Designing Cloud Servers for Lower Carbon

Jaylen Wang*, Daniel S. Berger†,‡, Fiodar Kazhamiaka†, Celine Irvene†, Chaojie Zhang†, Esha Choukse†, Kali Frost†, Rodrigo Fonseca†, Brijesh Warrier†, Chetan Bansal†, Jonathan Stern†, Ricardo Bianchini†, Akshitha Sriraman*

*Carnegie Mellon University †Microsoft ‡University of Washington

***Abstract*—To mitigate climate change, we must reduce carbon emissions from hyperscale cloud computing. We find that cloud compute servers cause the majority of emissions in a general-purpose cloud. Thus, we motivate designing carbon-efficient compute server SKUs, or *GreenSKUs*, using recently-available low-carbon server components. To this end, we design and build three *GreenSKUs* using low-carbon components, such as energy-efficient CPUs, reused old DRAM via CXL, and reused old SSDs.**

We detail several challenges that limit *GreenSKUs*' carbon savings at scale and may prevent their adoption by cloud providers. To address these challenges, we develop a novel methodology and associated framework, *GSF* (*GreenSKU Framework*), that enables a cloud provider to systematically evaluate a *GreenSKU*'s carbon savings at scale. We implement *GSF* within Microsoft Azure's production constraints to evaluate our three *GreenSKUs*' carbon savings. Using *GSF*, we show that our most carbon-efficient *GreenSKU* reduces emissions per core by 28% compared to currently-deployed cloud servers. When designing *GreenSKUs* to meet applications' performance requirements, we reduce emissions by 15%. When incorporating overall data center overheads, our *GreenSKU* reduces Azure's net cloud emissions by 8%.

I. Introduction

To mitigate climate change, we must reduce carbon emissions from Information and Communication Technology (ICT), which can cause 20% of global carbon emissions by 2030 [76]. Historically, ICT's emissions reduced when sharing compute resources using cloud computing [36]. However, today, projections show that significant emissions arise from cloud computing itself, due to its massive growth [73], [76]. Thus, it is now critical to reduce cloud computing's emissions [38], [47]. Indeed, major cloud providers have set aggressive decarbonization deadlines, targeting significant emissions reductions by 2030 [10], [27].

To reduce ICT's emissions from cloud computing, we must reduce the cloud's *operational emissions* (e.g., from producing electricity to run data centers) and *embodied emissions* (e.g., from semiconductor fabs that make server components) [65]. Historically, cloud computing's operational emissions exceeded its embodied emissions. To reduce operational emissions, hyperscale cloud providers improve energy efficiency [45], [52], [62], [79], [96], [118], [132], [134] and use more renewable energy [35], [48], [65]. Today, the decrease in operational emissions due to such solutions has caused embodied emissions to account for 50%–82% of cloud emissions [65], [88]. Thus, it is crucial to reduce both emission types.

To reduce cloud computing's operational and embodied emissions, we identify designing carbon-efficient cloud compute server Stock Keeping Units (SKUs) as a promising solution. Server SKU design is the process by which existing hardware components are selected and composed into servers. Typically, cloud providers design compute server SKUs to meet performance and cost goals. To reduce emissions, we introduce a new way of designing carbon-efficient compute server SKUs, or "*GreenSKUs*," that trade off performance for lower carbon.

We find that designing and deploying carbon-efficient *GreenSKUs* is promising for four reasons. First, we show that compute servers cause the majority of cloud emissions, and their design directly impacts both embodied and operational emissions. Indeed, with a ∼six-year lifetime for cloud servers [88], design choices made in the next two years directly affect the industry's 2030 carbon goals. Second, it is challenging for cloud providers to rely on manufacturing's decarbonization, as many manufacturers have decarbonization targets that lag behind cloud providers' targets by over a decade [88]. Third, as we will show, cloud servers are often underutilized [50], making a case for designing servers that right-size performance to save emissions. Fourth, *GreenSKU* design and deployment is more feasible today due to the availability of carbon-efficient commodity server components, e.g., energy-efficient cores [4].

Due to *GreenSKUs*' promise, we design and build three *GreenSKUs* using low-carbon components that mitigate cloud compute servers' key sources of operational and embodied emissions. Our *GreenSKUs* incrementally incorporate three low-carbon components: energy-efficient high-thread-count CPUs [4], reuse of old DRAM with Compute Express Link (CXL) [103], and reuse of old Solid State Drives (SSDs).

While *GreenSKUs* promise carbon savings, we demonstrate several challenges that limit cloud providers from practically deploying them at scale. First, we find that a *GreenSKU* may compromise performance. For example, a *GreenSKU* built with many energy-efficient, i.e., *efficient*, cores [4] typically has lower single-thread performance. In practice, only some cloud applications, e.g., those not bound by single-thread performance, will run on such a *GreenSKU*. Thus, it is challenging to design *GreenSKUs* while effectively navigating their emissions vs. performance tradeoff, to justify deploying them at scale. Second, a *GreenSKU*'s operational and embodied emissions can have complex tradeoffs. For example, reusing an older server component can reduce embodied emissions, but may increase operational emissions due to the component's poor energy efficiency [64]. Third, each new SKU adds operational complexity and cost. Thus, while cloud applications are highly diverse, cloud providers must limit how many SKU types they deploy. We refer to these factors (e.g., performance) that impact whether a *GreenSKU* can practically be deployed at scale, as the "adoption" of the *GreenSKU*. It is critical and challenging for cloud providers to identify which *GreenSKU* designs are adoptable, i.e., able to save carbon while meeting diverse applications' deployment requirements at scale.

To address these challenges, we develop a novel methodology and framework, *GSF*, to enable cloud providers to evaluate a *GreenSKU*'s carbon savings in the cloud. *GSF* systematically considers the major factors that influence a *GreenSKU*'s benefits at scale. *GSF*'s components model each major factor, such as modeling a *GreenSKU*'s at-scale impact on carbon, performance, maintenance, server adoption, resource allocation, and server fragmentation. *GSF* abstracts these components' relationships from how a cloud provider implements each component, enabling a cloud provider to flexibly use *GSF* in their cloud to estimate a *GreenSKU*'s net carbon savings.

We implement *GSF* within Microsoft Azure's production constraints to evaluate our three *GreenSKUs*' carbon savings at scale. Our carbon model reflects Azure's data center design. We study representative applications in Azure to identify those that run effectively on our *GreenSKUs*. We also simulate workload packing on our *GreenSKUs* under production constraints.

Using our *GSF* implementation, we show that our *GreenSKUs* reduce carbon emissions per core by 28% compared to currently-deployed cloud servers at Azure. When deploying *GreenSKUs* in a way that meets applications' performance goals, we reduce emissions by 15%. Finally, when incorporating overall data center overheads, our *GreenSKUs* reduce net cloud emissions by 8%, which is a significant reduction at scale.

In summary, we contribute:

- A demonstration of the opportunity to significantly reduce cloud emissions by designing and deploying carbon-efficient server hardware, i.e., *GreenSKUs*, at scale.
- The development of three new *GreenSKU* prototypes with carbon-efficient server components.
- A study of the challenges that limit *GreenSKUs*' adoption and carbon savings at scale.
- A novel methodology and associated framework, *GSF*[1], that helps cloud providers to systematically evaluate a *GreenSKU*'s carbon savings at scale.
- An evaluation of our *GreenSKUs* by using *GSF* within a leading cloud provider's production constraints, to demonstrate our *GreenSKUs*' carbon savings.

We motivate *GreenSKUs* in §II and describe our *GreenSKU* prototypes in §III. We detail *GSF* in §IV and its implementation in §V. We use *GSF* to evaluate our *GreenSKUs* in §VI. We discuss *GreenSKUs*' practicality in §VII, open questions in §VIII, and related work in §IX. We conclude in §X.

II. Opportunities and Challenges with Designing GreenSKUs

Cloud platforms offer diverse services including infrastructure, platform, and software as a service. In a general-purpose public cloud like Azure, applications typically run within Virtual Machines (VMs) on *compute servers* [66]. While there are many compute server types, the most common types today use a general-purpose x86 or ARM CPU, significant DDR5 memory capacity, a few NVMe SSDs, and a Network Interface Card (NIC). Additionally, storage services, such as object stores, are hosted on dedicated *storage servers* that contain arrays of hard disks, with fewer computational resources.

We focus on reducing emissions for general-purpose cloud compute servers that form a large portion of Azure's global fleet and carbon emissions. Although we do not focus on heterogeneous compute, our work may be used to develop *GreenSKUs* for heterogeneous platforms in future work.

In this section, we first detail a breakdown of a cloud's overall emissions. We then discuss how cloud servers are designed and utilized today. Finally, we introduce key design goals for low-carbon compute servers, i.e., *GreenSKUs*.

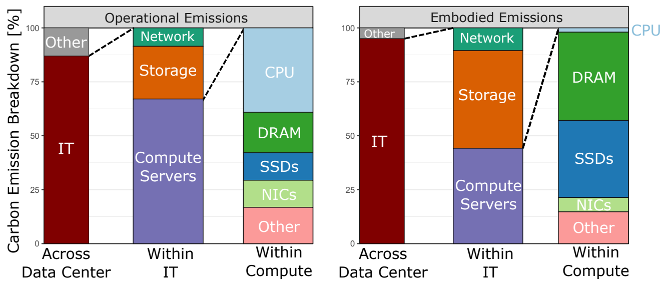


Fig. 1. Carbon breakdown of general-purpose data centers at Azure.

Sources of data center carbon emissions. To understand *GreenSKU* design opportunities, we analyze how different server types and their hardware components contribute to a general-purpose cloud's emissions. Similar to prior work [64], [65], we adopt the greenhouse gas protocol's definition of emissions from Scope 1 (i.e., direct emissions), Scope 2 (i.e., indirect emissions from consuming power), and Scope 3 (i.e., indirect emissions from manufacturing and transporting procured products like servers). We refer to Scope 1 emissions as *direct emissions*, Scope 2 emissions as *operational emissions*, and Scope 3 emissions as *embodied emissions*, matching prior work's terminology [64], [65], [115]. With these definitions, we use our carbon model (detailed later in §IV and §V) to estimate a cloud data center's carbon emissions breakdown.

We estimate operational emissions using power traces from Azure. To calculate embodied emissions, we estimate raw materials from vendor manifests, measure devices' silicon area, and use averaged emissions for manufacturing processes reported in industry datasets such as IMEC [21] and Makersite [25]. Our embodied emission estimation counts emissions once per component across the supply chain, making it directly comparable to operational emissions. The sum of all three emission types, i.e., direct, operational, and embodied, defines a *cloud's total carbon emissions*.

We find that direct emissions are negligible, as they mainly arise from backup diesel generators [78]. Thus, we only study operational and embodied emissions. The relative weight

[1]We open-source an implementation of *GSF*'s carbon model [18], [123], which is available at https://github.com/Azure/AzurePublicDataset

between the two differs between data centers due to local energy mixes [35]. Hence, we present these emission breakdowns separately and then discuss their relative weights.

Fig. 1 shows the breakdown of operational and embodied emissions in Azure's cloud data centers. At the data center level, IT equipment dominates operational and embodied emissions. The rest arises due to cooling, power distribution, and building emissions. Within IT, there are three dominant server types: compute, storage, and network servers. Of these, compute servers consume most of the power, while storage servers have a large embodied footprint and consume relatively less power.

We further attribute compute servers' emissions to their composite hardware components. This breakdown is influenced by server generation and vendor; we focus on a current-generation AMD Genoa server [6]. We find that the largest carbon contributors differ between the operational and embodied emissions breakdown. For operational, CPUs have the largest impact with the remaining emissions distributed across DRAM, SSDs, NICs, and other components like fans. For embodied, DRAM and SSDs dominate emissions, mainly due to their high capacities and large silicon area: our server has 12 DIMMs and 6 SSDs, each containing many chips.

In our accounting, we only count renewable energy purchases that match a data center's location. We find that most data centers use 40%–80% renewable energy at Azure. This renewable energy mix leads to operational emissions accounting for about 58% of total carbon emissions, implying that compute servers account for 57% of data center emissions. Within compute servers, the top three component contributors are DRAM (35% total contribution), SSDs (28%), and CPUs (24%). With a hypothetical 100% renewable energy mix, operational emissions would account for 9% of data center emissions and compute servers for 44% of data center emissions. These results motivate the urgent need to reduce compute servers' emissions.

Today's performance-focused cloud server design. To enable carbon-aware server design, we must first understand the conventional server design process and objectives. In the last decade, to design cloud servers, cloud providers have followed the *faster-at-similar-price* (FSP) business model. With FSP, the provider introduces a *faster VM generation* at roughly the same price every few years. Faster implies a target X% higher per-core performance on a fixed benchmark set [3], [9], [17].

Unfortunately, achieving these per-core speed-ups is increasingly challenging in today's age of limited technology scaling [54]. For example, consecutive Intel VM generations on Amazon AWS (M6i and M7i) achieve X=15% higher performance per core at almost the same price per core [28]. However, achieving this higher performance forces AWS to use 48-core Intel Xeon CPUs [8], [28], which maximize power, cache, and memory bandwidth per core. Intel also offers 60-core Xeon CPUs in the same generation [2], which have significantly lower carbon emissions per core, due to their lower Thermal Design Power (TDP) per core (i.e., 5.83 Watts vs. 6.25 Watts). However, 60-core Xeon CPUs would not have achieved the required 15% per-core speedup. In this paper, we denote SKUs constructed within the FSP model as *baseline* SKUs. Typically, there is one baseline SKU for every CPU generation.

Cloud customers underutilize cloud servers. There is significant evidence that cloud users frequently do not utilize high per-core performance, thereby exacerbating emissions. First, prior work has extensively documented that cloud CPUs are severely underutilized [37], [50], [63], [88], [116], [121], [128]. For example, 75% of Azure VMs exhibit less than 25% CPU utilization [50]. Of note, this low CPU utilization persists despite significant advances in cluster scheduling, which allocate up to 85% of cores to VMs [39], [66], [116], [121]. However, VMs frequently do not use these allocated cores, which causes underutilization [50].

