless computing. We accomplish this by developing a new energy profiling methodology which provides *accurate, complete, and fairly attributed* carbon footprints for functions. In the rest of this section, we discuss the key drawbacks and challenges with existing techniques to achieve these three requirements.

3.1 Power Profiling Methods

The power consumption of an application is a key input for computing its carbon footprint. However, existing power profiling methods are both fundamentally and practically ill-suited to provide fine-grained function-level measurement. We can divide these methods into two broad classes:

3.1.1 Direct Attribution. In this approach, the hardware power sensors are read periodically, and the power consumed in the sampling interval is attributed to the software components (such as processes and functions) running during that interval. It is used by popular tools like Scaphandre [64], which rely on high-frequency CPU power measurements using RAPL. The fundamental challenge is attributing a single power reading to a large number of concurrently executing components (such as multiple processes). For this power *disaggregation*, the total power is often evenly distributed [20, 83]. High sampling rates and accurate hardware power sensors are vital: smaller sampling intervals (few milliseconds) contain fewer concurrently running components, which makes the disaggregation feasible. Thus the direct attribution approach uses RAPL sensors which can be read with high-frequency (100s of Hz). The overhead of power profiling is also a concern with direct attribution: our evaluation shows that the popular Scaphandre tool can increase CPU and energy consumption by more than 5%.

Compared to CPU power sensors, *system-level* energy can only be reliably obtained at low frequency, resolution, accuracy, and has temporal skew. FaaS workloads compounds the fundamental challenges and these measurement errors. Functions can be very short lived (<1 s), and FaaS servers run hundreds of functions concurrently. These issues are illustrated in Figure 1a, which shows system and CPU power when a *single* compute-intensive ML training function is run in a loop. The "Server" power is measured through the IPMI and inbuilt chassis-level power sensor, and the "Desk(top)" uses a plug-level power meter for full-system power. There is a maximum of one active invocation at a given time, and each "dip" in power corresponds to the gap between invocations. The system power on the server has poor resolution and has large jumps. There are also large synchronization differences between the system and RAPL power on both platforms. On the desktop with a more accurate power-meter, the resolution is higher and the function signatures are more discernible, but the system power's time-diffusion problem persists *even in this best-case and unrealistic scenario of serial invocations.* These issues are amplified in FaaS servers running large numbers of small concurrent functions.

3.1.2 Model-based Power Estimation. Power models of the application and hardware [29, 94] are commonly used in energy measurement. For example, power can be modeled as a function of CPU utilization, which can be estimated with hardware performance counters such as instructions retired, cache misses, etc. For longer-lived entities like VMs, sophisticated ML models can be customized to the application behavior [56]. Compared to power, *performance* measurement can be done with high fidelity, using the wide variety of fine-grained metrics are provided by the hardware and virtualized by the OS. Once a power model for the server (and workload) is built, it can be used to infer the power consumption of individual software components based on their resource consumption.

Existing profilers are not cognizant of function boundaries and execution lifecycle. For example, FaaS control planes employ keep-alive techniques [51] to reduce the cold-start overheads, and keep the container resident in memory between invocations. The container only consumes CPU resources when the function is invoked, which results in a highly non-stationary resource consumption behavior. Existing power models work well for stationary workloads, but highly "bursty" and concurrent FaaS workloads results in inferior fidelity. To illustrate, the output of state of the art

process-based accounting tool, PowerAPI [41] is shown in Figure 1b. The server is running multiple concurrent invocations of a single function, representing the easiest disaggregation case. The PowerAPI energy estimate of the function's containers and the number of "active" function invocations are shown in the figure. When the function is not running, we should expect its container's power to be zero. However, we can see that the predicted energy is not correlated with the number of function invocations, and has temporal skew—making accurate energy footprints difficult to obtain.

3.2 Validation

Empirical validation of power profiling poses many fundamental and practical challenges. We find that prior work on power profiling does not adequately compare against the "ground truth", resulting in uncertain validity of the measurements. The predominant metric for evaluating the accuracy of power profilers is difference between the measured power and the *total* predicted power of all applications (used in [67]). This metric, which we shall call the *total power error*, does not capture the accuracy of the power footprints of the *individual* applications, and is decoupled from any ground truth.

Individual power footprints are sometimes validated using *isolated* measurements. The different applications are run individually, and the total system power can be attributed to the application as the "ground truth" power consumption. A major drawback of this approach is that is not "online", and does not capture the function's energy footprint under realistic loads. To illustrate this, the energy footprints (energy consumed by a function per invocation) for such isolated measurements are shown in Figure 2. We show the average energy per invocation over a 10 minute period where the same function is invoked in a closed-loop. The hardware power consumption is highly dependent on the system load and the power states, and thus increasing the system load by running more concurrent functions affects the footprints. In the figure, we run 1, 4, and 8 concurrent invocations of each function, and we can see that the footprints reduce with load, as the shared and idle power is amortized across individual invocations. Measuring energy in isolation is thus not suitable for validating function-level profiling. We therefore need a new validation methodology with *additional metrics and ground truth.*

3.3 Shared Components

Measuring the energy of function invocations alone is insufficient and does not provide complete accounting. The FaaS control plane also performs many actions on behalf of the functions and is a major shared resource with its own energy footprint, which must be carefully attributed to the individual functions. The sandboxing and management of functions imposes significant work on the control plane, which also increases their energy footprint [97]. The time spent by OpenWhisk for a single (warm) invocation can be up to 600 milliseconds per invocation [50]. This is separate from the actual function execution time (i.e., the "function context"), and is a significant fraction of the total time (and hence resource and energy) consumption of the function.

The control plane interposes on many aspects of function execution asynchronously (such as dealing with the OS virtualization layer, caching container state, etc.). This results in a *fuzzy boundary* between the function execution and the control plane, and exacerbates the challenges in system-level energy measurement described previously. The boundary is also fuzzy in time: since the function's initialized sandboxed is usually kept warm in memory [51], this results in a function's memory-energy footprint outlasting the function execution. The control plane's eviction and container life-cycle management operations also consume CPU resources and energy. The potentially large footprint of shared resources such as the control plane raises new challenges in *fair* attribution: *How should we measure and divide the control plane energy among the functions?*

Due to these major challenges, existing energy profilers are thus unable to provide the necessary input for carbon footprinting. This is illustrated in illustrated in Figure 3 which shows the operational and embodied components of our carbon footprints on desktop and server hardware. For this figure, we assume hardware lifetimes of 5 years and the 2024 average U.S emissions intensity of 386 g CO_2 per kWh. The server and desktop embodied emissions are 471 kg and 175 kg respectively and obtained from the manufacturer [33, 34]. The function's "shared" individual footprint comprises of the control plane and the server's idle power (multiplied by the grid carbon intensity), which can be significant for larger servers.

The popular Scaphandre profiler only captures the CPU power, and provides incomplete and inaccurate footprints (and fails for the disk-intensive `dd` function). Experimental details and further discussion for this figure are presented later in Section 7.2.

4 ENERGY PROFILING DESIGN

We develop a simple and practical approach to energy measurement based on the requirements and challenges identified in the previous section. In this section we focus on the *energy* footprint, but our footprints are multi-dimensional and provide the necessary input to computing the carbon footprints, which is described in Section 5. The high level flow is shown in Figure 4, and this section focuses on the energy profiling component. Our insight is that a "residence time" instead of a traditional utilization based modeling approach can provide complete and accurate footprints.

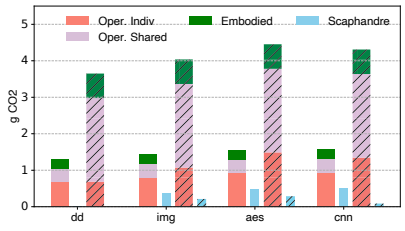


Figure 3: Our carbon footprints consist of the function's individual and shared operational emissions, and hardware embodied emissions. Existing power profilers like Scaphandre provide incomplete footprints. The hatched bars on the right for each function are footprints on a much larger server.

It is a server-level system which integrates with monitoring infrastructure and the FaaS control plane such as OpenWhisk or Iluvatar [50] (see Figure 4). Robustness to measurement noise and workload dynamics is our key design requirement and influences our power modeling. We use three broad categories of input: hardware power measurements; OS and hardware level metrics (such as process-level CPU utilization and CPU performance counters); and a trace of function executions (start and end time of each invocation). The availability and resolution of input metrics can be highly non-uniform (i.e., some hardware sensors may not be available on all platforms). We are thus flexible about input data availability, and can work with a small subset of coarse-grained metrics if necessary.

Our power modeling is deliberately simple to be generalizable and robust, and we prefer explainable linear models to more complex "black box" models such as deep neural networks. We combine both the direct attribution and model-based techniques, and leverage temporal locality of repeated function invocations for statistical disaggregation. We provide *complete* energy accounting of system-wide power by using Shapley value principles of fair division. This provides a wide spectrum of per-function energy footprints with different shared-energy contributions, which are suitable for different tasks pertaining to energy accounting and pricing, capping and control, etc. Figure 2 illustrates this energy spectrum: the function's total energy profile comprises of its "individual" contribution due to function execution, as well as its share of the control plane energy and the idle energy of the server.

Our power profiling has two major components. The input power and workload measurements are disaggregated using the statistical model (Section 4.1), which we augment with a CPU power model (Section 4.3) when RAPL is available to provide a finer-grained power profile. The footprints are continuously updated based on a FaaS-tailored Kalman filter approach (Section 4.2).

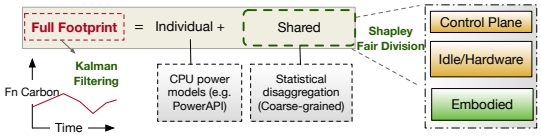


Figure 4: Carbon footprints are a combination of the function's individual and shared energy and embodied emissions.

4.1 Statistical Power Disaggregation

We use the repeated invocation of functions for statistical power disaggregation among functions and shared resources such as the FaaS control plane. The key idea is time-based attribution: the total system power at various points in time can be attributed among all the functions that are executing in a time period of δ. We then collect N such sequential samples (each over a small δ interval) for the various power and workload metrics. M is the total number of unique functions running on the server. The power measurement (W) can be system-wide power, CPU power, or the "rest" of system power which is total system power minus the CPU power.