Other cloud server resources are similarly underutilized. For example, average memory bandwidth utilization in Azure is only $\sim$15% [41]. We find similar underutilization of SSD IOPS and bandwidth. While underutilization occurs on most servers, some customers' VMs utilize all available performance. Thus, the demand for higher per-core performance exists and is likely to continue to exist on some servers.

Second, we find that customers continue to use old VM generations even when higher-performing generations are available. We even see *new* deployments of VMs that are multiple generations behind the latest. Thus, the core-hours of old VM generations continue to grow, which can require running old VM generations on new servers, where components like the CPU may be under-clocked to match old servers' performance. Maintainability and compatibility in large code bases may take precedence over new VMs' higher performance.

Design goals and constraints for low-carbon servers. Our goal is to reduce net cloud emissions by designing low-carbon *GreenSKUs*. To this end, we must address three design goals.

(D1) Account for tradeoffs between operational vs. embodied emissions. Operational and embodied emissions have complex tradeoffs [88]. For example, reusing components reduces embodied [64], but may increase operational emissions due to worse energy efficiency. Thus, a *GreenSKU* that reduces cloud emissions must balance operational and embodied emissions.

(D2) Account for data center impacts and side effects from introducing GreenSKUs. Adding server SKUs to a data center fleet can have side effects that may increase emissions. For example, cloud providers deploy extra servers as a buffer to absorb spikes in demand growth, to account for the time it takes to deploy additional servers. Offering numerous server options can reduce demand multiplexing among applications, which may increase the variance in demand growth for each option. Thus, adding many server options may require larger buffers, increasing emissions. We detail this challenge in §IV-D.

(D3) Model performance and adoption impacts for user applications. We must design a server that will be widely adopted and in a way that reduces net emissions. This design is challenging as many low-carbon components' performance is lower than the servers built under the FSP model (see §III).

III. Our GreenSKU Prototypes

The design space for *GreenSKUs* is large. We show the practicality and effectiveness of building *GreenSKUs* that target

CPU Characteristic	Bergamo	Rome(Gen 1)	Milan (Gen 2)	Genoa (Gen 3)
Cores per socket	128	64	64	80
Max core freq. (GHz)	3.0	3.0	3.7	3.7
LLC size per socket (MiB)	256	256	256	384
TDP (W)	350	240	280	300-350

TABLE I
COMPARING BASELINE AMD CPUS TO THE EFFICIENT BERGAMO CPU.

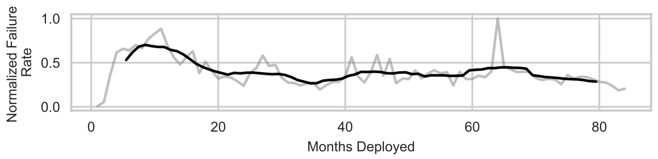


Fig. 2. Moving average (black) of raw (gray) normalized failure rates vs. DDR4 DIMMs' deployment time in production. Failure rates tend to stay constant over a 7-year period.

the top three carbon contributors in compute servers: CPUs, DRAM, and SSDs (Fig. 1), which cause 67% of a server's net emissions. We build *GreenSKUs* with efficient CPUs and reduce embodied emissions by reusing DRAM and SSDs from decommissioned servers. These designs are deployable today.

Other *GreenSKU* designs that reuse NICs or use low-power DRAM may be feasible, but, yield low returns today. These designs can help target residual emissions for a potential second-generation *GreenSKU*. Future *GreenSKU* designs may also include optimizations that are at a research stage today (e.g., leaner processor microarchitectures [65]). We design *GSF* to flexibly consider various such *GreenSKU* designs.

Low-carbon components. We use three low-carbon components in our prototype. First, we use efficient CPUs. Power-efficient cores, which enable very high thread and core counts for scale-out applications, are now widely available [102]. They include Ampere's 192-core ARM CPUs [7], AMD's 128-core/256-thread x86 Bergamo CPUs [4], and Intel's 288-core x86 Sierra Forest CPUs [22]. Since these CPUs have 40%–60% more cores at comparable power consumption to mainstream CPUs, they significantly reduce operational emissions per core.

We specifically choose AMD Bergamo in our *GreenSKU* prototype, as it provides the highest thread-count option on the market today and has full support for Type 3 CXL devices (CXL.mem), making it practically deployable in our cloud.

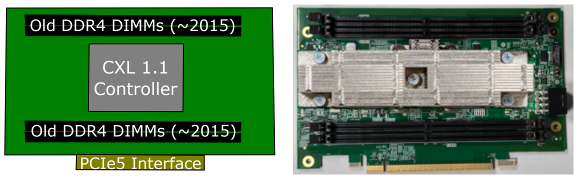


Fig. 3. Old DDR4, deployed starting in 2015, can be attached in new servers with a CXL controller card that attaches to PCIe5.

Next, we consider reused DRAM. We find that it is practical to reuse old DRAM in new servers. In a review of old server life cycles at Azure, we find that numerous old DDR4 DIMMs can be reused due to their host servers reaching the end of their deployment. Critically, these old DIMMs show no sign of increasing failure rates. Prior work notes that DRAM shows no signs of aging within five years [106]. Fig. 2 shows failure rates for DDR4 DIMMs in Azure over a 7-year deployment period. After an initial period of higher Annual Failure Rates (AFRs), they tend to stay constant [111]. While we do not yet have at-scale data for beyond 7 years, internal accelerated aging studies show that AFRs remain flat beyond 12 years.

Historically, reusing old DRAM was challenging as DDR generations are not backward compatible, i.e., it was infeasible to attach DDR4 in currently-deployed DDR5 servers. However, with the wide availability of CXL [103], old DDR4 can be attached to CXL controllers, which, in turn, are attached to the modern PCIe5 interface. Fig. 3 shows an exemplary CXL card that can hold DDR4 DIMMs. Reusing old DRAM can significantly reduce embodied emissions at the cost of higher operational emissions, due to more power consumed by CXL and the old DIMMs' lower density compared to new DIMMs.

We use off-the-shelf CXL controllers that support DDR4 (e.g., SMC [32], MXC [12]). We decommission a rack of Azure servers that was deployed in 2018. These servers have two sockets, each with six low-capacity and six high-capacity DDR4 DIMMs. We reuse the high-capacity DDR4 DIMMs in our prototype, attaching four DIMMs to each CXL controller.

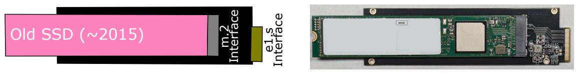


Fig. 4. Old m.2 SSDs from 2015 can be attached in new servers with a passive adapter card in the modern e1.s format.

Finally, we consider reused SSDs, as it is also practical to reuse old SSDs in new servers. Similar to DRAM, the SSD interface standard has moved from m.2 PCIe3 to E1.S PCIe5 drives. Fortunately, PCIe is backward compatible.

SSDs have enough lifetime left for reuse. Typically, modern SSDs fail due to exhausting flash erasure cycles [91], [101]. After seven years, most SSDs offer more than half of the guaranteed erasure cycles. Reusing SSDs can reduce embodied emissions at the cost of higher operational emissions.

We use m.2 SSD drives from decommissioned Azure servers. They are attached via an off-the-shelf passive PCB-adapter (e.g., 2008M2 [24]), as shown in Fig. 4. Further, we 3D print cases, so that the PCB fits into existing E1.s rails and cages.

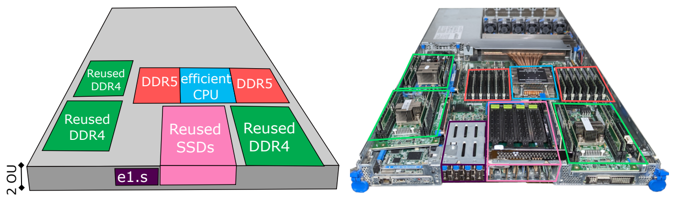


Fig. 5. Our *GreenSKU-Full* design with AMD's efficient CPU, reused DDR4 DRAM (via CXL), and reused m.2 SSDs (via e1.s and PCIe adapters).

Prototype SKUs. In Fig. 5, we show a logical diagram of our *GreenSKU* and an image of our *GreenSKU* prototype. We build three *GreenSKUs* by incrementally adding each component.

1) *GreenSKU-Efficient*: a *GreenSKU* with AMD's efficient Bergamo CPU

2) *GreenSKU-CXL*: *GreenSKU-Efficient* with reused DDR4 memory attached through CXL
3) *GreenSKU-Full*: *GreenSKU-CXL* with reused SSDs

The *GreenSKU-Full* design reuses 8 DDR4 DIMMs attached via two CXL cards (2DPC). We also reused 12 old m.2 SSDs in addition to two new E1.s drives. In aggregate, our components use all 128 PCIe lanes on the AMD Bergamo server.

Performance characteristics. Low-carbon components may have lower performance than baseline SKUs' components. We briefly describe these performance properties, which motivates key components in *GSF*.

AMD Bergamo has a lower frequency and less LLC capacity compared to three AMD generations deployed at Azure (see Table I). Thus, Bergamo incurs 10% and 6% per-core slowdown in Sysbench [77], relative to Genoa and Milan, respectively.

Reusing DDR4 memory via CXL incurs higher latency [81] of about 280ns at medium load, compared to 140ns for local DDR5 accesses. While CXL adds memory bandwidth on top of DDR5, bandwidth per core may be lower than in baseline SKUs. For example, 32 CXL/PCIe5 lanes offer about 100 GB/s using CXL's 256-byte interleaving [103]. AMD Genoa, with 80 cores and 460 GB/s, offers 5.8 GB/s per core. AMD Bergamo, with 128 cores and 460 + 100 GB/s, offers 4.4 GB/s per core.

Our internal analyses show some deployed applications for which memory bandwidth usage is growing, as well as many important applications that exhibit low memory bandwidth usage. We expect low-bandwidth applications to continuously provide opportunities for memory reuse via CXL.

To reduce CXL-induced slowdowns, we use Pond's approach [81]. We use hardware counters to identify which applications can run entirely using CXL memory without facing a slowdown. For other applications, we provision memory across DDR5 and DDR4 and use Pond's prediction model [81] to identify untouched memory regions that can be located on DDR4. On average, untouched memory is almost half of a VM's memory capacity [81]. When this untouched memory is exposed as a virtual compute-less NUMA node, the VM leaves it untouched and does not incur a slowdown [81]. This approach ensures that 98% of applications incur $<5\%$ slowdown with CXL, compared to running entirely with DDR5. In future CPUs, hardware tiering can further improve CXL performance [136].

Reused SSDs also provide lower bandwidth and lower random IO per second. In our measurements for random write speeds, old SSDs offer 1GB/s and 250 IOPS, whereas new SSDs offer 2.3 GB/s and 600 IOPS. We mitigate lower SSD performance using multiple striped RAID sets that each offer more bandwidth and IOPS than the FSP configurations. Due to this mitigation, old SSDs have no adoption side effects.

Although we build our *GreenSKUs* using low-carbon components, they may not be deployable due to the challenges outlined in §II. Thus, we need a systematic way of accounting for these challenges, to evaluate *GreenSKUs*' benefits at scale.

IV. GSF: The GreenSKU Framework

To evaluate a *GreenSKU*, *GSF* estimates a data center's emissions from deploying a *GreenSKU* at scale. Thus, *GSF* enables a cloud provider to evaluate different *GreenSKU* designs. *GSF* systematically considers seven major factors that influence a *GreenSKU*'s carbon savings at scale. *GSF* considers each factor using distinct *components* in its framework, as shown in Fig. 6. We believe *GSF* can be flexibly used by other cloud providers, as it abstracts components' relationships from cloud-provider-specific component implementation details.

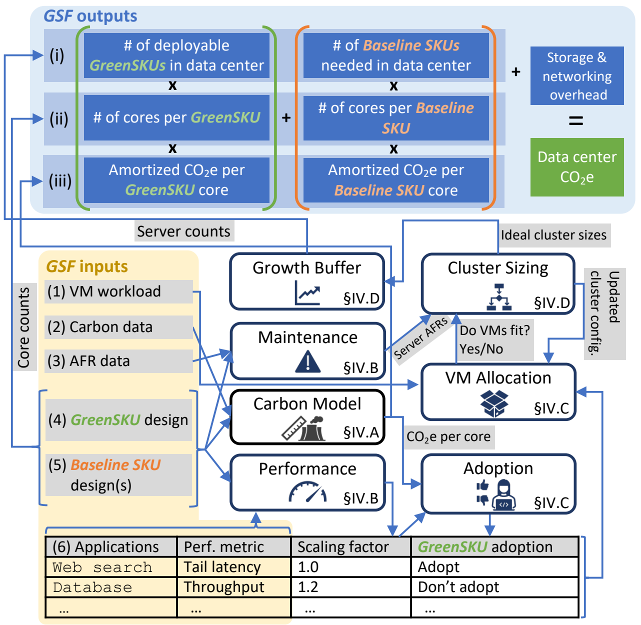
Assumptions. *GSF* estimates operational and embodied emissions and excludes negligible direct emissions (§II). It assumes that cloud users make SKU adoption decisions based on application performance. However, *GSF* can also accommodate other decision factors with minimal changes. *GSF* considers key first-order effects of *GreenSKU* design and assumes that other effects stay constant. For example, it assumes that the total workload (e.g., application's load), networking emissions, and storage emissions remain the same. While we have limited experience with using *GSF* within automatic design space exploration tools, we recommend humans in the SKU design process.

High-level overview. *GSF*'s initial inputs are highlighted in yellow in Fig. 6. They are: (1) a target data center workload, represented as a record of VM deployments over a time period, (2) data used to calculate data center emissions, which includes carbon data (e.g., a component's power and embodied emissions) and data center parameters (e.g., a component's lifetime), (3) component annual failure rates (AFRs), (4) a *GreenSKU* design, (5) a set of currently-deployed baseline SKU designs, and (6) a set of representative applications that report their performance and can run on the *GreenSKU*. We believe this framework specification is generic and can apply to evaluate carbon optimizations beyond the ones we explore.

GSF's final output is the data center emissions from deploying a *GreenSKU*. To estimate this final output, *GSF* calculates the following intermediate outputs for both the *GreenSKU* and baseline SKUs (shown in blue boxes in Fig. 6): (1) the number of servers that must be deployed to serve a given data center workload, (2) the cores per server, and (3) the carbon-per-core emitted over the server's lifetime. Multiplying these intermediate outputs per SKU and adding all SKUs' results yields compute clusters' carbon emissions. Then, adding in other carbon sources (e.g., storage servers' emissions) estimates the overall data center emissions. Note that while we use cores as the main server resource unit, other units can be substituted.

GSF formalizes relationships between components with explicit definitions of inputs and outputs that connect each component. These definitions account for dependencies between components while allowing a cloud provider to implement a component based on their unique cloud constraints. In §V, we describe how we implement each *GSF* component at Azure.

We organize *GSF*'s components into three levels, i.e., the server-, rack-, and data center-level, based on which physical level of the data center the component models. We first detail *GSF*'s carbon model (§IV-A) which spans all levels. We then discuss each component in each level (§IV-B - §IV-D) by specifying (1) which factor the component considers, (2) the component's inputs, and (3) the component's outputs.



(6) Applications	Perf. metric	Scaling factor	*GreenSKU* adoption
`Web search`	Tail latency	1.0	Adopt
`Database`	Throughput	1.2	Don't adopt
...	...	...	...

Fig. 6. Overview of the *GreenSKU* Framework (*GSF*). Components are solid-line boxes. *GSF*'s inputs are highlighted in yellow. Inputs and outputs between components are in gray boxes. Each component contributes to producing the necessary outputs, as shown in blue, to calculate data center emissions.

A. Carbon Model Component

GSF's carbon model component must calculate a given SKU's emissions as a carbon dioxide equivalent (CO_2e), which is a common metric to measure global warming potential [94]. To calculate carbon, the model must estimate embodied emissions and power consumption at the server, rack, and data center level. The model's inputs include server lifetime and the carbon intensity of the energy consumed (CO_2e/MWh). By accounting for the operational emissions over the server's lifetime, the model can aggregate operational and embodied emissions. Finally, the model must account for power consumed by onsite non-IT equipment, e.g., cooling and power distribution devices, to determine the Power Usage Effectiveness (PUE) factor.

The model must also amortize emissions across levels in the data center. For example, a rack's emissions can be divided across the rack's servers to amortize it. Thus, the model must consider constraints on the number of servers-per-rack and racks-per-data center, which depend on space and power. Moreover, as components can have different lifetimes, each component's embodied emissions must be normalized.

To model emissions at the application level, the carbon model must output emissions amortized at a hardware resource granularity that allows attributing emissions to VMs. For example, we chose to use CO_2e-per-core as a common metric and output of this model.

B. Server-Level Components

We identify server performance and maintenance as the main considerations that directly influence a *GreenSKU*'s emissions.

Performance. Since an application running on a *GreenSKU* may face a lower performance-per-core, it may scale up/out to suitably serve the target workload. *GSF*'s performance component must quantify such performance effects.

As shown in the bottom table in Fig. 6, the performance component profiles a *GreenSKU*'s relative performance. To this end, it takes as input (1) the *GreenSKU* and baseline SKU designs to compare against and (2) representative applications that each report a metric to define the application's performance (e.g., tail latency, peak throughput, low-load latency).

This component outputs a *scaling factor* for each application. This factor defines how many *GreenSKU* cores per baseline SKU core are needed for a VM to achieve the application's performance goals. To maintain accurate application characterization over time, *GSF* can work in tandem with existing capacity planning approaches, such as Flux [53], which maintains an active view of the major services in a data center.

Maintenance. When server failures occur, a fraction of servers are out of service, waiting to be repaired [89]. These failures result in the need for additional servers, i.e., an out-of-service overhead. Since *GreenSKUs* can influence the rate of server failures, this overhead must be calculated (e.g., using server components' AFRs). The maintenance component outputs out-of-service overheads for all SKUs.

C. Cluster-Level Components

A *GreenSKU*'s adoption and VM allocation can impact the entire compute cluster's design and efficiency.

Adoption. *GSF*'s adoption component helps decide which applications in the target workload can run on a *GreenSKU* while meeting deployment goals (e.g., performance goals).

As performance is often a first-order goal for applications, this component's inputs are (1) each application's scaling factor as outputted by the performance component and (2) the CO_2e-per-resource value for the baseline SKUs and *GreenSKU* from the carbon model. The carbon information enables the adoption component to balance a *GreenSKU*'s carbon savings against the carbon cost of additional server resources required to scale on the *GreenSKU*. If applications have other deployment constraints (e.g., an application requiring a full baseline server to run on), they must be annotated with such constraints.

The adoption component outputs whether each application should adopt the *GreenSKU*, i.e., whether the *GreenSKU* meets applications' goals and reduces carbon emissions.

VM allocation and packing. Server design impacts how well VMs can be packed into a *cluster*: a logical unit of hundreds of servers to which a VM deployment is routed. Packing density, which is the ratio of allocated to allocatable resources, such as CPU cores and memory, on non-empty servers [66], directly affects the required number of servers. Packing density is influenced by a *GreenSKU*'s resource capacities and the number of servers in a cluster. The VM deployment also defines the workload demand over time that the cluster must serve.

This component takes as input (1) a VM workload, which defines a trace of server resource requests, (2) the adoption and scaling factor for each application to account for which VMs can run and then must scale on *GreenSKUs*, and (3) the cluster configuration, i.e., the number of *GreenSKUs* and baseline SKUs, that is being used to support the VM workload.

This component leverages packing efficiency information to output whether the given cluster configuration can support the workload without rejecting any VM's resource requests.

D. Data Center-Level Components

We identify two factors, cluster sizing and growth buffers, that influence the total number of servers required across all compute clusters in a data center.

Cluster sizing. The cluster sizing component's goal is to determine how many baseline SKUs and *GreenSKUs* are required to service a data center's VM workload. This component must determine the maximum portion of the cluster that can run *GreenSKUs* while still servicing the demand from applications that cannot adopt the *GreenSKU*.

To this end, this component takes as input (1) the VM allocation component's output, i.e., whether a given cluster of *GreenSKUs* and baseline SKUs can support a workload and (2) the maintenance component's output, i.e., the servers' out-of-service overhead, which influences how clusters are sized to manage out-of-service *GreenSKUs* and baseline SKUs.

This component tunes the cluster size using the VM allocation component to check if a cluster can host the workload trace, via simulation. The final output is the number of *GreenSKUs* and baseline SKUs in a right-sized cluster configuration.

Growth buffer. Following standard inventory management practices, a cloud provider maintains a *growth buffer*, i.e., extra server capacity to absorb spikes in VM deployment growth rates, thus mitigating delays in acquiring and deploying additional servers. This buffer is sized to trade off the cost of deploying unused capacity with the risk and subsequent opportunity cost of not having enough capacity.

To determine the growth buffer size, this component's input is the cluster sizing component's final output, which provides a cluster configuration that is properly sized for a certain VM workload demand while not considering future growth.

This component's output is the total number of *GreenSKUs* and baseline SKUs required in a cluster deployment to both service the current demand and also handle VM deployment growth using the growth buffer.

V. *GSF* Implementation for Azure

To evaluate the carbon savings of the three *GreenSKUs* we built (see §III), we implement each *GSF* component under Azure's production constraints.

Implementing GSF's carbon model component. Our carbon model implementation aggregates embodied and operational emissions from server, rack, and data center components. We build on prior carbon models [64], [115] to model server emissions in the cloud. Similar to the ACT model [64], we calculate server-level emissions by aggregating server components' embodied and operational emissions.

First, to model operational emissions, we calculate average server power (P_{s}), which includes the power consumed by each server component, including CPUs, DIMMs, SSDs, NICs, CXL controllers, fans, and secure control management boards. Typically, a server's average power consumption is lower than the sum of its components' TDP [79]. Thus, we scale components' TDP using a derating factor, d. We model inefficiencies in power electronics (e.g., voltage regulators) using a loss factor, l. Thus,

$$P_{\text{s}} = \Big(\sum_{\text{comp. } i} TDP_i * d_i \Big)(1 + l) \tag{1}$$

To model rack-level power (P_{r}), we estimate each rack-level component's power and the number of servers per rack (N_{s}):

$$P_{\text{r}} = N_{\text{s}} * P_{\text{s}} + \sum_{\text{rack comp. } j} P_j \tag{2}$$

To calculate N_{s}, if $P_{\text{r,cap}}$ is the rack's power capacity and $N_{\text{s,cap}}$ is the number of servers that can fit in the rack: $N_{\text{s}} = \min(\lfloor P_{\text{r,cap}}/P_{\text{s}} \rfloor, N_{\text{s,cap}})$.

Then, to model the data center's power (P_{DC}), we use the number of racks (N_{r}), the power dedicated to networking and storage (X), and PUE: $P_{\text{DC}} = (N_{\text{r}} * P_{\text{r}} + X) * PUE$. We calculate N_{r} similarly to how we calculate N_{s}, except using data center space/power limitations for compute racks. Since a data center's operational emissions depend on server lifetime (L) and the energy source's carbon intensity (CI), its operational emissions are: $E_{\text{op,DC}} = P_{\text{DC}} * L * CI$, which includes server- ($E_{\text{op,s}}$) and rack-level ($E_{\text{op,r}}$) operational emissions.

Next, we model embodied emissions by aggregating a data center components' embodied emissions. To model a server's embodied emissions ($E_{\text{emb,s}}$), we use carbon data derived using a model similar to those in the literature [64]. To model the carbon embodied in silicon chips, circuit boards, auxiliary electronics, server chassis, and power supplies, we use component tear downs, CO_2e/cm^2 data, and CO_2e/kg values from public and private data sources [21], [25]. Similar to prior work [115], we consider reused server components to be in their "second life," with zero embodied emissions.

To model rack-level embodied emissions ($E_{\text{emb,r}}$), we add embodied emissions from servers, rack structures, and other rack-level hardware (e.g., power bus, rack controller):

$$E_{\text{emb,r}} = N_{\text{s}} * E_{\text{emb,s}} + \sum_{\text{rack comp. } j} CO_2e_j \tag{3}$$

To model data center-level embodied emissions ($E_{\text{emb,DC}}$), we add the embodied emissions from compute racks, networking/storage (Y), and the non-IT equipment/building (Z): $E_{\text{emb,DC}} = N_{\text{r}} * E_{\text{emb,r}} + Y + Z$.

Finally, we determine the final CO_2e-per-core value by first calculating the number of cores in the data center: $N_{\text{c,DC}} = N_{\text{c,s}} * N_{\text{s}} * N_{\text{r}}$, where $N_{\text{c,s}}$ is the number of cores per server. Then, the CO_2e-per-core is: $(E_{\text{op,DC}} + E_{\text{emb,DC}})/N_{\text{c,DC}}$.

We now show an example of how we calculate emissions at the server- and rack-level using our above model. For brevity, we omit data center-level carbon calculations, which are similar to rack-level carbon calculations. In our carbon model, we use proprietary carbon values. As we are unable to open-source this data, in this example, we use values from public datasets of components' carbon, as well as best-effort estimates when data is unavailable. From these datasets, we source embodied emissions and power data for key *GreenSKU-CXL* components—CPU, DRAM, SSD, and CXL controller. We provide this data in Table V in Appendix A. We describe key carbon calculations for brevity[2], round intermediate calculations' outputs, and show how to calculate amortized emissions across CPU cores.

To calculate *GreenSKU-CXL*'s embodied emissions, $E_{\text{emb,s}}$, we multiply each server component's capacity by its embodied emissions-per-capacity. *GreenSKU-CXL* has an AMD Bergamo CPU, 768GB of DDR5 DRAM, 256GB of reused DDR4 DRAM, 20TB of SSD, and a CXL controller. Thus, for example, to calculate the DDR5 DRAM's embodied emissions, we multiply its capacity (768GB) with its embodied emissions-per-capacity (1.65 kgCO$_2$e). We then add these components' embodied emissions to calculate a total $E_{\text{emb,s}}$ of 1644 kgCO$_2$e.

To estimate *GreenSKU-CXL*'s operational emissions, we use Eq. 1 to calculate server-level power as the product of each component's capacity, TDP-per-capacity, and derating factor, while considering losses from power electronics. As an example of power loss, we model that the CPU faces a 5% power overhead from its voltage regulator losses. We derive the derating factor as a fraction of TDP utilization at a given percentage of max SPEC rate [122]; at 40% SPEC rate, the corresponding derating factor is 0.44. Applying this derating factor to every component, Eq. 1 results in $P_{\text{s}} = 403\text{W}$.

Next, we calculate rack-level emissions. Since there is a lack of public data on an empty rack's emissions, we use the estimates for a rack's TDP and embodied emissions in Table V. To calculate rack-level emissions, we consider the number of servers in a rack; a rack with 32U of space available for servers can fit 16 *GreenSKU-CXL* servers with a form factor of 2U. We calculate rack power constraints by subtracting the rack's power (500W) from the rack's power capacity limit (15,000W) and dividing by the server-level power, P_{s}, i.e., $\lfloor(15,000 - 500)/403\rfloor = 35$. As this value is greater than 16, the rack is space-constrained to $N_{\text{s}} = 16$ servers.

To calculate rack-level embodied emissions, we use Eq. 3 to multiply the number of *GreenSKU-CXL*s and the server-level embodied emissions. We then add an empty rack's embodied emissions, i.e., $E_{\text{emb,r}} = 16 * 1644 + 500 = 26,804$ kgCO$_2$e.

To calculate rack-level operational emissions, we apply Eq. 2 to calculate rack-level power, P_{r}. An empty rack's power consumption is 500W. Thus, $P_{\text{r}} = 16 * 403 + 500 = 6953\text{W}$. The lifetime of our servers, L, is 6 years ($52,560$ hours). We calculate that $CI = 0.1$ kgCO$_2$e/kWh by averaging the estimated carbon intensity across Azure's large data center regions. Thus, $E_{\text{op,r}} = L * CI * P_{\text{r}} = 36,547$ kgCO$_2$e.