The key parameter for disaggregation is the "function contribution to power", which is the matrix C with M columns and N rows. We use function running times as proxy for the contribution: $C[j]$ is the total amount of time the function j was running during the interval. Another useful parameter is the number of invocations or activations of each function in the interval, stored in matrix A. The total number of functions the server runs, M, is large, and the number of active functions with non-zero entries in A and C is small. We use simple linear regression for estimating the per-function *power* consumption X:

$$X_{\text{Full}} = \min_X CX - W \tag{1}$$

This is the simplest case which does not consider control plane or idle power, and the power values (X) obtained are referred to as the **full** power. The per-invocation *energy*, J in the interval is obtained by multiplying the power with the average function latency τ: $J_{\text{full}} = X_{\text{full}}\tau$. In some cases, subtracting the idle server power provides more meaningful footprints: $X_{\text{No Idle}} = \min_X CX - (W - W_{\text{idle}})$.

The choice of the measurement interval δ has important tradeoffs. At small intervals ($\delta \sim$ 10ms), only a few functions are active, which makes C sparse. In the extreme case, only one function is active, and all system power can be attributed to it without any further disaggregation. However, in practice, the noise in system-level power measurement increases with the sampling frequency and increases the error. Conversely, larger δ values yield lower variance in power measurements, but denser contributions matrices, which

increases errors in the linear regression solution. We use δ = 1second by default.

For heterogeneous hardware such as GPUs, the hardware energy consumption is separately disaggregated only among functions using the accelerator. Because these accelerators provide a lower degree of statistical multiplexing, the disaggregation is simpler.

Shared Principals. As described in the previous section, the FaaS control planes can also be a significant energy consumer, along with other shared principals like the OS. We augment the above statistical disaggregation to also include these shared principals as additional columns in the contributions matrix C.

Shared principals like the control plane and OS are always running, so unlike functions, we cannot simply use running-time as their "contribution". Instead, we use their CPU utilization as an indicator of energy use. For the control plane, we measure the CPU% of all its processes. Multiplying this CPU% by the time-interval δ gives the fraction of time the control plane was running. However, this underestimates the control plane overhead, since function executions don't necessarily consume 100% CPU. We thus normalize the control plane's contribution by the system-wide CPU:

$$c_{cp} = \frac{\text{control plane CPU\%}}{\text{system-wide CPU\%}} * \delta \tag{2}$$

This yields x_{cp}, the control plane power, which is then divided among all functions using the Shapley value fair share principles described later in this section. We can similarly account for other shared components like the OS, by using the kernel's CPU-time and applying similar normalization.

4.2 Kalman-Filter Guided Online Profiling

We continuously update the function power estimates X based on new measurements. For simple online footprints, various techniques such as online least squares regression and exponentially weighted moving averages can be used. Our insight is that the Kalman filter framework [82, 109] provides intuitive online estimates which are robust to measurement noise and FaaS workload dynamics. For example, we can capture the change in function input, running time, and system load—all of which can affect the per-invocation footprint.

The high-level intuition is to combine the previous estimates X_{i-1} with the new measurements C_i, W_i, and also account for the changes or variance in the new measurements (i.e., the process and measurement noise). The outline of our Kalman filtering algorithm is presented in Figure 5. The filtered estimate is then given by:

$$\hat{X}_i = \alpha \hat{X}_{i-1} + \beta U_i + K Z_i, \tag{3}$$

where α, β are tunable parameters. Z_i is the error we get if we use the previous estimate with the new measurements (also referred to as the "innovation" in standard Kalman filtering). K is the Kalman gain, which is the main component influencing how the innovation is distributed among functions and how footprints evolve. Our intuition is that updates to function footprints should be based on two factors: i) the number of invocations in the interval (A), and ii) their historical latency variance ($\sigma(T)$). For instance, functions not executed in the interval should see no changes in their footprint. The latency variance is a factor because our footprints are proportional to function latency (C), and functions with higher latency variance should receive a smaller update. The latency variance is cumulative and also updated in each step (not shown in the algorithm), and γ is the third tunable parameter.

The Kalman gain consists of the overall change in the state of the system and the measurement noise r, which is the error in hardware power measurement due to high-frequency sampling. Note that hardware power measurement is coarse-grained [71], so a high sampling frequency leads to stale measurements, which contribute to the measurement noise. Thus based on the sampling tradeoffs, the measurement noise is set proportional to $1/\delta$.

The process noise (P) is updated after the Kalman step, and reflects how much the workload (i.e., the state of the system) has changed. We use the relative invocation frequencies (A_i) of the function in the new time period and compare it to the historic frequencies to determine workload similarity (used to set K in the algorithm in Figure 5). The intuition is that if the workload changes significantly, then the historical power estimates have less weight.

For new functions without any estimates, we set $\alpha = 0, \beta = 1$, and $K = 0$. The initial estimates X_0 are obtained using statistical disaggregation on a large initial time-step ($N_{\text{Init}} \sim$ 2minutes). Optionally, estimates from previous profiling runs or other servers in the cluster can also be used as the initial value. The subsequent Kalman steps are performed over the time-step N_K in the range of $1-2$ minutes. The same sparsity tradeoffs apply: smaller time-steps result in more sparsity and frequent updates, but are impacted more by latency variance and measurement noise.

Using the above Kalman-filter approach which incorporates noise and workload dynamics significantly improves the accuracy and stability of the footprints, which is illustrated in Figure 6. The figure shows the energy per invocations of four functions over time. The "memoryless" policy does not keep past history and runs a new power profiling step every time-step (which is one minute in duration)—resulting in high jitter. By contrast, the "cumulative" policy uses *all* the past history to run the statistical disaggregation. While this results in stable footprints, it does not adjust to changing function workload, and yields different footprints compared to the Kalman-filter approach—highlighting the

```
σ(T): variance of function latencies
A_i: num fn invocations during interval i
r ∝ 1/delta

def Kalman-step (X_{i-1}, C_i, W_i, P_{i-1}):
  U_i = min_X (C_i X − W_i)
  Z_i = W_i − C_i X̂_{i-1}
  P = αP_{i-1} + γσ(T)
  K = PA_i^T / A_i P A_i^T + r
  P_{i-1} = (1 − KA_i)P
  return X̂_i = αX̂_{i-1} + βU_i + KZ_i
```

Figure 5: Kalman filter-inspired approach for updating function per-invocation power X over time.

importance of online adjustments. A more thorough evaluation of the accuracy of the footprints is presented later in Section 7.

4.3 CPU Power Modeling

The statistical disaggregation technique described above has many advantages: it is simple, and requires only coarse grained power and latency measurements. We combine this phenomenological approach with more a causal CPU power model for increased accuracy. We build on the plethora of CPU power models [29] and use hardware performance counters to map function CPU usage to power consumption. A CPU model θ_{CPU} is built and used to predict the function's CPU-only power X_{CPU} over each time-step.

$X_{\text{CPU}} = \theta_{\text{CPU}}(S)$, where S is a vector comprising of the function's performance counters, normalized by the system-wide counters. Our approach is similar to PowerAPI and SmartWatts [41], and uses the standard performance counters: `UNHALTED_CORE_CYCLES`, `UNHALTED_REFERENCE_CYCLES`, `LLC_MISSES`, and `INSTRUCTION_RETIRED`. We use `perf` to obtain the function-container counters and aggregate the values for multiple concurrent containers of the same function. The model θ_{CPU} is trained using SVR with a linear kernel during initial operation [95]. This model is stable as long as the function execution footprint (CPU work done and IPC) is stationary across invocations. We continuously monitor the model error (difference between observed CPU power and the sum of all predicted function powers), and retrain the model if error exceeds a set threshold (default of 5%).

We can combine the CPU power and rest of the system power estimates. The rest of the system power is obtained using the statistical disaggregation (and Kalman Filter): $X_{\text{Rest}} = \min_X(CX - W_{\text{Rest}})$, where $W_{\text{Rest}} = W_{\text{Sys}} - W_{\text{CPU}}$. When RAPL is available, the default is this **combined** mode, $X_{\text{Combined}} = X_{\text{CPU}} + X_{\text{Rest}}$.

5 FAIR ENERGY AND CARBON ATTRIBUTION

Once the function-level power estimates (e.g., X_{Combined}) are computed over some interval of time (N), we can use them to compute different energy and carbon footprints.

Individual Footprint. In the simplest case, the function's average individual energy consumption is computed: $J_{\text{Indiv}} = X_{\text{No Idle}}\tau$, where τ is the average function latency in the interval. Note that X is dependent on the function code, and in practice does not vary significantly since popular functions such as ML inference and multimedia processing are fairly deterministic [57]. Of course, the execution time τ can vary based on the input size and resource contention—we analyze its effect on the energy in Section 7.

Shapley Footprint. The above individual footprint does not take into account the power consumed by the shared resources such as the control plane and server hardware. Complete footprints require that we fairly dividing the energy (and carbon footprint) of these shared resources, for which we use Shapley value principles [110] and techniques. The Shapley value of each function would be it's "true" energy footprint, and satisfy many desirable properties, and can be considered the gold-standard of energy attribution [39] in multi-tenant environments. Shapley values are also being used for interpretability of machine learning models [102], by attributing the importance of model features.

Unfortunately, computing Shapley values requires sampling an exponential number of energy readings that cover all the permutations of function invocations (i.e., entries of the C matrix), and the true marginal energies for all function combinations, which we have no way of obtaining accurately. Exact Shapley values are thus infeasible and impractical, especially considering measurement noise and under an online setting. Instead, we approximate the Shapley values by satisfying its four properties in a best effort manner:

(1) *Efficiency:* the sum of all function footprints should add up to the total system-level energy. We try to achieve this by minimizing net error in the Kalman filter.

(2) *Null-player* property requires functions not executing to have 0 energy, which we get by construction of our C matrix.

(3) *Symmetry:* identical functions (both in their code and invocation frequency) should have the same footprints.

(4) *Linearity:* the total shared-resource energy attributed to a function should be the sum of the individual shared resources.