The net rack-level emissions, E_{r}, is the sum of a rack's operational and embodied emissions, i.e., $E_{\text{r}} = E_{\text{op,r}} + E_{\text{emb,r}} = 26,804 + 36,547$ kgCO$_2$e $= 63,351$ kgCO$_2$e.

Finally, we calculate the rack-level CO$_2$e-per-core by dividing E_{r} by number of cores in a rack, $N_{\text{c,r}}$. To calculate $N_{\text{c,r}}$, we multiply the number of servers in a rack by the number of cores per server, i.e., $N_{\text{c,r}} = 16 * 128 = 2048$. *GreenSKU-CXL*'s rack-level CO$_2$e-per-core is then $E_{\text{r}}/N_{\text{c,r}} = 63,351/2,048$ kgCO$_2$e $= 31$ kgCO$_2$e.

Implementing GSF's performance component. Prior work identifies that six application classes run in the majority of VMs in Azure [95]. They include: (1) big data (e.g., in-memory data stores), (2) web applications (e.g., information retrieval), (3) real-time communication or RTC (e.g., speech recognition), (4) Machine Learning (ML) inference (e.g., image recognition), (5) web proxy (e.g., front-end web server), and (6) DevOps (e.g., code compilation). The reported share of production core-hours for each application class [95] is shown in Table III.

Across these six classes, we benchmark 20 open-source and closed-source applications' performance. For big data, we study `Redis` [43]: an in-memory key–value store, `Masstree` [74], [90]: a key-value database, `Silo` [74], [108]: an Online Transaction Processing (OLTP) database, and `Shore` [68], [74]: an OLTP database. For web applications, we study `Xapian` [33], [74] and four Microsoft production services—`WebF-Dynamic`, `WebF-Hot`, `WebF-Cold`, and `WebF-Mix`. For RTC, we study Moses: a speech translation service [74], [130] and Sphinx [74], [80]: a speech recognition service. For ML inference, we study `Img-DNN` [74]—an image recognition service. For web proxy, we study front-end web servers like `Nginx` [109], `Caddy` [58], `Envoy` [16], `HAProxy` [19], and `Traefik` [104]. For DevOps, we evaluate `Build-Python`, `Build-Wasmer`, and `Build-PHP`.

We measure a *GreenSKU*'s performance by setting a Service Level Objective (SLO) based on a baseline SKU's performance. Our baseline SKUs are three deployed server generations, Gen 1, 2, 3 (see Table I). Successive baseline SKUs use newer hardware and have better performance. To achieve comparable performance as the high-performance baseline SKU, we scale up the number of VM cores on the *GreenSKU* to 8, 10, and 12 cores and compare the resulting performance against an 8-core VM running on the baseline SKU. Using these results, we calculate the scaling factors relative to each baseline SKU.

Implementing GSF's maintenance component. We use Little's law [107] to estimate that the fraction of out-of-service servers is the product of average repair time and server AFR. From observations at Azure, we find that our *GreenSKUs*' design choices do not significantly affect repair time. Our *GreenSKUs*' components like reused DIMMs and SSDs are easily accessible, and diagnosing them is a well-established process due to their previous deployment at Azure.

To estimate our *GreenSKU-Full*'s AFR, we must consider failures from reusing older DIMMs and SSDs. These components typically increase server AFR, as they often fail even in baseline SKUs[3]. We approximate the average failure rate by adding these components' AFRs[4]. For example, a baseline SKU with 12 DIMMs and 6 SSDs has an AFR of 4.8. Our *GreenSKU*-

[2]The complete set of calculations using our open-source carbon model [18] is in Appendix A.

[3]DIMMs and SSDs constitute half of a server's AFR [89], with AFRs of approximately 0.1 and 0.2, respectively.

[4]Concurrent failures largely occur due to class failures, e.g., recalls for production days of a component, and occur rarely for old reused components.

Full has 20 DIMMs and 14 SSDs (Table IV), causing an AFR of 7.2. We use the same AFRs for new and reused components, as we empirically observe that reused DIMMs and SSDs have lower or equal AFRs than new components (§II).

To reduce repair rates, we use Fail-In-Place (FIP) [89] at Azure. FIP is highly effective for *GreenSKU-Full*, due to its large number of DIMMs and SSDs. Using a conservative FIP effectiveness rate of 75% for DRAM and SSD [89], the repair rate per 100 servers for the baseline SKU and *GreenSKU-Full* reduces to 3 and 3.6 (from AFRs of 4.8 and 7.2), respectively.

We now estimate the higher maintenance emissions, C_{OOS}, due to *GreenSKU-Full*'s higher repair rate. C_{OOS} is the product of a SKU's per-server repair rate, the number of servers (N_s) needed to run our applications, and the per-server emissions (E_s). On average, we need 0.66 *GreenSKU-Fulls* per baseline SKU, when we factor in a *GreenSKU-Full*'s increased resources while scaling VM cores to match baseline SKU performance. However, we must also account for our *GreenSKU-Full*'s per-server carbon being 26.2% higher than the Gen3 baseline SKU, as *GreenSKU-Full* has more resources. On multiplying the calculated repair rate, N_s, and E_s normalized to the baseline SKU's E_s, we get: $C_{OOS} = 3 \times 1 \times 1 = 3$ for the baseline SKU and $C_{OOS} = 3.6 \times 0.66 \times 1.262 = 2.98$ for *GreenSKU-Full*. Thus, *GreenSKU-Full*'s maintenance overheads are negligible.

Implementing GSF's adoption component. Our adoption component assumes that cloud users aim to reduce their applications' emissions while meeting performance goals. Thus, to decide whether an application can adopt a *GreenSKU*, we calculate the carbon required to service the application on a *GreenSKU*. To this end, we multiply the number of *GreenSKU* cores needed to achieve the baseline SKU's performance (determined from *GSF*'s performance component) by the CO_2e-per-core determined from the carbon model. We also calculate this value for the baseline SKU, using 8 cores and the baseline SKU's CO_2e-per-core. We model that an application will adopt a *GreenSKU* if the calculated carbon value to run the application on the *GreenSKU* is lower than the baseline's, i.e., running on the *GreenSKU* saves carbon while meeting performance goals. We repeat this step for each representative application.

Implementing GSF's VM allocation component. To evaluate how effectively varied-sized VMs can be packed within *GreenSKU* servers, we use a VM allocation simulator that captures Azure's production scheduler's key VM placement rules. These rules include (1) using best-fit placement heuristics that reduce resource fragmentation, (2) preferring to place VMs on non-empty nodes, and (3) enforcing VM placement constraints. Our simulations use real VM arrival/departure traces and VM configurations from multiple Azure data centers.

Since the applications running in VMs are opaque in production traces, we assign each VM in our trace to one of our representative benchmark applications. We determine the application class by sampling from the core hour percentages in Table III. We then uniformly sample from that application class to assign an application to the VM.

The VM's server generation (i.e., Gen 1, 2, or 3) is pre-defined in our traces. We determine whether a VM can run its application on a *GreenSKU* instead, i.e., adopt it, using our adoption model and the VM's application assignment. If a VM can adopt the *GreenSKU*, we multiply the VM's core count and memory allocation size by the scaling factor required to run its application on the *GreenSKU*. We use the scaling factor that corresponds to the VM's pre-defined server generation.

Apart from VMs that do not adopt the *GreenSKU*, we have long-living "full-node VMs" that require a dedicated server. We strictly assign these VMs to baseline SKUs, as they have fewer resources, i.e., dedicating a *GreenSKU*'s increased cores and memory to such a VM would cause wasted resources.

Implementing GSF's cluster sizing component. We use the VM arrival/departure trace and each VM's *GreenSKU* adoption decision to determine how many baseline SKUs and *GreenSKUs* are required to serve the cluster's VM workload. We find the number of such servers using our VM allocation simulator. To this end, we first right-size a baseline SKU-only cluster by increasing the number of simulated servers until no VM is rejected, i.e., identify the minimum number of servers in a baseline SKU-only cluster that can host all VMs. Next, we incrementally replace each baseline SKU with enough *GreenSKU* servers until no VM is rejected. We repeat this process until we can no longer replace baseline SKUs, to identify the right number of baseline SKUs required to run the VMs that cannot adopt the *GreenSKU*. This search identifies the cluster size of *GreenSKUs* and baseline SKUs that minimizes emissions while supporting our VM workload.

Implementing GSF's growth buffer component. Typically, the growth buffer size is calculated using models that require historical workload trends [49]. As we do not have such trends for a new *GreenSKU*, we use a workaround that enables a VM to fungibly run on a *GreenSKU* while there is enough *GreenSKU* capacity available. If there is no such capacity, the VM may also run on a baseline SKU. This approach maintains the growth buffer using only baseline SKUs, whose historical workload trends are available. It also overcomes the need for multiple buffers, simplifying *GreenSKU* deployment. However, this approach marginally increases emissions, as the entire buffer has carbon-inefficient baseline SKUs. We consider these emissions in our savings estimate.

VI. Evaluating our GreenSKUs Using GSF

We use our *GSF* implementation at Azure to evaluate our *GreenSKUs* from §III. We compare each of our *GreenSKUs*, i.e., *GreenSKU-Efficient*, *GreenSKU-CXL*, and *GreenSKU-Full*, against baseline SKUs from three generations. Since we noted the quantitative carbon contributions of some *GSF* components (e.g., maintenance) earlier, in §V, we omit those components.

Evaluating GreenSKUs' performance and adoption. To evaluate our *GreenSKUs*' performance, we measure the $95^{th}\%$ tail latency across different Queries Per Second (QPS) loads for one representative applications in each application class. Similar to prior work [46], [85], [137], we set applications' SLO as the $95^{th}\%$ (tail) latency achieved at 90% of the compared baseline SKU's peak saturation throughput. Across all experiments with these applications, we conduct three trials and report our data

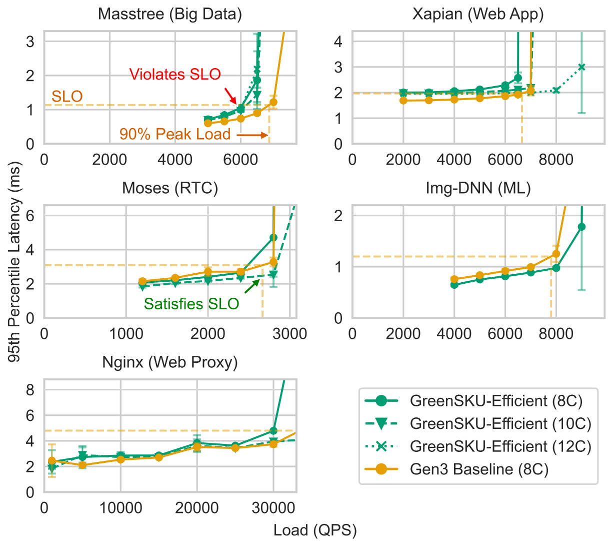


Fig. 7. 95th% tail latency vs. load (QPS) for applications spanning five of our six application categories. Tail latency is shown for 8-core configurations for the Gen3 baseline SKU (in orange). Results for *GreenSKU-Efficient* are shown up to the scaling required to achieve comparable performance, if possible, to Gen3. The dotted orange line indicates an SLO set using Gen3's latency at 90% of peak load. For some applications (e.g., `Xapian` and `Nginx`), our *GreenSKU-Efficient* can achieve the SLO with scaling; for other applications (e.g., `Masstree`), the scaling required outweighs carbon savings.

DevOps App.	Gen1	Gen2	Gen3	*GreenSKU-Efficient*	*GreenSKU-CXL*
`Build-PHP`	1.27	1.11	1.00	1.17	1.38
`Build-Python`	1.28	1.13	1.00	1.15	1.21
`Build-Wasmer`	1.34	1.19	1.00	1.15	1.28

TABLE II
GREENSKU-EFFICIENT'S NORMALIZED SLOWDOWN COMPARED TO BASELINE SKUS WHEN COMPILING THREE DEVOPS PROGRAMS.

with 99% confidence intervals. We also measure 99th% latency and notice similar behaviors.

We first evaluate *GreenSKU-Efficient*'s performance and adoption. We study 20 applications running on *GreenSKU-Efficient* and compare its performance against the baseline Gen 1, 2, 3 servers. For brevity, in Fig. 7, we show the 95th% (tail) latency across different loads for one representative application in each application class. To compare performance and determine the scaling factors, we scale the number of VM cores on *GreenSKU-Efficient* to 8, 10, and 12 and compare the resulting tail latency against an 8-core VM running on Gen 1, 2, 3 servers. We show results up to the minimum number of cores on *GreenSKU-Efficient* that achieves a peak saturation throughput closest to our Gen3 server. We omit results for Gen1 and Gen2, as they consistently perform worse than Gen3.

We observe that for applications such as `Masstree`, even with 12 cores, *GreenSKU-Efficient* cannot match Gen3's peak throughput, and violates SLOs beyond 6000 QPS. However, for other applications, such as `Xapian`, `Moses`, and `Nginx`, *GreenSKU-Efficient* achieves SLOs with 10–12 cores. Thus, *GreenSKU-Efficient* effectively meets the performance goals of several latency-critical applications, albeit with scaling.

Next, we show our DevOps applications' results in Table II separately, as they only report throughput. We report average

			GreenSKU Performance Scaling Factor		
Application Category	**% of Fleet Core Hours**	**Application**	**Gen1**	**Gen2**	**Gen3**
Big Data	32	`Redis`	1	1	1
		`Masstree`	1	1	>1.5
		`Silo`	>1.5	>1.5	>1.5
		`Shore`	1	1	1
Web App	27	`Xapian`	1	1	1.5
		`WebF-Dynamic` *	1	1.25	1.25
		`WebF-Hot` *	1	1.25	1.5
		`WebF-Cold` *	1	1	1
		`WebF-Mix` *	1	1	1
Real-Time Communication	24	`Moses`	1	1	1.25
		`Sphinx`	1	1.25	1.25
Machine Learning Inference	11	`Img-DNN`	1	1	1
Web Proxy	4	`Nginx`	1	1	1.25
		`Caddy`	1	1	1
		`Envoy`	1	1	1
		`HAProxy`	1	1	1.25
		`Traefik`	1	1	1.25
DevOps	1	`Build-Python`	1	1	1.25
		`Build-Wasmer`	1	1	1.25
		`Build-PHP`	1	1	1.25

TABLE III
GREENSKU-EFFICIENT'S PERFORMANCE SCALING FACTOR COMPARED TO GEN 1, 2, 3 FOR EACH APPLICATION. "*" IS A PRODUCTION APPLICATION.

slowdowns normalized to the Gen3 server, when using 8 cores. *GreenSKU-Efficient* outperforms Gen1 for all applications, while facing only 1.15x-1.17x slowdown compared to Gen3.

We repeat these experiments for all 20 applications, calculating the scaling factors required relative to the three baseline SKUs. We report these scaling factors in Table III.

For seven applications, *GreenSKU-Efficient* meets Gen3's SLO without any scaling. For another nine applications, scaling by 25% is required to achieve Gen3's SLO. Some applications, such as `Silo`, have a significant scaling factor. Our *GSF* adoption component's implementation notes that these applications cannot be run on *GreenSKU-Efficient*, as they offset *GreenSKU-Efficient*'s carbon savings. These results show that *GSF* can help determine scaling factors to identify when an application can be deployed on a *GreenSKU* to achieve carbon savings while meeting performance goals.

As with prior work [46], [119], we define 30% of peak throughput as "low" load to measure *GreenSKU-Efficient*'s impact on low-load latency, which is a key metric for some latency-critical applications. We compare the low-load latency of each application running on *GreenSKU-Efficient* when scaled with the scaling factor relative to our baseline SKUs. Running on *GreenSKU-Efficient* results in a median low-load latency (across applications) that is 8.3% and 2% lower than Gen1 and Gen2, respectively, and 16% higher than Gen3. As applications often express SLOs in terms of tail latency and peak throughput [95], this 16% increase in low-load latency is unlikely to significantly impact the *GreenSKU*'s adoption.

Next, we study *GreenSKU-CXL*'s performance by evaluating the performance impact of reusing older memory via CXL. In Fig. 8, we show the 95th% latency vs. load for two representative applications, `Moses` and `HAProxy`, that, respectively, exhibit a high and low latency penalty with CXL compared to *GreenSKU-Efficient*'s performance.

We find that running on *GreenSKU-CXL* greatly affects

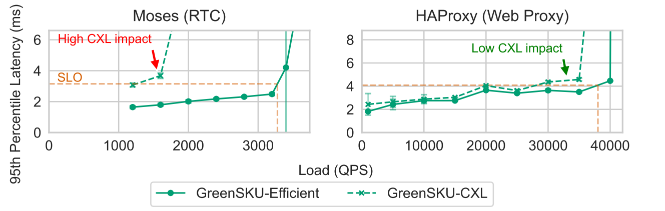


Fig. 8. $95^{th}\%$ tail latency vs. load for an application that is more (`Moses`) and less (`HAProxy`) impacted by reusing memory via CXL. The number of cores is equivalent in both *GreenSKU* configurations and is the number of cores required to achieve that application's SLO relative to Gen3.

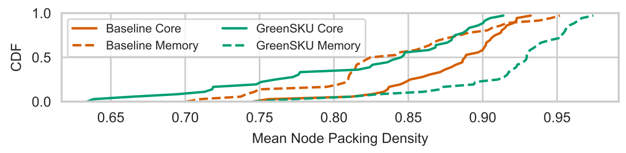


Fig. 9. CDF of the mean packing density for core (solid) and memory (dashed) across servers for each trace. We show the packing density for the all-baseline cluster (in orange) and the *GreenSKU-Fulls* in the final simulated cluster (in green). Our *GreenSKU-Full* design makes a carbon tradeoff, as it shows better memory packing density with worse core packing density.

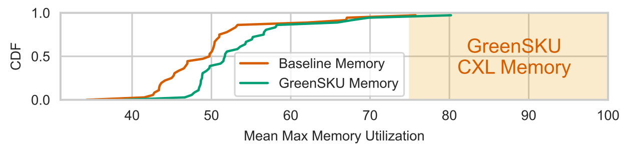


Fig. 10. CDF of the mean per-server maximum memory utilization across all servers for a cluster of baseline SKUs and *GreenSKU-CXL*. The shaded region indicates memory accessed through CXL on the *GreenSKU*. Almost all servers can service their VMs' memory demand with local DDR5 memory.

`Moses`'s performance. `Moses` saturates early, at 1700 QPS, and fails to meet the SLO from 1200 QPS. Since `Moses` uses speech language models that have a large memory footprint, it is significantly memory bound. Thus, its performance is heavily impacted by the increased memory latency with CXL.

In contrast, `HAProxy` meets the SLO across most high load conditions and only faces an 11% reduction in peak throughput compared to *GreenSKU-Efficient*. Since `HAProxy` is a load balancer that is compute- and network-bound, its performance is minimally affected when running on *GreenSKU-CXL*.

We note that 20.2% of our applications, weighted by proportion of fleet core-hours, do not face significant performance penalties when running on *GreenSKU-CXL*. These applications can run in VMs whose entire memory is backed by CXL. Other VMs can benefit from running on a *GreenSKU* that uses a mix of DDR5 and CXL-attached DDR4 memory (§III).

Evaluating GreenSKUs' impact on VM packing. We now evaluate how well we can pack VMs on our *GreenSKUs*. We run VM packing simulations on 35 production VM traces. For each trace, we select the minimum size of the baseline SKU and *GreenSKU* clusters that avoid VM rejection.

Fig. 9 shows the mean VM packing densities' distribution across production traces, comparing right-sized clusters of baseline SKUs and *GreenSKU-Fulls*. We find different VM packing densities across the baseline SKU and *GreenSKU-Full* due to differing core counts, i.e., 80 vs. 128 cores, and memory:core ratios, i.e., 9.6 vs. 8, respectively. The baseline SKU's higher memory:core ratio causes higher core-packing density at the expense of memory wastage. Thus, we show how *GSF*'s VM allocation and cluster sizing components can help study VM packing on *GreenSKUs* and its impact on emissions.

The packing density also helps understand server components' utilization. Using the packing density, we validate if we can back untouched memory with reused DRAM via CXL, thus reducing CXL-induced performance loss. To this end, we note that in our VM traces, each VM reports the maximum amount of its allocated memory that it uses over its lifetime. We replay each trace and periodically take snapshots of the servers, to aggregate the maximum memory usage across all VMs on each server. We then average across servers and snapshots to identify a trace's average maximum memory utilization.

We show a CDF of the mean per-server maximum memory utilization in Fig. 10 for clusters with both baseline-only SKUs and *GreenSKU-CXLs*. The shaded portion is the 25% of memory reused via CXL. In most traces, we see a maximum memory utilization below 60%, which can be accommodated by *GreenSKU-CXL*'s local memory. Only 3% of traces have a memory utilization that would require using CXL.

Moreover, we can leverage our observation that CXL does not cause a performance loss for 20% of our applications, to schedule these applications to use CXL-backed memory. Thus, using the VM allocator, we show that reusing DRAM via CXL does not significantly impact *GreenSKU-CXL*'s adoption.

Evaluating GreenSKUs' carbon savings. We now evaluate our *GreenSKUs*' carbon savings using the average grid carbon intensity across major Azure data center regions. Table IV shows our *GreenSKUs*' per-core operational and embodied emissions savings over the Gen3 baseline SKU. "Baseline-*Resized*" is the baseline SKU when its memory:core ratio is reduced from 9.6 to 8. This ratio is carbon-optimal for our workload traces, based on *GSF*'s estimates. Thus, "Baseline-*Resized*" saves 4% carbon compared to the baseline SKU (3% operational and 6% embodied carbon savings).

GreenSKU-Efficient uses the carbon-optimal memory:core ratio of 8 with its efficient CPU. Due to its efficient CPU's lower power consumption per core, *GreenSKU-Efficient* saves 29% in operational emissions. Furthermore, as *GreenSKU-Efficient* leads to more cores per server and rack, it better amortizes the rack- and data-center-level embodied emissions per core. Thus, *GreenSKU-Efficient* saves 14% embodied emissions per core, resulting in 23% net per-core carbon savings.

GreenSKU-CXL replaces 30% of *GreenSKU-Efficient*'s memory with 32GB DDR4 DIMMs connected via CXL. Since these CXL controllers consume additional power, *GreenSKU-CXL* achieves 6% lower per-core operational emission savings compared to *GreenSKU-Efficient*. However, due to DDR4 memory reuse, *GreenSKU-CXL* saves 11% more per-core embodied emissions compared to *GreenSKU-Efficient*, improving the net per-core savings by 1% compared to *GreenSKU-Efficient*.

Finally, *GreenSKU-Full* replaces 60% of *GreenSKU-CXL*'s

SKU Config.	**# Cores**	**# $\times$ DIMM (GB)**	**# $\times$ SSD (TB)**	**Operational Savings**	**Embodied Savings**	**Total Savings**
Baseline	80	12 $\times$ 64	6 $\times$ 2	-	-	-
Baseline-*Resized*	80	10 $\times$ 64	6 $\times$ 2	3%	6%	4%
GreenSKU-Efficient	128	12 $\times$ 96	5 $\times$ 4	29%	14%	23%
GreenSKU-CXL	128	12 $\times$ 64 8 $\times$ 32 CXL	5 $\times$ 4	23%	25%	24%
GreenSKU-Full	128	12 $\times$ 64 8 $\times$ 32 CXL	2 $\times$ 4 12 $\times$ 1 Reuse	17%	43%	28%

TABLE IV
PER-CORE OPERATIONAL, EMBODIED, AND TOTAL CARBON SAVINGS (CALCULATED BASED ON THE AVERAGE CARBON INTENSITY FOR MAJOR AZURE REGIONS) RELATIVE TO OUR GEN3 BASELINE SKU FOR FOUR INCREMENTAL GREENSKU CONFIGURATIONS.

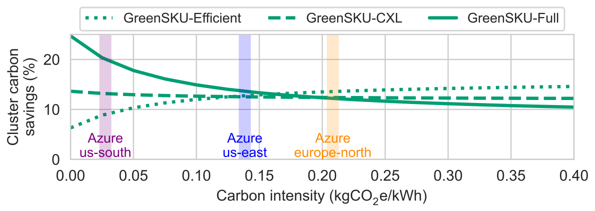


Fig. 11. End-to-end cluster-level carbon savings relative to clusters of all baseline SKUs across a range of carbon intensities evaluated for our three *GreenSKU* configurations. The vertical lines mark estimated carbon intensities for the energy used by three Azure data center regions [26]. The best *GreenSKU* design depends on the data center's operating conditions.

storage with reused SSDs. As reused SSDs are less energy efficient, *GreenSKU-Full* saves 6% lesser per-core operational emissions compared to *GreenSKU-CXL*. However, due to SSD reuse, *GreenSKU-Full* saves 18% more embodied emissions compared to *GreenSKU-CXL*. Thus, *GreenSKU-Full* achieves a 28% per-core carbon savings compared to the baseline SKU.

We now evaluate our *GreenSKUs*' cluster-level carbon savings. We compare the carbon savings achieved by a cluster of *GreenSKUs* and baseline SKUs against a cluster of all baseline SKUs. In Fig. 11, we show the cluster-level carbon savings achieved by *GreenSKU-Efficient*, *GreenSKU-CXL*, and *GreenSKU-Full*. We evaluate across a spectrum of carbon intensity values. We annotate the carbon intensities for the energy used by three Azure data center regions.

The cluster with *GreenSKUs* saves 6%–25% carbon compared to the baseline SKUs' cluster. The lower-carbon *GreenSKU* configuration depends on the carbon intensity of the data center's grid, which affects operational emissions. At higher carbon intensity, as with the `Azure-europe-north` data center region, *GreenSKU-Efficient* saves more carbon as it does not have inefficient reused components. At lower carbon intensity, as with the `Azure-us-south` region, saving embodied emissions matters more. Thus, *GreenSKU-Full* saves more carbon as it saves more embodied emissions from reuse.

VII. WHY DESIGN GREENSKUS?

We detail why reducing emissions by deploying *GreenSKUs* (1) also reduces cost and (2) is more practical compared to other cloud emission reduction approaches. We use the flexibility of our *GSF*'s carbon model to perform such comparisons.

A. *GreenSKU Cost Analysis*

GSF can be adapted to analyze *GreenSKUs*' effect on Total Cost of Ownership (TCO) by replacing the carbon model with a TCO model. Since TCO data is sensitive, we share high-level insights from our TCO analysis.

Our TCO analysis reveals that a cost-efficient server SKU is only 5% less costly compared to our carbon-efficient *GreenSKU*. This relatively small TCO loss may be tolerable to a cloud provider seeking to meet aggressive decarbonization targets. Moreover, cloud providers can use *GSF* to evaluate other SKUs that achieve the required carbon vs. cost tradeoff.

B. *GreenSKU vs. Other Carbon Reduction Strategies*

To reduce emissions, prior work has proposed (1) increasing renewables use [10], (2) improving servers' energy efficiency [93], and (3) increasing server lifetimes [115], [126]. We comment on how designing *GreenSKUs* can be a more practical and complementary approach to reduce emissions.

Increasing renewable generation. Cloud providers reduce emissions by increasing renewable energy use [1], [10]. We calculate that an increase of 2.6% in the percentage of energy coming from renewables for the average Azure data center is required to match our *GreenSKU-Full*'s data center-wide carbon savings. While this increase appears small, there are major challenges in realizing it. First, in many parts of the world, including the United States, grid decarbonization has been slow due to infrastructural and political challenges, with the renewable percentage increasing by 1.2% annually on average in the last five years [15]. Second, it is challenging for a data center to fully utilize an increase in renewables; prior work suggests that increasing renewable energy coverage from 95% to 99.9% requires investing 5$\times$ the cost required to go from 0% to 95%, due to the long tail in generation variance [35].