Based on these principles, we divide the shared idle and control plane energy among the functions. Idle power is a "static" shared resource, and it must be *evenly* among all functions (proof omitted, but follows similarly from [66] which applies Shapley values to datacenter power). On the other hand, the control plane energy is "dynamic" and depends on

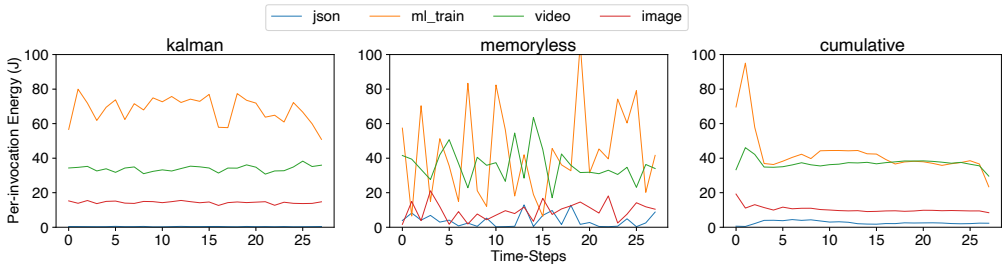


Figure 6: Change ine energy footprints over time. The Kalman-filter approach is able to adjust to changing workload and system dynamics.

its use (i.e., how many functions are invoked), and it must be divided *proportionally* among functions on a *per-invocation* basis. The efficiency and linearity properties require that we add the individual and static and dynamic energy shares to obtain the total energy for each function:

$$J_{\text{Shap}} = J_{\text{Indiv}} + J_{\text{Shared}} = J_{\text{Indiv}} + \phi_{\text{cp}} + \phi_{\text{idle}}, \tag{4}$$

where $\phi_{\text{cp}}, \phi_{\text{idle}}$ are the shares of control plane and idle energy respectively. J_{Indiv} is obtained by discounting the idle power, i.e., using $X_{\text{No Idle}}$. $\phi_{\text{cp}} = J_{\text{cp}} A_i / \sum(A)$, where A_i is the number of invocations of function i, and A is the vector of A_i. We divide the control plane energy proportional to function invocation frequency (over the time interval). $\phi_{\text{idle}} = J_{\text{idle}}/M$, where M is the number of unique active functions in the interval, which is the number of non-zero entries in A. The function footprints are then the *per-invocation* energy, which is J_{shap}/A.

The Shapley footprint gives applications a full and complete picture of their energy consumption. Since the energy footprints are linear combinations of the individual, dynamic, and static shared power, they can be combined in different ways depending on the intended use-case. For example, when developers are optimizing the energy footprint of their functions, only the direct and individual energy (without any control plane or idle overheads) is suitable.

Finally, our energy footprints are also designed to explore the limits of a simple model and system-level power measurements. If device-level (such as disks and NICs) power can be instrumented (which it cannot in current commodity hardware), then their power can be similarly disaggregated using our techniques. While currently we use aggregate GPU power, specialized power models for GPUs [63, 70] can also be used, similar to how we use RAPL-based models. Without any specialized models, fine-grained per-device footprints can also be considered to be the "dynamic" shared power and be part of the Shapley footprints.

Carbon Footprints are computed by combining the energy footprints with the operational and embodied emissions:

$$C_{\text{Shap}} = kJ_{\text{Shap}} + e/M, \tag{5}$$

where k is the carbon intensity of the electricity source, and e is the embodied emission "rate" during the period. The total embodied cost of the server is $\mathcal{E}$, which must be paid over the lifetime $\mathcal{L}$ of the server (typically 5 years). The embodied rate e is then $e = \frac{\mathcal{E}N}{\mathcal{L}}$, where N is our original measurement interval (typically 1 minute).

Note that the embodied fraction is based on whether a function runs on the server or not (during the time interval), irrespective of the number of invocations. Thus, all "active" functions have equal embodied contribution. Since the server is a static fixed cost, Shapley valuation dictates that it be evenly shared among the active functions and irrespective of how much of the server they use.

This subtle difference has important ramifications on the footprints and incentives for providers and users to optimize carbon. First, it strongly favors locality of execution and load balancing policies which run functions on the same server to reduce cold-starts. With the embodied carbon accounting, there is even more incentive to retain locality. "Popular" functions with higher invocation frequencies naturally have a higher total operational footprint anyways, and thus this approach reduces their embodied burden. Finally, our proposed carbon accounting scheme also imposes different incentives on the function developers. If a larger function is split into multiple (say two) smaller functions, then each smaller function invocation is required to pay the embodied "tax" and also double the dynamic control plane cost. This also incentivizes locality and performance, since it reduces additional networking and sandboxing latency as well.

Comparison with SCI. We can also derive and adapt the Software Carbon Intensity [48] metric in the context of FaaS. The SCI can be simplified as : $C_{\text{SCI}} = kJ_{\text{Full}} + er$, where r is the resource usage fraction of the function. For function i, $r_i = \frac{A_i \tau_i}{\sum_{i<M} A_i \tau_i}$. Compared to our Shapley value formulation, the main difference in the SCI metric is that it considers embodied emissions as a usage-based cost, whereas we believe that embodied emissions are more appropriately viewed as static sunk costs. $C_{\text{SCI}} \approx J_{\text{Server}}(k+e)r$, where J_{Server} is the empirically measured server energy. Finally, we note that

"fairness issues" in carbon accounting and pricing (such as who pays for past embodied emissions) continues to be a highly globally vexing and subjective issue [30, 74], which is well outside our scope.

5.1 Limitations

Some of the limitations of our energy profiling approach are a result of our focus on simple and validatable footprints. Our focus is also on individual functions. For multi-function applications and workflows, the footprints are approximated as the aggregation (sum) of the component functions. This will not account for the shared networking and storage costs, since we only consider server-level power. At a high-level, we can view networking and distributed storage as shared resources, and use our fair division techniques to disaggregate their energy consumption among all the functions. Finer-grained profiling will require additional storage and networking power models along with distributed tracing to track function resource usage. Nevertheless, networking and storage are not power proportional and network elements like switches can have a fairly "static" power consumption, which will again require fair division. While these more "complete" footprints can be obtained by applying our proposed techniques, they are harder to validate against ground-truth, and are part of our future work.

Our simple power models are complementary to finer-grained deep neural network models trained on more hardware performance counters and function inputs. Because of our linear models, we do not need offline training, and have better generalizability across hardware platforms and workloads. Moreover, larger models do not address the main challenges identified in the previous section such as shared resources and diffused energy consumption.

6 IMPLEMENTATION AND VALIDATION METHODS

Our energy profiler and carbon footprint estimation techniques are implemented in the open source Iluvatar FaaS control plane. The energy footprints are made available to the control plane for internal power capping and control. Our changes are implemented in Python and Rust in around 6,000 lines of code. We chose Iluvatar because it provides low latency function invocations and has reduced jitter. Prior work has shown that popular FaaS frameworks like OpenWhisk can add 100s of milliseconds of latency even for warm invocations [50]. Time-based energy attribution is a major component of our power model, and OpenWhisk's own power consumption adds significant noise to the energy footprints. Validating footprints in such excessively noisy conditions was also a challenge (larger number of trials needed to obtain a reasonable confidence interval of the marginal energies). At a conceptual and implementation level, our techniques are independent of the FaaS framework, but the tighter latency bounds of Iluvatar aids in a more rigorous validation.

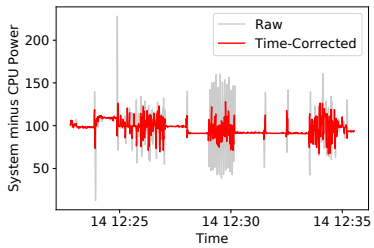


Figure 7: System-level power can be synchronized using CPU power as a reference signal, reducing the errors due to lag.

Different full-system power sources are supported. For servers equipped with chassis-level current and power sensors, we use IPMI to query the BMC (baseboard management controllers). We also support external plug-level power meters, and query their power via serial or telnet interfaces. Many low-power edge devices also provide system-level power. For example, the Nvidia Jetson Orin AGX has current and power sensors which we query using tegrastats. Finally, for laptops and other battery powered devices, the battery charge controller can provide the energy discharge, which we can obtain via ACPI interfaces, and obtain the power consumption. We use the `perf` tool for both the system-wide RAPL and per-function RAPL and per-function CPU performance counters.

Power De-noising and Synchronization. We filter and synchronize the raw power signals before using them for disaggregation. "External" power using external plug-level power meters and even BMC/IPMI can have a time-skew in their measurement and reporting path. Synchronizing the system-power is crucial—otherwise energy is attributed to previous/future functions, reducing the footprint accuracy. We correct the temporal skew (i.e., lag) by correlating (in time) the power signal with some other reference.

For instance, we have observed that IPMI power has significant lag, when compared to the RAPL power which is much more "real-time". This can be observed from Figure 7 which shows the difference between system-level and CPU power. The workload is a CPU dominant application (ML training), and the server has no other major dynamic power-consuming devices (no GPU etc.). Thus we should expect this difference to be constant. However, we see that the raw difference shows significant variance, which we attribute to the measurement lag in the IPMI power-sensor.

We find and fix this lag using simple signal-shifting methods [88]. We find the time-offset s which minimizes the chi-square difference between the power signal, W and the reference signal R, after normalizing by the average:

$$s^* = \min_s \left(\frac{W(t+s)}{\bar{W}} - \frac{R(t)}{\bar{R}} \right)^2 \quad (6)$$

This can be solved using general optimization solvers such as limited-memory BFGS [89], after setting bounds on s of a few seconds. The difference in the power and reference signal after time-skew correction is also shown in Figure 7, and we can see a significant reduction in the variance (i.e., the noise). We compute this skew both during an initialization phase, and periodically, to capture any drift. The reference signal is CPU power by default—other reliable load metrics like CPU instruction and cycle counters are the fallback synchronizing inputs.

6.1 Validation Methods and Metrics

We develop and use a range of metrics for energy and carbon footprints. Given an energy footprint, we can compute the operational and the total carbon footprint. Our metrics are also applicable to carbon footprints, but we focus on energy for simplicity.

Our methodology and metrics required for validating these footprints fall into two broad classes. Through *external validation*, we compare the energy footprints with other reliable energy measurement methods, and develop the primary accuracy metrics. On the other hand, the *internal validity* looks at the consistency of energy footprints with respect to each other, or other system utilization and performance metrics.