Improving server energy efficiency. Improving energy efficiency reduces operational emissions [64], [131]. We analyze how much servers' energy efficiency must increase to achieve our *GreenSKU-Full*'s data center-wide carbon savings. We optimistically assume that improvements in energy efficiency (1) do not increase embodied emissions and (2) occur uniformly across server components. We use the current average grid intensity of some of Azure's largest regions for our analysis.

We estimate that all server components must become 28% more energy efficient. Achieving such improvements can take years. For example, upgrading from AMD's Zen 3 to Zen 4 (separated by two years) improves energy-efficiency by 25% [6]. Note that upgrading every cloud server every two years would cause high embodied emissions, decreasing carbon savings.

Increasing server lifetime. To reduce emissions, prior works extend server lifetime via scheduling [126], better maintenance [88], and reuse of discarded devices [115]. We analyze how much server lifetimes must increase by, to achieve our *GreenSKU-Full*'s carbon savings, with a simplifying assumption that extending lifetimes does not increase operational emissions. Using Azure's renewables mix, we estimate the required lifetime extension to be 6 $\rightarrow$ 13 years.

Extending lifetimes to 13 years requires a radical data center stack redesign. For example, maintenance can become cost prohibitive over this time frame [88], [89]. Older servers also tend to have higher per-core operational emissions relative to newer hardware [64], [75]. *GSF* can evaluate server lifetime extension by considering such extension's impact on maintenance, performance, and emissions.

Summary. Thus, strategies like increasing renewable energy use, improving energy efficiency, and increasing lifetimes require significant investment and navigation of deployment challenges, to achieve our *GreenSKUs*' carbon savings. The latter two approaches trade off operational vs. embodied emissions in a way that can hurt net carbon savings in some deployment scenarios. Our evaluation with *GSF* shows that our *GreenSKU* designs navigate this tradeoff to save carbon across diverse carbon intensities, as summarized in Fig. 11.

VIII. Discussion

We briefly discuss open questions and limitations.

Scheduling real-time applications. We show how to evaluate *GreenSKU* deployment. Run-time systems that leverage *GreenSKUs*, post-deployment, are an opportunity for future work. For example, auto-scalers [98], [100] can improve *GreenSKUs*' performance during load changes. Tuning CPU configurations (e.g., frequency) [70], [112] can also help a *GreenSKU* adapt to application changes post-deployment.

Navigating component search space. While our *GreenSKU* achieves significant carbon savings, it may not be the optimal configuration. When designing our *GreenSKUs*, we used parts of *GSF* to iterate through hundreds of configurations. To identify optimal configurations, we must consider components' dynamic interactions in terms of performance (e.g., memory and core frequency) and compatibility (e.g., CXL-compatibility). We expect that a future search framework could consider such interactions and repeatedly run *GSF* to evaluate emissions.

Heterogeneous compute on GreenSKUs. *GSF* focuses on general-purpose compute servers. Extending *GSF* to study *GreenSKUs* with heterogeneous accelerators, e.g., for ML, may require adjustments. For example, the adoption model's "scaling factor" may need to reflect scaling out across CPUs and/or accelerators. Such extensions can help study accelerator-reuse for less compute-intensive ML models and reusing offload engines for less IO-intensive tasks [55], [82], [83].

Assumptions. *GSF* pessimistically assumes that scaling out requires a proportional increase in core count, memory, and disk capacity. While this assumption may be true for some applications, it was rarely true in our experiments. Thus, it may be possible to further reduce DRAM and SSD provisioning.

Limitations. While our carbon model considers key carbon contributors, it may not cover all factors. The underlying carbon data is also changing as emissions-tracking processes mature.

IX. Related Work

To the best of our knowledge, *GSF* is the first framework to help cloud providers systematically evaluate server designs' emissions at data center scale. We now discuss related work.

Designing server SKUs. Prior work on server SKU design [44], [69] primarily improves performance and cost, rather than carbon. SoftSKU [112] considers the performance impact of running services on efficient cores, but does not consider carbon. Other works [56], [67], [93], [114], [133] redesign servers to improve energy efficiency. In contrast, we show how to systematically evaluate a carbon-efficient server SKU's potential to reduce emissions.

Reducing carbon emissions. Many works build systems to improve cloud resource utilization [46], [57], [59], [60], [72], [86], [87], [92], [97], [113], [135], energy efficiency [45], [51], [52], [62], [105], and power management [71], [79], [96], [118], [132], [134]. While these systems might indirectly lower carbon emissions, they do not explicitly consider the tradeoffs between operational and embodied emissions. Moreover, these works can augment *GreenSKUs* to improve resource utilization.

Prior work reduces operational emissions by shifting a data center's workload spatially (i.e., across data centers) and temporally (i.e., batching workloads during certain periods) to leverage renewables' availability [20], [35], [40], [99], [110], [129]. These solutions can apply on top of *GreenSKUs*.

Switzer et al., [115] run some services on discarded smartphones, as they find that it offers more carbon savings compared to old laptops and servers. Prior works [89], [117], [124]–[126] also extend server lifetimes to reduce embodied emissions, which adds a dimension to the *GreenSKU* design space. Gupta et al. [64] motivate saving carbon from reducing and reusing hardware. *GSF* enables evaluating such carbon-efficient server optimizations' benefits at data center scale.

Planning cloud capacity. Prior works on capacity planning determine the resource capacity required to support a data center's workload [42], [61], [84], [120], [127]. We build on these approaches by introducing emissions as a key metric to consider in capacity planning decisions. Flux [53] distributes service capacity across geo-distributed servers. Such capacity management solutions can help intelligently distribute services across data centers with different *GreenSKU* capacities.

X. Conclusion

To reduce cloud emissions, designing carbon-efficient *GreenSKUs* is a promising solution. Thus, we designed three *GreenSKUs* with low-carbon server components. However, it is challenging to determine our *GreenSKUs*' carbon savings at scale. To this end, we developed a novel framework, *GSF*, to help cloud providers to systematically evaluate a *GreenSKU*'s benefits. We applied *GSF* within Azure's production constraints to evaluate our *GreenSKUs*. We found that our *GreenSKUs* reduce emissions by 28% compared to the currently-deployed cloud servers. They also reduce Azure's net emissions by 8%.

Acknowledgement

We thank the anonymous reviewers for their insightful feedback. This work was partially supported by an NSF CAREER Award CCF-2340042. Jaylen Wang was supported by an NSF Graduate Research Fellowship, a Benjamin Garver Lamme/Westinghouse Graduate Fellowship, and a Jack and Mildred Bowers Scholarship in Engineering.

References

[1] "2022 Environmental Sustainability Report," https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RW14sJN, (Accessed on 04/26/2024).

[2] "4th Gen Intel Xeon Scalable Processors," https://download.intel.com/newsroom/2023/data-center-hpc/4th-Gen-Xeon-Scalable-Product-Brief.pdf, (Accessed on 04/26/2024).

[3] "Amazon EC2 M7a Instances," https://aws.amazon.com/ec2/instance-types/m7a/, (Accessed on 04/26/2024).

[4] "AMD adds 128-core Bergamo and 3D V-Cache Genoa CPUs to Zen 4 Epyc lineup," https://www.xda-developers.com/amd-128-core-bergame-genoa-epyc-cpu/, (Accessed on 04/26/2024).

[5] "AMD EPYC 9754 Benchmarks For The 128-Core Bergamo Review," https://www.phoronix.com/review/amd-epyc-9754-bergamo, (Accessed on 04/26/2024).

[6] "AMD's Financial Analyst Day 2023 Press Release," https://ir.amd.com/news-events/press-releases/detail/1078/amd-details-strategy-to-drive-next-phase-of-growth-across, (Accessed on 04/26/2024).

[7] "AmpereOne 192-core CPU Family Product Brief," https://amperecomputing.com/briefs/ampereone-family-product-brief, (Accessed on 04/26/2024).

[8] "AWS is running a 96-core, 192-thread, custom Xeon server," https://www.theregister.com/2023/08/03/aws_custom_xeon_m7i_instances/, (Accessed on 04/26/2024).

[9] "Azure Compute Units," https://learn.microsoft.com/en-us/azure/virtual-machines/acu, (Accessed on 04/26/2024).

[10] "Carbon free energy for Google Cloud regions," https://cloud.google.com/sustainability/region-carbon, (Accessed on 04/26/2024).

[11] "CONDA: Managing Environments," https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html, (Accessed on 04/26/2024).

[12] "CXL Memory eXpander Controller (MXC)," https://www.montage-tech.com/MXC, (Accessed on 04/26/2024).

[13] "Data center rack density: How high can it go?" https://www.sdxcentral.com/articles/analysis/data-center-rack-density-how-high-can-it-go/2023/09/, (Accessed on 04/26/2024).

[14] "Dell LCA, Poweredge r740," https://corporate.delltechnologies.com/content/dam/digitalassets/active/en/unauth/data-sheets/products/servers/lca_poweredge_r740.pdf, (Accessed on 04/26/2024).

[15] "Electricity Data - U.S. Energy Information Administration (EIA)," https://www.eia.gov/electricity/data.php#summary, (Accessed on 04/26/2024).

[16] "Envoy proxy," https://www.envoyproxy.io/, (Accessed on 04/26/2024).

[17] "Google Cloud Machine Types," https://cloud.google.com/compute/docs/machine-resource, (Accessed on 04/26/2024).

[18] "GreenSKU-Model," https://github.com/Azure/AzurePublicDataset.

[19] "HAProxy - The Reliable, High Perf. TCP/HTTP Load Balancer," https://www.haproxy.org/, (Accessed on 04/26/2024).

[20] "Helping you pick the greenest region for your Google Cloud resources," https://cloud.google.com/blog/topics/sustainability/pick-the-google-cloud-region-with-the-lowest-co2, (Accessed on 04/26/2024).

[21] "IMEC netzero virtual fab," https://netzero.imec-int.com/, (Accessed on 04/26/2024).

[22] "Intel Announces 288-Core Sierra Forest CPU, 5th-Gen Xeon," https://www.tomshardware.com/news/intel-announces-288-core-processor-5th-gen-xeon-arrives-december-14, (Accessed on 04/26/2024).

[23] "Intel Core i7-5960X, -5930K And -5820K CPU Review: Haswell-E Rises," https://www.tomshardware.com/reviews/intel-core-i7-5960x-haswell-e-cpu,3918.html, (Accessed on 04/26/2024).

[24] "M.2 E1.S Adapter," https://www.microsatacables.com/m-2-nvme-ssd-to-edsff-e1-s-interface-adapter-card, (Accessed on 04/26/2024).

[25] "Makersite Data Platform," https://makersite.io/makersite-ai-data-apps/, (Accessed on 04/26/2024).

[26] "Microsoft Datacenter Statistics and Sustainability Fact Sheets," https://datacenters.microsoft.com/, (Accessed on 04/26/2024).

[27] "Microsoft will be carbon negative by 2030," https://blogs.microsoft.com/blog/2020/01/16/microsoft-will-be-carbon-negative-by-2030/, (Accessed on 04/26/2024).

[28] "New Seventh-Generation General Purpose Amazon EC2 Instances," https://aws.amazon.com/blogs/aws/new-seventh-generation-general-purpose-amazon-ec2-instances-m7i-flex-and-m7i/, (Accessed on 04/26/2024).

[29] "Nytro 3530 - 1920GB | Seagate US," https://www.seagate.com/esg/planet/product-sustainability/nytro-3530-sustainability-report/, (Accessed on 04/26/2024).

[30] "PEX 8732," https://www.broadcom.com/products/pcie-switches-retimers/pcie-switches/pex8732, (Accessed on 04/26/2024).

[31] "Server Rack Sizes Matter: 3 Critical Rack Server Dimensions," https://www.vertiv.com/en-emea/about/news-and-insights/articles/educational-articles/server-rack-sizes-matter-get-these-3-critical-rack-server-dimensions-right/, (Accessed on 04/26/2024).

[32] "Smart Memory Controllers — Microchip Technology," https://www.microchip.com/en-us/products/memory/smart-memory-controllers, (Accessed on 04/26/2024).

[33] "The Xapian Project," https://xapian.org/, (Accessed on 04/26/2024).

[34] "Understanding the Advantages and Disadvantages of Linear Regulators," https://www.digikey.com/en/articles/understanding-the-advantages-and-disadvantages-of-linear-regulators, (Accessed on 04/26/2024).

[35] B. Acun, B. Lee, F. Kazhamiaka, K. Maeng, M. Chakkaravarthy, U. Gupta, D. Brooks, and C.-J. Wu, "Carbon Explorer: A Holistic Approach for Designing Carbon Aware Datacenters," *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2023.

[36] D. Albano, D. Abood, A. Armstrong, R. Murdoch, and J. Whitney, "Cloud Computing and Sustainability: The Environmental Benefits of Moving to the Cloud," 2010, (Whitepaper).

[37] P. Ambati, I. Goiri, F. Frujeri, A. Gun, K. Wang, B. Dolan, B. Corell, S. Pasupuleti, T. Moscibroda, S. Elnikety, M. Fontoura, and R. Bianchini, "Providing SLOs for Resource-Harvesting VMs in Cloud Platforms," in *Symposium on Operating Systems Design and Implementation*, 2020.

[38] N. Amla, D. D. Silva, M. Littman, and M. Parashar, "NSF on Chien's Grand Challenge for Sustainability," *Communications of the ACM*, vol. 66, pp. 36–37, 2023.

[39] H. Barbalho, P. Kovaleski, B. Li, L. Marshall, M. Molinaro, A. Pan, E. Cortez, M. Leao, H. Patwari, Z. Tang, T. Santos, L. R. Gonçalves, D. Dion, T. Moscibroda, and I. Menache, "Virtual Machine Allocation with Lifetime Predictions," *Conference on Machine Learning and Systems*, 2023.