External validity for energy disaggregation is challenging: we want to estimate the function's energy contribution in a long and dynamic workload. Our primary benchmark and "ground truth" for external validity is the **marginal energy**, which we compute by running two nearly identical workload traces, and subtracting their *total* energy consumption. A workload trace ($\mathcal{T}$) is characterized by the set of functions ($\mathcal{S}$), their IAT CDFs, and the total duration. Even using extremely coarse-grained power measurements, we can obtain $\mathcal{J}(\mathcal{T})$, the total energy consumption of running the workload trace on a server. The marginal energy of a function f is obtained by running a new trace $\mathcal{T}(\mathcal{S}-f)$ which does not contain the function, and is given by:

$$\mathcal{M}_f = \frac{\mathcal{J}(\mathcal{T}(\mathcal{S})) - \mathcal{J}(\mathcal{T}(\mathcal{S}-f))}{\text{number of invocations of f in S}}. \quad (7)$$

The marginal energy is thus the increase in total energy consumption caused by the function. Note that it *does not* account for the idle server energy, since it is present in both the traces.

This marginal energy validation is essential, since energy is highly sensitive to the system power states (such as CPU frequency). Measuring the function energy footprints "in isolation" is a simpler and alternative technique, where a single function is run without any other function, and per-invocation energy is obtained from the total system energy. But because of the different power states, the per-function energy footprints obtained through this conventional and simpler method have a very high range, dependent on the system load. Figure 2 shows the energy per invocation with the isolated measurement technique when different numbers of the same function are executed concurrently, and we can see significant differences (more than 10×) in footprints based on the load.

Metric	Definition
Individual-Difference	$\lvert J - J^* \rvert / J^*$
Cosine-Similarity	$J \cdot J^* / \lvert\lvert J \rvert\rvert \lvert\lvert J^* \rvert\rvert$
Total-Error	$E[\ \lvert W(t) - \hat{W}(t) \rvert / W(t)\ ]$
Latency-normalized-Variance	$\sigma(J)/\sigma(T)$

Table 1: External and internal validity metrics. J and $\hat{W}$ are profiler outputs, J^* is ground truth, and T is function latency.

Validation Metrics. Table 1 lists our external and internal validation metrics. For external validity, we define and compute different *distance metrics* between the energy profiler output vector (J) and some reference ground-truth J^* (e.g., marginal energy footprints). The first metric provides the *per-function* individual difference to the ground truth. Since the marginal per-invocation energy is obtained by running a separate, smaller workload, it may not always reflect the "live" online energy. These are primarily due to differences in system power consumption and efficiency, and underlying hardware control. Deviations from the marginal can also arise because of different attribution policies for idle and other shared energy. To achieve "complete" energy accounting, the footprints may be elevated for *all* functions. To account for these issues, we use the *cosine similarity* between the energy footprints, to capture the ratios of energy footprints among the different functions. Higher cosine similarity (closer to 1) indicates that a closer footprint match. *Cosine similarity with the marginal energies is our primary external validation metric.*

The second category of metrics are for internal validity. The conventional and popular metric for energy profilers captures the "completeness" of accounting, by computing the difference over time, between the observed ($W(t)$) and predicted total power ($\hat{W}(t)$). Optimizing solely for this total error can be at the expense of the error in energy footprints. This total power error is thus of secondary importance to us, and is controllable via our Kalman filter parameters: higher

values of β in Figure 5 reduce the total error but result in higher variance in footprints.

Our primary internal validation metric is the variance in the energy footprints, normalized to the variance in latency. This helps use evaluate the precision and feasibility of our footprints for energy pricing. Currently, cloud functions are priced based on their execution time (i.e., latency), and thus the latency variance serves as the baseline for comparison.

7 EXPERIMENTAL EVALUATION

Our evaluation is centered around two main questions, i) How accurately can we obtain full-system energy profiles of functions, and ii) What are the characteristics of energy and carbon footprints? We empirically evaluate these footprints on more than 100 workload traces (including marginal energy traces) using seven external and internal validity metrics. These workload and measurement traces and associated ground truths are made publicly available [3], to help build better energy profiling models and accelerate empirical sustainability research.

Measurement Platforms. We use three different types of hardware platforms:

(1) *Server:* Supermicro X11DPT-PS board with 2 48-core Intel Xeon Platinum 8160 CPUs and 1TB of RAM, running Ubuntu 20.04. It idles at 95 Watts.

(2) *Desktop:* Dell Optiplex with 12th Gen Intel i5-12500 running Ubuntu 20.04.5. It has idle power of 15W. Power is being measured using an external SpecPower-approved power-meter (Instek GPM-8310) every 0.25 seconds through the meter's telnet/SCPI interface.

(3) *Edge:* Nvidia Jetson Orin AGX [12]. Power is measured using inbuilt current sensors (via `tegrastat`) and also validated with an external power meter. We run a combination of CPU and GPU functions on this edge device.

The grid carbon intensity (grams CO2 per kWh) varies significantly by time and location [80], and we assume the constant US average grid carbon intensity of 386 g CO2 per kWh. Similarly, the embodied footprint also varies based on hardware type and the carbon accounting methodology. For our analysis, the embodied emissions of the server and desktop are 471 kg and 175 kg respectively [32, 35], and all devices are assumed to have a five year lifespan. Instead of varying and controlling the above carbon parameters, we fix them, and in some of our evaluation results, we omit them and focus on the energy footprints. This reduces the number of variables in the experiment design, and also helps reduce the uncertainty, since many of the carbon parameters can have a very high variance [22, 76]. For carbon footprints, we will largely focus on the operational footprint, since the embodied component can be significant in both magnitude and variance, and can thus obscure the energy validation.

Name	Latency (s)	Description
dd	0.7	Read and write local storage.
image	1.5	Performs several transformations on an image.
video	7.8	Download and grayscale a small video.
AES	1.4	Encrypt and decrypt payload multiple times.
json	0.25	Download, parse, and serialize a json blob.
CNN	1.3	Inference on a TensorFlow model (CPU and GPU).
ml_train	5.1	Train a regression model on a 20 MB dataset

Table 2: Super-set of functions used in our empirical energy and carbon analysis. Latency is average warm running time on the desktop.

FaaS workloads are generated using different combinations of functions and inter arrival time (IAT) distributions. We use different sizes and types of functions from functionbench [73]—their characteristics are described in Table 2. Recall from Equation 1 that the main variables determining the power profiles (X_i) are the latencies and popularities of each function which make up its energy contribution (C_i). We test energy disaggregation with different combinations (subsets) of functions in each trace. Our evaluation covers a large multi-dimensional space of function-subsets, inter-arrival-times, hardware platforms, and metrics. We cover a sparse subset of this evaluation space, and use different types of workloads to show the versatility and generalizability of our approach. Since different metrics are analyzed using different traces, we also summarize the key metrics across all traces towards the end of this section.

The arrival-rates are either determined by: i) the traditional exponential distribution, or ii) sampled from the Azure trace [96], or iii) non-stationary bursty arrivals. For the Azure sampling, we use a combination of random function sampling and scaling the IATs to be suitable on a single server. We use Iluvatar's load generator [50] for generating the workload traces, which have arrival times for each function. We use a sufficient keep-alive cache [51] to ensure > 99% warm starts, and all the latency and energy footprints we report are for warm starts. Cold-starts are tagged by the FaaS control plane, and we can obtain separate cold and warm energy fingerprints if necessary.

We use the Kalman filter with $\alpha = 0.8, \beta = 0.2, \gamma = 0.1$, with an initial window of 100 seconds, and subsequent intervals of 60 seconds. Our empirical analysis has not found the profiler output to be particularly sensitive to these parameters. We compare against Scaphandre [64], a popular process-level CPU-only power profiler which uses the direct disaggregation approach. We have implemented additional post-processing for Scaphandre to turn the process-level power output to per-function-invocation metrics, by correlating the host process id with the function (Iluvatar runs functions in `containerd` containers by default).

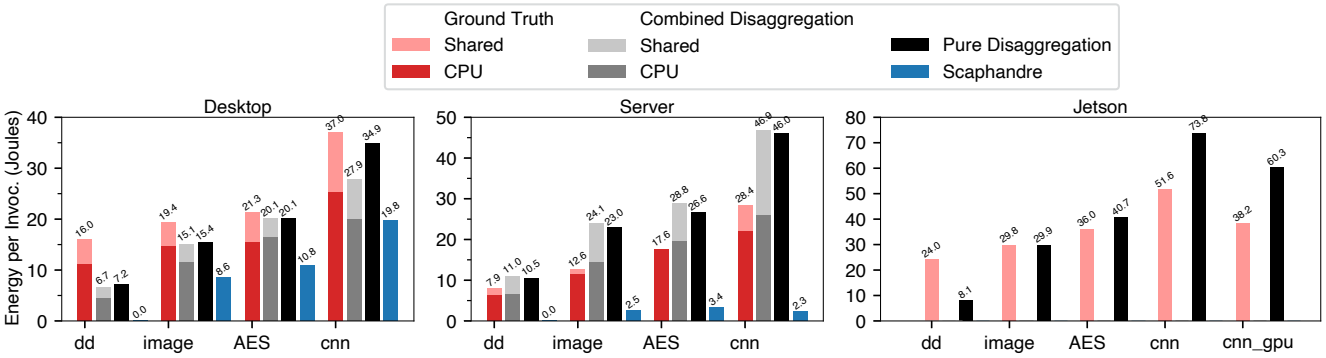


Figure 8: Energy-per-invocation for a four-function trace. Marginal energy serves as ground-truth. Our energy footprints with combined and pure disaggregation are accurate across all three hardware platforms. Scaphandre [64] provides inaccurate footprints, especially for non-CPU-intensive functions (dd), and requires x86 RAPL counters (not available on Jetson).

Platform	Full Disagg.	Combined Disagg.	Scaphandre
Desktop	**0.985**	0.984	0.910
Server	**0.998**	0.998	0.623
Jetson	**0.992**	N/A	N/A

Table 3: The high cosine similarity of footprints indicates high accuracy with respect to ground truth.

7.1 Energy Profiler External Validity

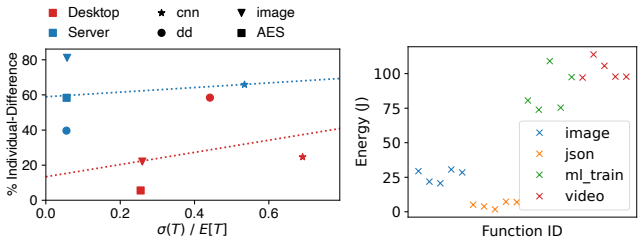
We focus the first part of our evaluation on one heterogeneous trace with four functions, with the goal to understand the accuracy and robustness of the energy profiling method. The function IATs are scaled for the three hardware platforms, such that the desktop and Jetson are at 80% utilization, and the server is at 40%. The per-invocation individual energy footprints (J_{Indiv}) for each function are shown in Figure 8. We compare against the marginal energy as the ground truth baseline, and also show the results of Scaphandre [64]. We evaluate two different profiler configurations. In the *combined* mode, the CPU power is measured separately and added to the rest of the system power using statistical disaggregation. In the *pure disaggregation* mode, only the coarse grained statistical disaggregation is used on the full-system power. In both cases, we subtract the idle hardware power, and thus compute the $X_{\text{No idle}}$ described in Section 4.

On the desktop, both these approaches are within 1–40% of the marginal energy (the Individual-Difference metric in Table 1). Scaphandre measures only CPU power, and is unable to attribute energy the disk-intensive dd function, and has a 100-130% difference vs. marginal for the rest of the functions on the desktop.

While marginal energy provides good ground truth, it may be *consistently* higher or lower than our profiler estimates. In Figure 8 the marginal energy of *all* desktop functions is higher, and all server functions is lower than our estimates. This is because the marginal footprints are affected by the CPU power states and power non-linearities. On the lower-utilization server with a large number of CPU cores (48), because of the latency in switching to lower power states, the additional fourth function can utilize the residual time of the higher power states. This results in the reported marginal footprints to be *lower*, as seen in the figure. Conversely, the desktop is at high utilization, and running the fourth function consistently puts the CPUs into a higher frequency state, which results in larger footprints since power is proportional to the square of the frequency. Thus, the specific power and workload characteristics in our marginal energy calculation contribute to the perceived discrepancy in the Individual-Difference metric in Figure 8. Finally, we note that the consistent over or under prediction is also naturally corrected by our Shapley footprints: the total power consumption error is less than 10% on average (discussed in next subsection in Figure 10).

We emphasize that individually comparing the energy estimates of different functions in Figure 8 provides an incomplete analysis of the profiler capabilities. For the individual energy footprints, how *proportional* the estimates are relative to the marginal energy is more important. Our cosine similarity metric captures this proportionality, and is shown for this workload in Table 3. *Using cosine similarity corrects for the consistent over or under prediction, and we see that our corrected individual energy footprints are within 98.4% and 99.8% of the ground truth.*

Note that invocations of the same function can have different arguments, which naturally leads to variance in the energy footprints and is the primary source of error, since our model uses function latency as the key feature. However, we are robust to such latency variance, as seen in Figure 9a, which shows a lack of any significant correlation. On the server with 40% load, the coefficient of variation of latency



(a) Function latency variance is uncorrelated with energy footprint error.

(b) Energy of identical functions can be clustered.

Figure 9: Energy footprints are robust to latency and hence the function input variation. Similar functions can be clustered based on their energy use.

is low, but the high idle energy results in the larger offset vs. marginal. The desktop has a 80% load and higher latency variance, but is closer to the marginal energy as a result. *Thus, even with variation in latency due to differences in function input, contention, etc., we provide accurate energy footprints.*

On the Jetson platform, we do not have access to a CPU power model—but the pure disaggregation approach still provides a high 99.2% cosine similarity. We are also able to measure the GPU function's footprint—highlighting its effectiveness on heterogeneous platforms without specialized power measurement instrumentation. Scaphandre uses the x86 RAPL counters, and is unable to work on the ARM Jetson platform. Its process-level direct attribution is unable to account for all the CPU power on the x86 server, and has an error of $10\times - 23\times$. Moreover, it has a high profiler overhead due to periodically scanning process info via `procfs`: its CPU consumption on the server is more than 5%, causing a 15 Watt (30%) increase in power consumption. For comparison, the *combined* CPU consumption of our profiler and Iluvatar is 3%. Scaphandre's high error on the server is also due to its high profiling overhead: the high latency of synchronously reading and disaggregating `procfs` information for more than 1000 processes (corresponding to function active and kept-alive containers) results in highly stale RAPL readings (several seconds) and inaccurate disaggregation.

Using noisy system-level power is one of our system's main features, and these results illustrate its robustness and accuracy. It provides more than an order of magnitude improvement in accuracy compared to existing process- and CPU-based tools, and is effective even on heterogeneous hardware.

7.2 Energy and Carbon Footprints

Having looked at the Individual-Error and the Cosine-Similarity metrics, we now present the evaluation of other metrics and properties. To illustrate generalizability, in some cases we show results on different workload traces from the one presented in the previous subsection. The remaining evaluation of the earlier four-function trace (`dd, image, AES, CNN`) is included in Figure 11 and 13.

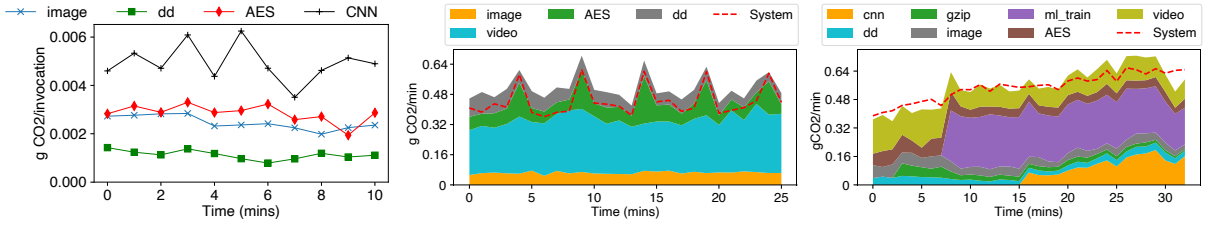
Full carbon footprint. The per-function operational carbon footprint (using the Shapley value approach) over a 15 minute interval is shown earlier in Figure 3. Because the server is under-utilized and has a high idle power (95 vs. 15 W), the shared operational emissions for all functions are close to 8x higher. The operational carbon footprint using Scaphandre power estimates misses the I/O-bound `dd` completely, and significantly underpredicts the other functions by an order of magnitude. *This highlights the importance of complete full-system footprints, since shared hardware and software components can be a large source of emissions.* These footprints can be used by both FaaS users and operators to quantify emissions. We now evaluate the feasibility to use them in addition to traditional resource based accounting and pricing.

Symmetry, a Shapley value property, requires that identical functions have similar footprints. We run a large number (20) of different functions, with each function belonging to the one of four classes (`image, json, ml_train, video`). This is different from the previous set of functions, and shows the generalizability and ability to handle very small (`json`) and large functions (`ml_train, video`). From the profiler's perspective, these are 20 different functions, and their footprints are shown in Figure 9b. We can see that the functions can be clustered based on their energy footprints, i.e., the functions running image processing have similar per-invocation energy, thus exhibiting the symmetry property.

Next, we look at the **Total-Error** from Table 1, which measures the difference in measured and the estimated total energy/carbon footprint (which is the aggregation of individual estimated footprints). This is also the *efficiency* property of Shapley values: we want all the energy and carbon accounted for. The (operational) carbon footprint of the functions from the previous workload used in Figure 8 is shown in Figure 10a. The figure shows the CO2 per invocation of the functions over time, and also illustrates the smoothing behavior of the Kalman filter.

Note that the actual footprint of a function also depends on the number of its invocations. We show this in the stacked-plots in Figure 10, which also illustrates a common use-case for profiling tools for identifying "top" carbon producers and their relative contributions. Figure 10b shows the carbon contribution of four functions in a "bursty" workload on the desktop, which also incorporates our largest function `video`. We can see that the Kalman filter is able to track the dynamics of the workload, since the sum of all function contributions closely matches the empirical operational CO2 emissions.

Another challenging workload is when the functions are dynamically introduced into the workload, and the "active



(a) CO2 footprints of functions over time. **(b) Bursty function invocations.** **(c) New functions being added.**

Figure 10: The misprediction of total carbon footprint is small, even for dynamic and non-stationary workloads.

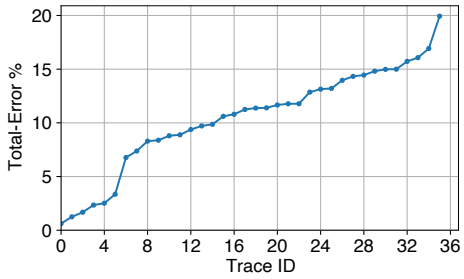


Figure 11: The Total-Error captures how much of the total server energy/carbon footprint is over or under-estimated, and is less than 20% across a range of workloads.

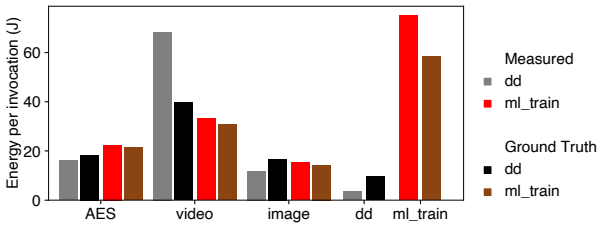


Figure 12: The three functions are co-located either with `dd` or `ml_train`. This choice has negligible impact on both the estimated footprints and ground-truth.

set" is dynamic. The total carbon breakdown of this workload is shown in Figure 10c, where we also see low Total-Error. As mentioned in Section 6.1, minimizing Total-Error is *not* our primary objective, since it can often reduce the accuracy (cosine similarity and other external validation metrics). Our Total-Error across 35 workloads (with different functions and IATs) on the three hardware platforms is shown in Figure 11. *Because of the Kalman filter and continuous footprint refinement, we see that the Total-Error is small, and less than 10% for more than 50% of the tested workload configurations.*

Noisy neighbors: *are function energy footprints impacted by co-located functions?* To answer this, we run functions with different "neighbors". Three functions (`image`, `AES`, and `video`) are run together either with `dd` or `ml_train` as the co-located function in Figure 12. These functions are very different: `dd` is short and disk-intensive, whereas `ml_train` is long and cpu-intensive. Inspite of these differences, the marginal energy in the two cases are nearly identical, and differs by at most 5%. The footprints are also similar, and vary by roughly 5–10% between the two cases. The lack of significant noisy neighbor effect reduces the variance in energy footprints.