[40] N. Bashir, T. Guo, M. Hajiesmaili, D. Irwin, P. Shenoy, R. Sitaraman, A. Souza, and A. Wierman, "Enabling Sustainable Clouds: The Case for Virtualizing the Energy System," in *Symposium on Cloud Computing*, 2021.

[41] D. S. Berger, D. Ernst, H. Li, P. Zardoshti, M. Shah, S. Rajadnya, S. Lee, L. Hsu, I. Agarwal, M. D. Hill, and R. Bianchini, "Design Tradeoffs in CXL-Based Memory Pools for Public Cloud Platforms," *IEEE Micro*, vol. 43, pp. 30–38, 2023.

[42] R. Boone, ""Capacity Prediction" instead of "Capacity Planning": How Uber Uses ML to Accurately Forecast Resource Utilization," https://www.usenix.org/conference/srecon18americas/presentation/boone, 2018, (Accessed on 04/26/2024).

[43] J. Carlson, *Redis in Action*. Simon and Schuster, 2013, (Book).

[44] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A Cloud-Scale Acceleration Architecture," in *International Symposium on Microarchitecture*, 2016.

[45] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services," in *Symposium on Networked Systems Design and Implementation*, 2008.

[46] S. Chen, C. Delimitrou, and J. F. Martínez, "PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[47] A. A. Chien, "Computing's Grand Challenge for Sustainability," *Communications of the ACM*, vol. 65, p. 5, 2022.

[48] A. A. Chien, C. Zhang, and L. Lin, "Beyond PUE: Flexible Datacenters Empowering the Cloud to Decarbonize," *Workshop on Sustainable Computer Systems*, 2022.

[49] S. Chopra, G. Reinhardt, and M. Dada, "The Effect of Lead Time Uncertainty on Safety Stocks," *Decision Sciences*, vol. 35, pp. 1–24, 2004.

[50] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource Central: Understanding and Predicting Work-

loads for Improved Resource Management in Large Cloud Platforms," in *Symposium on Operating Systems Principles*, 2017.

[51] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware Scheduling for Heterogeneous Datacenters," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2013.

[52] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-Efficient and QoS-Aware Cluster Management," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[53] M. Eriksen, K. Veeraraghavan, Y. Abdulghani, A. Birchall, P.-Y. Chou, R. Cornew, A. Kabiljo, R. K. S, M. Lieuw, J. Meza, S. Michelson, T. Rohloff, H. Russell, J. Qin, and C. Tang, "Global Capacity Management With Flux," in *Symposium on Operating Systems Design and Implementation*, 2023.

[54] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *International Symposium on Computer Architecture*, 2011.

[55] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *Symposium on Networked Systems Design and Implementation*, 2018.

[56] E. Frachtenberg, A. Heydari, H. Li, A. Michael, J. Na, A. Nisbet, and P. Sarti, "High-Efficiency Server Design," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[57] K. Fu, W. Zhang, Q. Chen, D. Zeng, X. Peng, W. Zheng, and M. Guo, "QoS-Aware and Resource Efficient Microservice Deployment in Cloud-Edge Continuum," in *International Parallel and Distributed Processing Symposium*, 2021.

[58] M. Furtak and M. P. Wittie, "The Oddness of Webpages," in *Network Traffic Measurement and Analysis Conference*, 2019.

[59] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: Practical and Scalable ML-Driven Performance Debugging in Microservices," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[60] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[61] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers," *Transactions on Computer Systems*, vol. 30, pp. 1–26, 2012.

[62] J. Gao, H. Wang, and H. Shen, "Smartly Handling Renewable Energy Instability in Supporting A Cloud Datacenter," in *International Parallel and Distributed Processing Symposium*, 2020.

[63] P. Garraghan, P. Townend, and J. Xu, "An Analysis of the Server Characteristics and Resource Utilization in Google Cloud," in *International Conference on Cloud Engineering*, 2013.

[64] U. Gupta, M. Elgamal, G. Hills, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "ACT: Designing Sustainable Computer Systems With An Architectural Carbon Modeling Tool," in *International Symposium on Computer Architecture*, 2022.

[65] U. Gupta, Y. G. Kim, S. Lee, J. Tse, H.-H. S. Lee, G.-Y. Wei, D. Brooks, and C.-J. Wu, "Chasing Carbon: The Elusive Environmental Footprint of Computing," in *International Symposium on High-Performance Computer Architecture*, 2021.

[66] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, and T. Moscibroda, "Protean: VM Allocation Service at Scale," in *Symposium on Operating Systems Design and Implementation*, 2020.

[67] J. Hamilton, "Cooperative Expendable Micro-Slice Servers (CEMS): Low Cost, Low Power Servers for Internet-Scale Services," in *Conference on Innovative Data Systems Research*, 2009.

[68] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker, *OLTP through the Looking Glass, and What We Found There*, 2018, (Book).

[69] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective," in *International Symposium on High Performance Computer Architecture*, 2018.

[70] C.-H. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski, "Adrenaline: Pinpointing and Reining in Tail Queries with Quick Voltage Boosting," in *International Symposium on High Performance Computer Architecture*, 2015.

[71] M. Jalili, I. Manousakis, I. n. Goiri, P. A. Misra, A. Raniwala, H. Alissa, B. Ramakrishnan, P. Tuma, C. Belady, M. Fontoura, and R. Bianchini, "Cost-Efficient Overclocking in Immersion-Cooled Datacenters," in *International Symposium on Computer Architecture*, 2021.

[72] Z. Jia and E. Witchel, "Nightcore: Efficient and Scalable Serverless Computing for Latency-Sensitive, Interactive Microservices," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[73] N. Jones, "How to stop data centres from gobbling up the world's electricity," *Nature*, vol. 561, pp. 163–166, 2018.

[74] H. Kasture and D. Sanchez, "TailBench: A Benchmark Suite and Evaluation Methodology for Latency-Critical Applications," in *International Symposium on Workload Characterization*, 2016.

[75] D. Kline, N. Parshook, X. Ge, E. Brunvand, R. Melhem, P. K. Chrysanthis, and A. K. Jones, "GreenChip: A tool for evaluating holistic sustainability of modern computing systems," *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 322–332, 2019.

[76] B. Knowles, "ACM TechBrief: Computing and Climate Change," 2021.

[77] A. Kopytov, "SysBench: a system performance benchmark," http://sysbench.sourceforge.net/, (Accessed on 04/26/2024).

[78] C. Koronen, M. Åhman, and L. J. Nilsson, "Data centres in future European energy systems—energy efficiency, integration and policy," *Energy Efficiency*, vol. 13, pp. 129–144, 2020.

[79] A. Kumbhare, R. Azimi, I. Manousakis, A. Bonde, F. V. Frujeri, N. Mahalingam, P. Misra, S. A. Javadi, B. Schroeder, M. Fontoura, and R. Bianchini, "Prediction-Based Power Oversubscription in Cloud Platforms," in *Annual Technical Conference*, 2021.

[80] K.-F. Lee, H.-W. Hon, and R. Reddy, "An Overview of the SPHINX Speech Recognition System," *Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, pp. 35–45, 1990.

[81] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, "Pond: CXL-Based Memory Pooling Systems for Cloud Platforms," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2023.

[82] H. Li, M. Hao, S. Novakovic, V. Gogte, S. Govindan, D. R. Ports, I. Zhang, R. Bianchini, H. S. Gunawi, and A. Badam, "LeapIO: Efficient and Portable Virtual NVMe Storage on ARMSoCs," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[83] A. Liguori, "The Nitro Project–Next Generation AWS Infrastructure," in *Symposium on High Performance Chips*, 2018.

[84] R. M. Llamas, "Capacity Planning at Scale," https://www.usenix.org/conference/srecon16europe/program/medrano-llamas, 2016, (Accessed on 04/26/2024).

[85] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards Energy Proportionality for Large-Scale Latency-Critical Workloads," in *International Symposium on Computer Architecture*, 2014.

[86] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis," in *Symposium on Cloud Computing*, 2021.

[87] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, J. He, G. Yang, and C. Xu, "Erms: Efficient Resource Management for Shared Microservices with SLA Guarantees," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.

[88] J. Lyu, J. Wang, K. Frost, C. Zhang, C. Irvene, E. Choukse, R. Fonseca, R. Bianchini, F. Kazhamiaka, and D. S. Berger, "Myths and Misconceptions Around Reducing Carbon Embedded in Cloud Platforms," in *Workshop on Sustainable Computer Systems*, 2023.

[89] J. Lyu, M. You, C. Irvene, M. Jung, T. Narmore, J. Shapiro, L. Marshall, S. Samal, I. Manousakis, L. Hsu, P. Subbarayalu, A. Raniwala, B. Warrier, R. Bianchini, B. Schroeder, and D. S. Berger, "Hyrax: Fail-in-Place Server Operation in Cloud Platforms," in *Symposium on Operating Systems Design and Implementation*, 2023.

[90] Y. Mao, E. Kohler, and R. T. Morris, "Cache Craftiness for Fast Multicore Key-Value Storage," in *European Conference on Computer Systems*, 2012.

[91] S. McAllister, B. Berg, J. Tutuncu-Macias, J. Yang, S. Gunasekar, J. Lu, D. S. Berger, N. Beckmann, and G. R. Ganger, "Kangaroo: Caching Billions of Tiny Objects on Flash," in *Symposium on Operating Systems Principles*, 2021.

[92] A. Mirhosseini, S. Elnikety, and T. F. Wenisch, "Parslo: A Gradient Descent-based Approach for Near-optimal Partial SLO Allotment in Microservices," in *Symposium on Cloud Computing*, 2021.

[93] A. Mirhosseini, A. Sriraman, and T. F. Wenisch, "Enhancing Server Efficiency in the Face of Killer Microseconds," in *International Symposium on High Performance Computer Architecture*, 2019.

[94] R. K. Pachauri and A. Reisinger, *Climate Change 2007: Synthesis Report. Contribution of Working Groups I, II and III to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change*. IPCC, 2007, (Book).

[95] A. Parayil, J. Zhang, X. Qin, Í. Goiri, L. Huang, T. Zhu, and C. Bansal, "Towards Cloud Efficiency with Large-scale Workload Characterization," *arXiv preprint*, 2024.

[96] P. Patel, Z. Gong, S. Rizvi, E. Choukse, P. Misra, T. Anderson, and A. Sriraman, "Towards Improved Power Management in Cloud GPUs," *Computer Architecture Letters*, vol. 22, pp. 141–144, 2023.

[97] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices," in *Symposium on Operating Systems Design and Implementation*, 2020.

[98] H. Qiu, W. Mao, C. Wang, H. Franke, A. Youssef, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer, "AWARE: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems," in *Annual Technical Conference*, 2023.

[99] A. Radovanović, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, and N. Care, "Carbon-Aware Computing for Datacenters," *Transactions on Power Systems*, vol. 38, pp. 1270–1280, 2022.

[100] K. Rzadca, P. Findeisen, J. Świderski, P. Zych, P. Broniek, J. Kusmierek, P. K. Nowak, B. Strack, P. Witusowski, S. Hand, and J. Wilkes, "Autopilot: Workload Autoscaling at Google Scale," in *European Conference on Computer Systems*, 2020.

[101] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash Reliability in Production: The Expected and the Unexpected," in *Conference on File and Storage Technologies*, 2016.

[102] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural Implications of Function-as-a-Service Computing," in *International Symposium on Microarchitecture*, 2019.

[103] D. D. Sharma, R. Blankenship, and D. S. Berger, "An Introduction to the Compute Express Link (CXL) Interconnect," *arXiv preprint arXiv:2306.11227*, 2023.

[104] R. Sharma, A. Mathur, R. Sharma, and A. Mathur, *Introduction to Traefik*. Springer, 2021, (Book).

[105] J. Shuja, K. Bilal, S. A. Madani, M. Othman, R. Ranjan, P. Balaji, and S. U. Khan, "Survey of Techniques and Architectures for Designing Energy-Efficient Data Centers," *IEEE Systems Journal*, vol. 10, pp. 507–519, 2016.

[106] T. Siddiqua, V. Sridharan, S. E. Raasch, N. DeBardeleben, K. B. Ferreira, S. Levy, E. Baseman, and Q. Guan, "Lifetime Memory Reliability Data from the Field," in *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2017.

[107] D. Simchi-Levi and M. A. Trick, "Introduction to "Little's Law as Viewed on Its 50th Anniversary"," *Operations Research*, vol. 59, pp. 535–535, 2011.

[108] U. Sirin, P. Tözün, D. Porobic, A. Yasin, and A. Ailamaki, "Micro-Architectural Analysis of In-Memory OLTP: Revisited," *International Conference on Management of Data*, 2021.

[109] R. Soni, *Nginx*. Springer, 2016, (Book).

[110] A. Souza, N. Bashir, J. Murillo, W. Hanafy, Q. Liang, D. Irwin, and P. Shenoy, "Ecovisor: A Virtual Energy System for Carbon-Efficient Applications," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2023.

[111] V. Sridharan and D. Liberty, "A Study of DRAM Failures in the Field," in *International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012.

[112] A. Sriraman, A. Dhanotia, and T. F. Wenisch, "SoftSKU: Optimizing Server Architectures for Microservice Diversity @Scale," in *International Symposium on Computer Architecture*, 2019.

[113] A. Sriraman and T. F. Wenisch, "μTune: Auto-Tuned Threading for OLDI Microservices," in *Conference on Operating Systems Design and Implementation*, 2018.

[114] J. Stojkovic, C. Liu, M. Shahbaz, and J. Torrellas, "μManycore: A Cloud-Native CPU for Tail at Scale," in *International Symposium on Computer Architecture*, 2023.

[115] J. Switzer, G. Marcano, R. Kastner, and P. Pannuto, "Junkyard Computing: Repurposing Discarded Smartphones to Minimize Carbon," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2023.

[116] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: the Next Generation," in *European Conference on Computer Systems*, 2020.

[117] A. Tomlinson and G. Porter, "Something Old, Something New: Extending the Life of CPUs in Datacenters," in *Workshop on Sustainable Computer Systems*, 2022.