For carbon-pricing, we want the footprints to be "stable" and have low variance. Figure 13 (right) shows the coefficient of variation ($CoV = \sigma(J)/E[J]$) for more than 50 workloads. The CoV depends on the measurement noise of the underlying hardware platform and the workload, and is the *precision*. Here, we focus on the energy variance, so that we can ignore the high operational carbon variance. The CoV is less than 0.3 on all three platforms for 60% of the traces, indicating feasibility of using energy footprints as an accounting and pricing measure. As noted before, the desktop energy variance is higher because its workloads are run at near-100% load.

Finally, we look at the **latency-normalized-variance** metric from Table 1. This metric is another proxy for the stability of the carbon pricing, since it compares against variance in currently used running-time based prices. Figure 13 (left) shows the CDF of the average normalized energy variance across all functions, for more than 50 workload traces across the three platforms. This ratio is less than 40 for more than 90% of the desktop and server workloads. Note that Iluvatar provides extremely low variance in latency which is $50\times - 100\times$ lower than OpenWhisk [50]. The variance in latency and pricing is also significant in public FaaS clouds [91, 104]. Thus, if were to use our footprints with the more widely used OpenWhisk, we achieve a latency-normalized-variance ratio of close to 1, indicating that the variance in energy price would be similar to the current variance in latency-based pricing. Thus, *our energy footprints have high precision (i.e., low variance) and can be used for carbon-based pricing.*

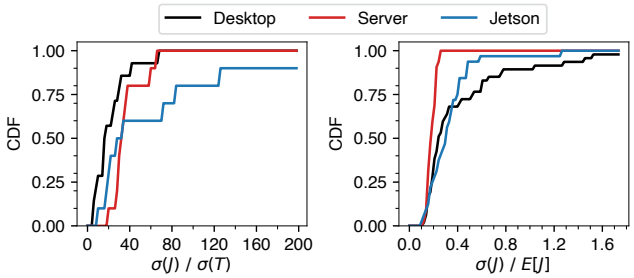


Figure 13: Variance of energy footprints is low, making carbon-based pricing feasible.

8 RELATED WORK

Our work is inspired and motivated by the quest to make energy as the first-class resource [44] in many environments.

Sustainable Computing. The challenge of reducing the carbon footprint of cloud systems and applications has led to "carbon-first" system designs [18, 79, 101]. The operational carbon footprint is computed by multiplying the energy consumption and the grid carbon intensity [6, 80], which has led to new energy tracking tools [28], but CPU power profiling continues to dominate. [65] introduces fine-grained NUMA-aware CPU energy measurement for individual applications. Tools for tracking the carbon footprint of AI applications mostly focus on large ML training batch jobs without multi-tenancy [19, 62]. The emissions due to AI training and inference is significant [111, 112], but our work focuses on the broader class of FaaS based applications.

Carbon Footprints. The *server-level* operational and embodied carbon footprints can be used for scheduling [61], load balancing [26, 79], and other resource management operations in distributed computing. Tools like CloudCarbon [1] also provide these VM-level estimates for public cloud VMs [38]. Disaggregating these for finer-grained applications is much more challenging as we have shown, and our footprints are also applicable to other multiplexed scenarios. Tools like carbond [93] can also be integrated into our framework for updating the operational grid intensity in real-time for more accurate operational carbon footprints. For embodied emissions, tools like ACT [59, 60] can provide the hardware's footprint based on its individual components (CPU type, storage and memory capacity, etc.). Carbon accounting at the grid level can also be tricky, with double counting due to power purchase agreements (PPAs) leading to incorrect estimates [78].

Energy Control. In the context of FaaS, [90] presents DAG scheduling for functions with a purely CPU-model based approach, but without any empirical power measurement or validation. DVFaaS [103] implements PID control for CPU frequency for minimizing latency QoS violations for function chains. More generally, a combination of hardware and software techniques for energy capping can be effective [119]. Due to hardware heterogeneity, we use a purely software approach for controlling system-wide power, and focus on empirical energy footprints of individual functions.

Fair Attribution. The problem of fairly sharing the energy consumption of shared resources occurs in many environments such as VM hosting [66, 68] and datacenter cooling [69, 108]. Shapley values [105] provide many desirable properties such as linearity and envy-freeness, and have also been used for energy accounting of mobile applications [39].

Mobile and embedded computing [45] faces similar energy measurement challenges. Disentangling shared OS and hardware energy consumption for applications has been done through tracing requests across various contexts and carefully attributing async tasks [24, 85, 87, 114]. Along with the uncertainty of the control plane, the highly dynamic and non-stationary nature of function workloads, and high degree of multiplexing makes such tracing challenging for FaaS.

9 CONCLUSION

Given the increasing importance of sustainability, our primary goal is to initiate a deeper investigation of energy metrology—especially full-system power measurements and their external validation. Our approach allows easy computation of carbon footprints. Using statistical disaggregation, Kalman filtering, and Shapley value principles, we provide full-spectrum carbon footprints with over 99% accuracy. Serverless workloads permit the use of marginal energy consumption for each function, which we use to develop new external validation metrics. This provides complete, validated, and full-system energy and carbon footprints for serverless functions, paving the way for carbon-based cloud pricing.

Acknowledgements. Abdul Rehman helped develop the power measurement and experimentation infrastructure. This paper has greatly benefited from the comments and feedback from our shepherd Dmitrii Ustiugov, and the anonymous conference reviewers. This work is supported in part by an NSF CAREER award CNS-2340722 and NSF grant OAC-2112606. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Cloud Carbon Footprint - An open source tool to measure and analyze cloud carbon emissions. https://www.cloudcarbonfootprint.org/.
[2] Cloudflare workers. https://blog.cloudflare.com/introducing-cloudflare-workers/.
[3] Data and code for faasmeter. https://github.com/COS-IN/faasmeter-socc-artifact.
[4] Google cloud carbon Footprint. https://cloud.google.com/carbon-footprint.
[5] Microsoft Sustainability Calculator helps enterprises analyze the carbon emissions of their IT infrastructure.

https://azure.microsoft.com/en-us/blog/microsoft-sustainability-calculator-helps-enterprises-analyze-the-carbon-emissions-of-their-it-infrastructure/.

[6] Watttime – The Power to Choose Clean Energy. https://www.watttime.org/.

[7] Apache OpenWhisk: Open Source Serverless Cloud Platform. https://openwhisk.apache.org/, 2020.

[8] AWS Lambda. https://aws.amazon.com/lambda/, 2020.

[9] Azure Functions. https://azure.microsoft.com/en-us/services/functions/, 2020.

[10] Google Cloud Functions. https://cloud.google.com/functions, 2020.

[11] Customer Carbon Footprint Tool | AWS News Blog. https://aws.amazon.com/blogs/aws/new-customer-carbon-footprint-tool/, Mar. 2022. Section: Announcements.

[12] Jetson AGX Orin Developer Kit User Guide. https://developer.nvidia.com/embedded/learn/jetson-agx-orin-devkit-user-guide/index.html, Mar. 2022.

[13] Acun, B., Lee, B., Kazhamiaka, F., Maeng, K., Chakkaravarthy, M., Gupta, U., Brooks, D., and Wu, C.-J. Carbon Explorer: A Holistic Approach for Designing Carbon Aware Datacenters, May 2022. arXiv:2201.10036 [cs, eess].

[14] Adzic, G., and Chatley, R. Serverless computing: economic and architectural impact. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (2017), pp. 884–889.

[15] Agache, A., Brooker, M., Iordache, A., Liguori, A., Neugebauer, R., Piwonka, P., and Popa, D.-M. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (2020), pp. 419–434.

[16] Agarwal, A., Sun, J., Noghabi, S., Iyengar, S., Badam, A., Chandra, R., Seshan, S., and Kalyanaraman, S. Redesigning Data Centers for Renewable Energy. *HotNets* (2021), 8.

[17] Anand, V., Xie, Z., Stolet, M., De Viti, R., Davidson, T., Karimipour, R., Alzayat, S., and Mace, J. The Odd One Out: Energy is not like Other Metrics. *HotCarbon 2022: 1st Workshop on Sustainable Computer Systems Design and Implementation* (July 2022).

[18] Anderson, T., Belay, A., Chowdhury, M., Cidon, A., and Zhang, I. Treehouse: A Case For Carbon-Aware Datacenter Software. *arXiv:2201.02120 [cs]* (Jan. 2022). arXiv: 2201.02120.

[19] Anthony, L. F. W., Kanding, B., and Selvan, R. Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models. *ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems* (July 2020). arXiv:2007.03051 [cs, eess, stat].

[20] Babakol, T., Canino, A., Mahmoud, K., Saxena, R., and Liu, Y. D. Calm energy accounting for multithreaded Java applications. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event USA, Nov. 2020), ACM, pp. 976–988.

[21] Bellosa, F. The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system* (2000), pp. 37–42.

[22] Bhagavathula, A., Han, L., and Gupta, U. Understanding the Implications of Uncertainty in Embodied Carbon Models for Sustainable Computing. *HotCarbon* (2024).

[23] Carreira, J., Fonseca, P., Tumanov, A., Zhang, A., and Katz, R. Cirrus: a Serverless Framework for End-to-end ML Workflows. In *Proceedings of the ACM Symposium on Cloud Computing - SoCC '19* (Santa Cruz, CA, USA, 2019), ACM Press, pp. 13–24.

[24] Carroll, A., and Heiser, G. An analysis of power consumption in a smartphone. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)* (2010).

[25] Castro, P., Ishakian, V., Muthusamy, V., and Slominski, A. The rise of serverless computing. *Communications of the ACM 62*, 12 (2019), 44–54.

[26] Chadha, M., Subramanian, T., Arima, E., Gerndt, M., Schulz, M., and Abboud, O. GreenCourier: Carbon-Aware Scheduling for Serverless Functions. In *Proceedings of the 9th International Workshop on Serverless Computing* (Bologna Italy, Dec. 2023), ACM, pp. 18–23.

[27] Chard, R., Babuji, Y., Li, Z., Skluzacek, T., Woodard, A., Blaiszik, B., Foster, I., and Chard, K. Funcx: A federated function serving fabric for science. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (New York, NY, USA, 2020), HPDC 20, Association for Computing Machinery, pp. 65–76.

[28] CNCF. Kepler: Kubernetes Efficient Power Level Exporter. https://sustainable-computing.io/.

[29] Colmant, M., Rouvoy, R., Kurpicz, M., Sobe, A., Felber, P., and Seinturier, L. The next 700 CPU power models. *Journal of Systems and Software 144* (Oct. 2018), 382–396.

[30] Cramton, P., MacKay, D. J., Ockenfels, A., and Stoft, S. *Global carbon pricing: the path to climate cooperation.* The MIT Press, 2017.

[31] Crawford, K. Generative AI's environmental costs are soaring — and mostly secret. *Nature 626*, 8000 (Feb. 2024), 693–693. Bandiera_abtest: a Cg_type: World View Publisher: Nature Publishing Group Subject_term: Machine learning, Computer science, Technology, Policy.

[32] Dell. Dell Desktop Carbon Footprint. https://i.dell.com/sites/content/corporate/corp-comm/en/Documents/dell-desktop-carbon-footprint-whitepaper.pdf.

[33] Dell. Dell server carbon footprint. https://i.dell.com/sites/content/corporate/corp-comm/en/documents/dell-server-carbon-footprint-whitepaper.pdf.

[34] Dell. Dell server carbon footprint. https://i.dell.com/sites/content/corporate/corp-comm/en/Documents/dell-desktop-carbon-footprint-whitepaper.pdf.

[35] Dell. Dell Server Carbon Footprint. https://i.dell.com/sites/content/corporate/corp-comm/en/documents/dell-server-carbon-footprint-whitepaper.pdf.

[36] Desrochers, S., Paradis, C., and Weaver, V. M. A Validation of DRAM RAPL Power Measurements. In *Proceedings of the Second International Symposium on Memory Systems* (Alexandria VA USA, Oct. 2016), ACM, pp. 455–470.

[37] Do, T., Rawshdeh, S., and Shi, W. pTop: A Process-level Power Profiling Tool. *HotPower* (2009), 5.

[38] Dodge, J., Prewitt, T., Tachet des Combes, R., Odmark, E., Schwartz, R., Strubell, E., Luccioni, A. S., Smith, N. A., DeCario, N., and Buchanan, W. Measuring the Carbon Intensity of AI in Cloud Instances. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul Republic of Korea, June 2022), ACM, pp. 1877–1894.

[39] Dong, M., Lan, T., and Zhong, L. Rethink energy accounting with cooperative game theory. In *Proceedings of the 20th annual international conference on Mobile computing and networking* (Maui Hawaii USA, Sept. 2014), ACM, pp. 531–542.

[40] Du, D., Liu, Q., Jiang, X., Xia, Y., Zang, B., and Chen, H. Serverless computing on heterogeneous computers. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne Switzerland, Feb. 2022), ACM, pp. 797–813.

[41] Fieni, G., Rouvoy, R., and Seinturier, L. SmartWatts: Self-Calibrating Software-Defined Power Meter for Containers. *arXiv:2001.02505 [cs]* (Jan. 2020). arXiv: 2001.02505.

[42] Fieni, G., Rouvoy, R., and Seiturier, L. SelfWatts: On-the-fly Selection of Performance Events to Optimize Software-defined Power

Meters. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (Melbourne, Australia, May 2021), IEEE, pp. 324–333.

[43] Flinn, J., and Satyanarayanan, M. Energy-aware adaptation for mobile applications. *ACM SIGOPS Operating Systems Review* (1999), 16.

[44] Flinn, J., and Satyanarayanan, M. PowerScope: a tool for profiling the energy usage of mobile applications. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications* (Feb. 1999), pp. 2–10.

[45] Fonseca, R., Dutta, P., Levis, P., and Stoica, I. Quanto: Tracking Energy in Networked Embedded Systems. *OSDI* (2008), 16.

[46] Fouladi, S., Romero, F., Iter, D., Li, Q., and Chatterjee, S. From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers. *USENIX Annual Technical Conference* (2019), 15.

[47] Fouladi, S., Wahby, R. S., Shacklett, B., Balasubramaniam, K. V., Zeng, W., Bhalerao, R., Sivaraman, A., Porter, G., and Winstein, K. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (2017), pp. 363–376.

[48] Foundation, G. S. Software Carbon Intensity (SCI) Specification. https://sci.greensoftware.foundation/.

[49] Foundation, G. S. SCI Specification Achieves ISO Standard Status. https://greensoftware.foundation/articles/sci-specification-achieves-iso-standard-status, Apr. 2024.

[50] Fuerst, A., Rehman, A., and Sharma, P. Ilúvatar: A fast control plane for serverless computing. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing* (June 2023), HPDC '23, Association for Computing Machinery.

[51] Fuerst, A., and Sharma, P. Faascache: Keeping serverless computing alive with greedy-dual caching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2021), ASPLOS 2021, Association for Computing Machinery, pp. 386–400.

[52] Garcia, C. Data Center Energy Use - AKCP Monitoring. https://www.akcp.com/blog/the-real-amount-of-energy-a-data-center-use/, July 2023.

[53] Ge, R., Feng, X., Song, S., Chang, H.-C., Li, D., and Cameron, K. W. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems 21*, 5 (May 2010), 658–671. Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[54] Ghanei, F., Tipnis, P., Marcus, K., Dantu, K., Ko, S., and Ziarek, L. OS-based Resource Accounting for Asynchronous Resource Use in Mobile Systems. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design* (San Francisco Airport CA USA, Aug. 2016), ACM, pp. 296–301.

[55] Ghanei, F., Tipnis, P., Marcus, K., Dantu, K., Ko, S. Y., and Ziarek, L. OS-Based Energy Accounting for Asynchronous Resources in IoT Devices. *IEEE Internet of Things Journal 6*, 3 (June 2019), 5841–5852. Conference Name: IEEE Internet of Things Journal.

[56] Guan, X., Bashir, N., Irwin, D., and Shenoy, P. WattScope: Non-intrusive application-level power disaggregation in datacenters. *Performance Evaluation 162* (Nov. 2023), 102369.

[57] Gujarati, A., Karimi, R., Alzayat, S., Hao, W., Kaufmann, A., Vigfusson, Y., and Mace, J. Serving {DNNs} like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (2020), pp. 443–462.

[58] Guo, L., Xu, T., Xu, M., Liu, X., and Lin, F. X. Power sandbox: power awareness redefined. In *Proceedings of the Thirteenth EuroSys Conference* (Porto Portugal, Apr. 2018), ACM, pp. 1–15.

[59] Gupta, U., Elgamal, M., Hills, G., Wei, G.-Y., Lee, H.-H. S., Brooks, D., and Wu, C.-J. ACT: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York New York, June 2022), ACM, pp. 784–799.

[60] Gupta, U., Kim, Y. G., Lee, S., Tse, J., Lee, H.-H. S., Wei, G.-Y., Brooks, D., and Wu, C.-J. Chasing Carbon: The Elusive Environmental Footprint of Computing, Oct. 2020. arXiv:2011.02839 [cs].

[61] Hanafy, W. A., Liang, Q., Bashir, N., Irwin, D., and Shenoy, P. Carbonscaler: Leveraging cloud workload elasticity for optimizing carbon-efficiency. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 7*, 3 (2023), 1–28.

[62] Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., and Pineau, J. Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. *Journal of Machine Learning Research 21* (2020), 1–43.

[63] Hong, S., and Kim, H. An integrated GPU power and performance model. In *Proceedings of the 37th annual international symposium on Computer architecture* (New York, NY, USA, June 2010), ISCA '10, Association for Computing Machinery, pp. 280–289.

[64] Hubblo. Scaphandre. https://github.com/hubblo-org/scaphandre, July 2023.

[65] Hè, H., Friedman, M., and Rekatsinas, T. EnergAt: Fine-Grained Energy Attribution for Multi-Tenancy. *HotCarbon* (2023).

[66] Islam, M. A., and Ren, S. A new perspective on energy accounting in {Multi-Tenant} data centers. In *USENIX Workshop on Cool Topics on Sustainable Data Centers (CoolDC 16)* (2016).

[67] Jay, M., Ostapenco, V., Lefevre, L., Trystram, D., Orgerie, A.-C., and Fichel, B. An experimental comparison of software-based power meters: focus on CPU and GPU. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (May 2023), pp. 106–118.

[68] Jiang, W., Liu, F., Tang, G., Wu, K., and Jin, H. Virtual Machine Power Accounting with Shapley Value. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (Atlanta, GA, USA, June 2017), IEEE, pp. 1683–1693.

[69] Jiang, W., Ren, S., Liu, F., and Jin, H. Non-IT Energy Accounting in Virtualized Datacenter. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (Vienna, July 2018), IEEE, pp. 300–310.

[70] Kandiah, V., Peverelle, S., Khairy, M., Pan, J., Manjunath, A., Rogers, T. G., Aamodt, T. M., and Hardavellas, N. AccelWattch: A Power Modeling Framework for Modern GPUs. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, Oct. 2021), MICRO '21, Association for Computing Machinery, pp. 738–753.

[71] Khan, K. N., Hirki, M., Niemi, T., Nurminen, J. K., and Ou, Z. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems 3*, 2 (June 2018), 1–26.

[72] Khan, K. N., Nyback, F., Ou, Z., Nurminen, J. K., Niemi, T., Eulisse, G., Elmer, P., and Abdurachmanov, D. Energy Profiling Using IgProf. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (Shenzhen, China, May 2015), IEEE, pp. 1115–1118.

[73] Kim, J., and Lee, K. FunctionBench: A Suite of Workloads for Serverless Cloud Function Service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)* (July 2019), pp. 502–504. ISSN: 2159-6182.

[74] Klenert, D., Mattauch, L., Combet, E., Edenhofer, O., Hepburn, C., Rafaty, R., and Stern, N. Making carbon pricing work for

citizens. *Nature Climate Change 8*, 8 (2018), 669–677.