[118] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic Resource Allocation and Power Management in Virtualized Data Centers," in *Network Operations and Management Symposium*, 2010.

[119] B. Vamanan, H. B. Sohail, J. Hasan, and T. Vijaykumar, "TimeTrader: Exploiting Latency Tail to Save Datacenter Energy for Online Search," in *International Symposium on Microarchitecture*, 2015.

[120] C. Verbowski, E. Thayer, P. Costa, H. Leather, and B. Franke, "Right-Sizing Server Capacity Headroom for Global Online Services," in *International Conference on Distributed Computing Systems*, 2018.

[121] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *European Conference on Computer Systems*, 2015.

[122] J. von Kistowski, K.-D. Lange, J. A. Arnold, S. Sharma, J. Pais, and H. Block, "Measuring and Benchmarking Power Consumption and Energy Efficiency," in *International Conference on Performance Engineering*, 2018.

[123] J. Wang, "GreenSKU-Model: ISCA Artifact." [Online]. Available: https://doi.org/10.5281/zenodo.10896255

[124] J. Wang, U. Gupta, and A. Sriraman, "Characterizing Datacenter Server Generations for Lifetime Extension and Carbon Reduction," in *Workshop on NetZero Carbon Computing*, 2023.

[125] J. Wang, U. Gupta, and A. Sriraman, "Giving Old Servers New Life at Hyperscale," in *Workshop on Hot Topics in System Infrastructure*, 2023.

[126] J. Wang, U. Gupta, and A. Sriraman, "Peeling Back the Carbon Curtain: Carbon Optimization Challenges in Cloud Computing," in *Workshop on Sustainable Computer Systems*, 2023.

[127] R. Wang, L. Zhang, Y. Yang, Y. Zhen, B. Long, T. Wang, V. Govindaraj, T. Palino, S. Tata, and V. Nair, "CapPredictor: A Capacity Headroom Prediction Framework in Cloud," in *Workshop on Cloud Intelligence*, 2020.

[128] Y. Wang, K. Arya, M. Kogias, M. Vanga, A. Bhandari, N. J. Yadwadkar, S. Sen, S. Elnikety, C. Kozyrakis, and R. Bianchini, "SmartHarvest: Harvesting Idle CPUs Safely and Efficiently in the Cloud," in *European Conference on Computer Systems*, 2021.

[129] P. Wiesner, I. Behnke, D. Scheinert, K. Gontarska, and L. Thamsen, "Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud," in *International Middleware Conference*, 2021.

[130] K. Wołk and K. Marasek, "Real-Time Statistical Speech Translation," in *New Perspectives in Information Systems and Technologies*, vol. 1, 2014, pp. 107–113.

[131] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. Aga, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H.-H. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. Hazelwood, "Sustainable AI: Environmental Implications, Challenges and Opportunities," in *Conference on Machine Learning and Systems*, 2022.

[132] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song, "Dynamo: Facebook's Data Center-Wide Power Management System," in *International Symposium on Computer Architecture*, 2016.

[133] J. H. Yahya, H. Volos, D. B. Bartolini, G. Antoniou, J. S. Kim, Z. Wang, K. Kalaitzidis, T. Rollet, Z. Chen, Y. Geng, O. Mutlu, and Y. Sazeides,

"AgileWatts: An Energy-Efficient CPU Core Idle-State Architecture for Latency-Sensitive Server Applications," in *International Symposium on Microarchitecture*, 2022.

[134] C. Zhang, A. G. Kumbhare, I. Manousakis, D. Zhang, P. A. Misra, R. Assis, K. Woolcock, N. Mahalingam, B. Warrier, D. Gauthier, L. Kunnath, S. Solomon, O. Morales, M. Fontoura, and R. Bianchini, "Flex: High-Availability Datacenters With Zero Reserved Power," in *International Symposium on Computer Architecture*, 2021.

[135] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: ML-Based and QoS-Aware Resource Management for Cloud Microservices," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[136] Y. Zhong, D. S. Berger, C. Waldspurger, I. Agarwal, R. Agarwal, F. Hady, K. Kumar, M. D. Hill, M. Chowdhury, and A. Cidon, "Managing Memory Tiers with CXL in Virtualized Environments," in *Symposium on Operating Systems Design and Implementation*, 2024.

[137] Z. Zhou, Y. Zhang, and C. Delimitrou, "AQUATOPE: QoS-and-Uncertainty-Aware Resource Management for Multi-Stage Serverless Workflows," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.

Appendix A
Artifact Appendix

A. Abstract

The artifact provides the carbon model that we use to evaluate our *GreenSKU* designs' carbon emissions (as described in §V). The artifact includes Python source code that implements the carbon and maintenance models. As input to the carbon model, it includes a set of open-source embodied emissions and power numbers for (1) the components we use to evaluate and design our *GreenSKUs* and (2) the Gen3 baseline SKU. The artifact also provides source code to reproduce the open-source carbon savings results reported in §A-F.

In summary, this artifact contributes (1) a reusable carbon model to calculate the carbon emissions of SKU designs, (2) open-source data that can be used as input to the carbon model, and (3) code to reproduce our paper's open-source results.

B. Artifact Check-List (meta-information)

- **Program:** Python3 scripts and Jupyter Notebook.
- **Data set:** We provide all required data sets in our artifact repository [18]. This repository includes open-source data sets for (1) carbon numbers for components used in our servers, (2) compute cluster information, and (3) data center carbon intensities.
- **Run-time environment:** Any standard Python environment.
- **Hardware:** No special hardware is required.
- **Output:** The Jupyter notebook will output the exact numbers, tables, and figures that can be directly compared against expected results.
- **Experiments:** We provide a Jupyter Notebook that uses the model and the inputs we provide to replicate the carbon savings calculations and analysis shown in the paper.
- **How much disk space is required (approximately)?** About 500MB (mainly for the Python environment).
- **How much time is needed to prepare the workflow (approximately)?** Less than an hour.
- **How much time is needed to complete the experiments (approximately)?** A few minutes.
- **Is the artifact publicly available?** The artifact is available on Zenodo: https://doi.org/10.5281/zenodo.10896255 and GitHub: https://github.com/Azure/AzurePublicDataset.
- **Code licenses (if publicly available)?** See GitHub: https://github.com/Azure/AzurePublicDataset.

C. Artifact Description

Access. The live repository [18], which contains all the information about performing the artifact evaluation and reproducing the desired results, is available on GitHub: https://github.com/Azure/AzurePublicDataset. The artifact is also archived and available on Zenodo with the DOI: https://doi.org/10.5281/zenodo.10896255.

Hardware dependencies. Any machine with a Python environment can execute our scripts.

Software dependencies. The carbon model requires a working Python environment. We suggest an Anaconda environment [11], installed through Miniconda (see the live repository for details on installation and setup). All Python dependencies are provided and are installed when performing the evaluation.

Data sets. In §VI, we use closed-source, internal carbon data to calculate our *GreenSKUs*' carbon savings. While this data is useful within Azure, we cannot use it to describe our *GreenSKU*'s carbon savings. Thus, we collect open-source carbon data from public sources to both explain our carbon model in §V and to provide a reproducible version of our results (§A-F). For brevity, in this appendix, we only include the data that is used in §V. However, the full data set and explanations for how we source each data value is available in the artifact's GitHub repository [18] within `analysis/GreenSKU-Framework`. Note that all paths in this artifact are relative to this subdirectory.

The open-source carbon data we use reasonably aligns with our internal carbon data, with inevitable differences, due to the specifics of Azure's supply chain and hardware sourcing. Thus, we do not officially endorse these data sources and we do not claim that this open-source data represents internal carbon values. Rather, we collect this open-source data to show an example of how to calculate a *GreenSKU*'s carbon savings.

In Table V, we show the TDP and embodied emissions values used in Sec. V's operational and embodied emissions example calculations. We require some additional parameters to convert these raw carbon numbers into data center-level carbon estimates. We detail these parameters in Table VI. The complete data set and derivations for the data we provide is available in `data/README.md` in our GitHub repository [18].

Component	**TDP (W)**	**Embodied carbon ($kgCO_2e$)**
AMD Bergamo CPU	400 [5]	28.3 [4], [64]
DRAM (DDR5)	0.37 per GB [23]	1.65 per GB [14], [64]
DRAM (DDR4)	0.37 per GB [23]	0 (reused)
SSD	5.6 per TB [29]	17.3 per TB [29]
CXL Controller	5.8 [30]	2.5 [32], [64]
Rack misc.	500	500

TABLE V
OPEN-SOURCE TDP AND EMBODIED CARBON VALUES FOR COMPONENTS USED IN THE CARBON MODEL'S EXAMPLE CALCULATION.

Parameter	**Value**
Carbon intensity	0.1 $kgCO_2e$/kWh [1]
Lifetime	6 years [88]
Derate factor at 40% SPEC throughput	0.44 [122]
Rack space capacity	42U (−10U overhead) [31]
Rack power capacity	15kW [13]
CPU voltage regulator loss	1.05 [34]

TABLE VI
OPEN-SOURCE MODEL PARAMETERS USED FOR THE CARBON MODEL'S EXAMPLE CALCULATION.

D. Artifact Installation

We now detail our artifact's installation instructions. The same instructions are available in our GitHub repository [18], which we suggest using to more easily copy commands. The installation should take less than half an hour.

First, install Anaconda [11]. Once installed, create the `conda` environment:

```
$ conda create --name carbon_model \
  python=3.9
```

To activate the virtual environment, run the following command:

Result in paper to reproduce	Run time	Output file(s)
Last three columns of Table VIII	<1 minute	`figures/generated_figures/Table_VIII.csv`
Appendix A-F claims: "*We re-calculate the savings we report to find an average cluster-level savings of **14%**, leading to an overall data center-level savings of **7%**.*"	<1 minute	`figures/generated_figures/cluster_savings.txt` `figures/generated_figures/dc_savings.txt`
Figure 12	1 minute	`figures/generated_figures/Figure_12.png`

TABLE VII
RESULTS TO REPRODUCE AND THEIR RESPECTIVE RUN TIMES AND OUTPUT FILES.

```
$ conda activate carbon_model
```

Next, clone our GitHub repository [18] into your working directory. Then, run the following commands with the environment activated to install the required dependencies.

```
$ cd AzurePublicDataset
$ cd analysis/GreenSKU-Framework
$ pip install -r requirements.txt
$ conda install jupyterlab
```

Once this is done, installation is complete.

E. Experiment Workflow

Once installed, the workflow to validate the model results is fully contained within `notebooks/carbon_savings.ipynb`. This notebook uses the carbon model source code, the details of which are in `src/README.md`.

The notebook performs the following tasks in order:

- Imports required packages, including the carbon model and maintenance model Python modules.
- Configures the required model parameters and steps through how parameters are calculated/derived.
- Performs the carbon savings calculations using the carbon model to reproduce the results detailed in the next section.

F. Evaluation and Expected Results

This artifact only reproduces results that can be obtained from the carbon model alone. All results will be generated from running `notebooks/carbon_savings.ipynb`. Table VII describes the results to reproduce, their run time, and where the results are outputted. The output exactly matches Fig. 12, as the model is fully deterministic for the same inputs.

We now show the main results of the paper reproduced using the open-source data we provide.

Per-core carbon savings. We use the open-source component carbon numbers outlined in §A-C to reproduce Table IV using our open-source data, which we show in Table VIII. We find that the relative savings in Table IV are similar to Table VIII. The net carbon savings of *GreenSKU-Full* in Table VIII is similar to the reported savings using internal numbers: 26% vs 28%.

Cluster carbon savings across carbon intensities We also use the open-source data to recreate Fig. 11, one of our main results, containing the cluster-level carbon savings across carbon intensities. The reproduced results using open-source data are shown in Fig. 12. While there are differences in terms of the achieved carbon savings for each design, we still see that reuse is especially effective at lower carbon intensities, where embodied emissions dominates. We re-calculate the savings we report to determine an average cluster-level savings of 14%, leading to an overall data center-level savings of 7%.

SKU Config.	**# Core**	**# × DIMM (GB)**	**# × SSD (TB)**	**Operational Savings**	**Embodied Savings**	**Total Savings**
Baseline	80	12 × 64	6 × 2	-	-	-
Baseline-*Resized*	80	10 × 64	6 × 2	6%	10%	8%
GreenSKU-Efficient	128	12 × 96	5 × 4	16%	14%	15%
GreenSKU-CXL	128	12 × 64 8 × 32 CXL	5 × 4	15%	32%	24%
GreenSKU-Full	128	12 × 64 8 × 32 CXL	2 × 4 12 × 1 Reuse	14%	38%	26%

TABLE VIII
OPEN-SOURCE PER-CORE OPERATIONAL, EMBODIED, AND TOTAL CARBON SAVINGS (CALCULATED BASED ON THE AVERAGE CARBON INTENSITY FOR MAJOR AZURE REGIONS) RELATIVE TO OUR GEN3 BASELINE SERVER FOR FOUR INCREMENTAL GREENSKU CONFIGURATIONS.

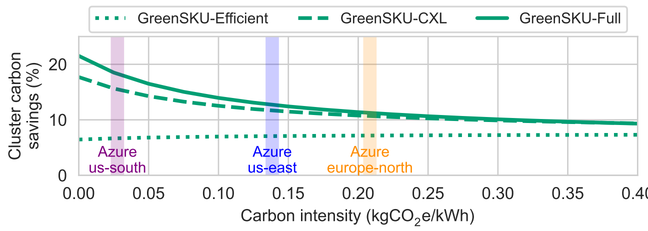


Fig. 12. End-to-end cluster-level carbon savings relative to baseline clusters across a range of carbon intensities evaluated for our three *GreenSKUs*. Vertical lines are estimated carbon intensities for the energy used by three Azure data center regions [26]. Calculated using open-source data.

G. Experiment Customization

No changes to the original scripts/notebook are necessary to reproduce the results. The model, however, is generalizeable to evaluate other server designs, and the original parameters and inputs can be changed.

H. Methodology

Submission, reviewing, and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html