[75] Lee, S., Kim, H., Park, S., Kim, S., Choe, H., and Yoon, S. CloudSocket: Fine-Grained Power Sensing System for Datacenters. *IEEE Access 6* (2018), 49601–49610. Conference Name: IEEE Access.

[76] Li, A., Liu, S., and Ding, Y. Uncertainty-Aware Decarbonization for Datacenters. *HotCarbon* (2024).

[77] Mahgoub, A., Yi, E. B., Shankar, K., Minocha, E., Elnikety, S., Bagchi, S., and Chaterji, S. WISEFUSE: Workload Characterization and DAG Transformation for Serverless Workflows. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 6*, 2 (May 2022), 1–28.

[78] Maji, D., Bashir, N., Irwin, D., Shenoy, P., and Sitaraman, R. K. Untangling Carbon-free Energy Attribution and Carbon Intensity Estimation for Carbon-aware Computing, Feb. 2024. arXiv:2308.06680 [cs].

[79] Maji, D., Pfaff, B., P R, V., Sreenivasan, R., Firoiu, V., Iyer, S., Josephson, C., Pan, Z., and Sitaraman, R. K. Bringing Carbon Awareness to Multi-cloud Application Delivery. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems* (jul 2023), ACM, pp. 1–6.

[80] Maji, D., Shenoy, P., and Sitaraman, R. K. Carboncast: multi-day forecasting of grid carbon intensity. In *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (2022), pp. 198–207.

[81] Maschi, F., Korolija, D., and Alonso, G. Serverless FPGA: Work-In-Progress. In *Proceedings of the 1st Workshop on SErverless Systems, Applications and MEthodologies* (Rome Italy, May 2023), ACM, pp. 1–4.

[82] Meinhold, R. J., and Singpurwalla, N. D. Understanding the Kalman filter. *The American Statistician 37*, 2 (1983), 123–127.

[83] Mukhanov, L., Petoumenos, P., Wang, Z., Parasyris, N., Nikolopoulos, D. S., De Supinski, B. R., and Leather, H. ALEA: A Fine-Grained Energy Profiling Tool. *ACM Transactions on Architecture and Code Optimization 14*, 1 (Apr. 2017), 1–25.

[84] Noureddine, A., Rouvoy, R., and Seinturier, L. A review of energy measurement approaches. *ACM SIGOPS Operating Systems Review 47*, 3 (2013), 42–49.

[85] Oliner, A. J., Iyer, A. P., Stoica, I., Lagerspetz, E., and Tarkoma, S. Carat: Collaborative energy diagnosis for mobile devices. In *Proceedings of the 11th ACM conference on embedded networked sensor systems* (2013), pp. 1–14.

[86] Ournani, Z., Belgaid, M. C., Rouvoy, R., Rust, P., Penhoat, J., and Seinturier, L. Taming energy consumption variations in systems benchmarking. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (2020), pp. 36–47.

[87] Pathak, A., Hu, Y. C., and Zhang, M. Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems* (2012), pp. 29–42.

[88] Pearson, K. A., Griffith, C. A., Zellem, R. T., Koskinen, T. T., and Roudier, G. M. Ground-based spectroscopy of the exoplanet xo-2b using a systematic wavelength calibration. *The Astronomical Journal 157*, 1 (2019), 21.

[89] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[90] Rastegar, S. H., Shafiei, H., and Khonsari, A. EneX: An Energy-Aware Execution Scheduler for Serverless Computing. *IEEE Transactions on Industrial Informatics* (2023), 1–13.

[91] Schirmer, T., Japke, N., Greten, S., Pfandzelter, T., and Bermbach, D. The Night Shift: Understanding Performance Variability of Cloud Serverless Platforms. In *Proceedings of the 1st Workshop on SErverless Systems, Applications and MEthodologies* (Rome Italy, May 2023), ACM, pp. 27–33.

[92] Schleier-Smith, J., Sreekanti, V., Khandelwal, A., Carreira, J., Yadwadkar, N. J., Popa, R. A., Gonzalez, J. E., Stoica, I., and Patterson, D. A. What serverless computing is and should become: The next phase of cloud computing. *Commun. ACM 64*, 5 (Apr. 2021), 76–84.

[93] Schmidt, A., Stock, G., Ohs, R., Gerhorst, L., Herzog, B., and Hönig, T. carbond: An Operating-System Daemon for Carbon Awareness.

[94] Schmitt, N., Iffländer, L., Bauer, A., and Kounev, S. Online Power Consumption Estimation for Functions in Cloud Applications. In *2019 IEEE International Conference on Autonomic Computing (ICAC)* (June 2019), pp. 63–72. ISSN: 2474-0756.

[95] Scikit-learn. Support vector regression. https://scikit-learn/stable/modules/generated/sklearn.svm.SVR.html.

[96] Shahrad, M., Fonseca, R., Goiri, , Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M., and Bianchini, R. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX annual technical conference (USENIX ATC 20)* (2020), pp. 205–218.

[97] Sharma, P. Challenges and opportunities in sustainable serverless computing. *HotCarbon 2022: 1st Workshop on Sustainable Computer Systems Design and Implementation* (July 2022).

[98] Sharma, P., Pegus, P. I., Irwin, D., Shenoy, P., Goodhue, J., and Culbert, J. Design and operational analysis of a green data center. *IEEE Internet Computing 21*, 4 (2017), 16–24.

[99] Shen, K., Shriraman, A., Dwarkadas, S., Zhang, X., and Chen, Z. Power containers: an OS facility for fine-grained power and energy management on multicore servers. *ASPLOS* (2013), 12.

[100] Siddik, M. A. B., Shehabi, A., and Marston, L. The environmental footprint of data centers in the united states. *Environmental Research Letters 16*, 6 (may 2021), 064017.

[101] Souza, A., Bashir, N., Murillo, J., Hanafy, W., Liang, Q., Irwin, D., and Shenoy, P. 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* (2023), pp. 252–265.

[102] Sundararajan, M., and Najmi, A. The many shapley values for model explanation. In *International conference on machine learning* (2020), PMLR, pp. 9269–9278.

[103] Tzenetopoulos, A., Masouros, D., Soudris, D., and Xydis, S. DVFaaS: Leveraging DVFS for FaaS workflows. *IEEE Computer Architecture Letters* (2023), 1–4.

[104] Ustiugov, D., Amariucai, T., and Grot, B. Analyzing Tail Latency in Serverless Clouds with STeLLAR. In *2021 IEEE International Symposium on Workload Characterization (IISWC)* (Storrs, CT, USA, Nov. 2021), IEEE, pp. 51–62.

[105] Vergara, E. J., Nadjm-Tehrani, S., and Asplund, M. Sharing the Cost of Lunch: Energy Apportionment Policies. In *Proceedings of the 11th ACM Symposium on QoS and Security for Wireless and Mobile Networks* (Cancun Mexico, Nov. 2015), ACM, pp. 91–97.

[106] Wagner, L., Mayer, M., Marino, A., Soldani Nezhad, A., Zwaan, H., and Malavolta, I. On the Energy Consumption and Performance of WebAssembly Binaries across Programming Languages and Runtimes in IoT. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering* (Oulu Finland, June 2023), ACM, pp. 72–82.

[107] Wang, B., Ali-Eldin, A., and Shenoy, P. Lass: Running latency sensitive serverless computations at the edge. In *Proceedings of the 30th international symposium on high-performance parallel and distributed*

computing (2021), pp. 239–251.

[108] Wang, R., Van Le, D., Tan, R., Wong, Y.-W., and Wen, Y. Real-Time Cooling Power Attribution for Co-Located Data Center Rooms with Distinct Temperatures. In *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (Virtual Event Japan, Nov. 2020), ACM, pp. 190–199.

[109] Welch, G., Bishop, G., et al. *An introduction to the Kalman filter*. Chapel Hill, NC, USA, 1995.

[110] Winter, E. The shapley value. *Handbook of game theory with economic applications 3* (2002), 2025–2054.

[111] Wu, C.-J., Acun, B., Raghavendra, R., and Hazelwood, K. Beyond Efficiency: Scaling AI Sustainably. *IEEE Micro* (2024), 1–8.

[112] Wu, C.-J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Behram, F. A., Huang, J., Bai, C., Gschwind, M., Gupta, A., Ott, M., Melnikov, A., Candido, S., Brooks, D., Chauhan, G., Lee, B., Lee, H.-H. S., Akyildiz, B., Balandat, M., Spisak, J., Jain, R., Rabbat, M., and Hazelwood, K. Sustainable AI: Environmental Implications, Challenges and Opportunities. *Proceedings of the 5th MLSys Conference, Santa Clara, CA, USA* (2022), 19.

[113] Xu, F., Qin, Y., Chen, L., Zhou, Z., and Liu, F. λ-dnn : Achieving predictable distributed dnn training with serverless architectures. *IEEE Transactions on Computers* (2021).

[114] Yoon, C., Kim, D., Jung, W., Kang, C., and Cha, H. AppScope: Application Energy Metering Framework for Android Smartphones using Kernel Activity Monitoring. *USENIX ATC* (2012), 14.

[115] Yu, M., Wang, A., Chen, D., Yu, H., Luo, X., Li, Z., Wang, W., Chen, R., Nie, D., and Yang, H. FaaSwap: SLO-Aware, GPU-Efficient Serverless Inference via Model Swapping, June 2023. arXiv:2306.03622 [cs].

[116] Zeng, H., Ellis, C. S., Lebeck, A. R., and Vahdat, A. ECOSystem: Managing Energy as a First Class Operating System Resource. *ASPLOS* (2002), 10.

[117] Zeng, H., Ellis, C. S., Lebeck, A. R., and Vahdat, A. Currentcy: A Unifying Abstraction for Expressing Energy Management Policies. *USENIX ATC* (2003), 14.

[118] Zhang, H., and Hoffmann, H. A Quantitative Evaluation of the RAPL Power Control System. *Feedback computing* (2015), 6.

[119] Zhang, H., and Hoffmann, H. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. *ASPLOS* (2016), 15.

[120] Zhang, X., Shen, Z., Xia, B., Liu, Z., and Li, Y. Estimating Power Consumption of Containers and Virtual Machines in Data Centers. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)* (Sept. 2020), pp. 288–293. ISSN: 2168-9253.