GreenScale: Carbon Optimization for Edge Computing

Yonglak Son, Udit Gupta, Andrew McCrabb, Young Geun Kim, Valeria Bertacco, David Brooks, and Carole-Jean Wu

***Abstract*—Given billions of mobile users, the environmental impact of edge computing is significant. To address this, future applications need to execute computations on a *green component* which is fueled by renewable energy sources. However, because of the intermittent nature of the renewable energy sources, the carbon intensity of computing components can significantly vary with *location* and *time* of use. This poses a new challenge for edge applications — deciding *when* and *where* to run computations across consumer devices at the edge and servers in the cloud. Such scheduling decisions become more complicated with the amortization of the rising embodied emissions and stochastic runtime variance. This work proposes *GreenScale*, an intelligent execution scaling engine that accurately selects the carbon-optimal execution target for edge applications in different runtime environments. Our evaluation with three representative categories of applications (i.e., AI, Game, and AR/VR) demonstrate that the carbon emissions of the applications can be reduced by 35.2%, on average, with *GreenScale*.**

***Index Terms*—Edge-cloud computing, carbon footprint optimization, computation offloading, reinforcement learning.**

I. Introduction

DIGITAL technology advancement has improved many dimensions of our lives. Despite the positive societal impact, the Information and Computing Technology (ICT) sector has caused significant energy and environmental overheads worldwide. As of 2019, estimates show that the carbon emissions from ICT account for 2% of worldwide emissions, half that of the aviation industry [1]. It is expected to increase to 14% of worldwide emissions in the next decade [1].

Recognizing the environmental implications, computing infrastructures have been significantly optimized for hyperscale datacenters [2], [3] in the past decades, primarily focused on operational efficiency. Further operational efficiency improvement is increasingly challenging. In addition to the efficiency optimization, renewable energy, such as solar and wind, is increasingly adopted to reduce computing's carbon footprint [4]–[6]. However, as renewable energy generation is intermittent [7], [8], the carbon intensity of computing varies with *location* and *time* of use.

Furthermore, recent works have highlighted a fundamental carbon bottleneck shift from operational to embodied carbon emissions [9], [10]. Taking consumer electronics, such as Google Pixel3 [11], as an example, embodied carbon footprint accounts for 50.6% of its total carbon footprint while running state-of-the-art edge applications, outweighing the operational carbon footprint. To minimize the total carbon footprint of computing, sustainability-driven optimization techniques must consider the role of embodied carbon — an important but overlooked aspect. While carbon modeling tools [10], [12], [13] and metrics [14] are starting to emerge for software development [15], [16] and hardware design [17], [18] considering the embodied emissions, prior works in carbon-aware scheduling have primarily focused on operational emissions only [19]–[21].

Recent work has started considering the design space of total carbon footprint for carbon-aware datacenter management [19]–[21], but they have not looked into the carbon optimization of scheduling for edge computing by considering both operational *and* embodied carbon emissions of computing and networking components. While [7], [22] consider embodied emission for computation shifts across datacenters, they do not consider the emissions of local edge devices and network infrastructure — their emissions are also significant though. Making mobile edge applications *green* with minimal carbon impact to the planet matters, especially given its scale — with billions of mobile users [23], it is expected that the annual carbon emissions of edge computing will be more than 481.7MtCO_2 in 2025 [24]–[30], accounting for 63.0% of the total ICT emissions.

Existing mobile applications can adopt scheduling policies to maximize energy efficiency, in order to prolong battery life while meeting application-specific performance requirements. Prior works have considered hardware heterogeneity on-device [31]–[33] and that across the edge-cloud computing spectrum [34], [35]. However, these schedulers are not tailor-designed for carbon optimization which has distinct features with the energy optimization — i.e., heterogeneous carbon intensity and embodied emissions. More recently, [36] proposed a carbon-aware scheduler which tries to minimize the CF of edge computing by considering heterogeneous carbon intensity. However, [36] does not consider heterogeneous charging behaviors of edge devices, embodied emissions, and runtime variance, which also have a significant impact on the overall CF of edge computing.

Yonglak Son and Young Geun Kim are with the Department of Computer Science and Engineering, Korea University, Seoul, South Korea (e-mail: yonglak_son@korea.ac.kr; younggeun_kim@korea.ac.kr).

Udit Gupta is with the Department of Electrical and Computer Engineering, Cornell Tech, Roosevelt Island, NY, USA (e-mail: ugupta@cornell.edu).

Andrew McCrabb and Valeria Bertacco are are with the Department of Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA (e-mail: mccrabb@umich.edu; vale@umich.edu).

David Brooks is with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA (e-mail: dbrooks@g.harvard.edu).

Carole-Jean Wu is with the AI Research, Meta, Cambridge, MA, USA (e-mail: carolejean.wu@gmail.com).

In this paper, we propose *GreenScale* — a reinforcement learning-based scheduler — that forms the foundation for *green* mobile applications. *GreenScale* minimizes the total carbon impact for edge computing while meeting application-specific performance requirements. *GreenScale* learns and adaptively acts on (1) varying carbon intensity of energy sources over time and across different geographical locations, (2) embodied carbon emissions, and (3) stochastic performance variance from resource interference and network instability. In our evaluation, *GreenScale* improves the carbon emissions of the important edge application categories — AI, Game, and AR/VR — by 35.2%, on average, over the state-of-the-art scheduler. The core contributions of this work can be summarized as follows.

1) We provide an in-depth characterization of carbon design space for edge computing. Our exploration demonstrates that the carbon-optimal execution target for edge computing significantly varies with the impact of runtime environments *and* the amortization of embodied emissions (Section IV).
2) We propose a carbon-aware execution scaling engine, *GreenScale*, that accurately selects the carbon-optimal execution target for edge-cloud applications in different runtime environments (Section V).
3) To demonstrate the feasibility and practicality of the proposed execution scaling engine, we implement and evaluate *GreenScale* with a variety of edge computing use cases using real systems and devices (Section VII).

II. Background

A. Edge Computing

Applications aiming to run on the client devices at the edge can exploit distributed heterogeneous computing resources in the network hierarchy, from the edge to the cloud data centers [37]. Fig. 1 shows examples of such execution scaling for edge computing.

Edge devices are the computing resources located at the edge of the network [37]. Examples of the edge devices include smartphones, smartwatches, laptops, appliances, and so on. Traditionally, edge devices acted only as frontend user interfaces, user-end sensors, or both for edge applications. Recently, with the advancements in powerful mobile systems-on-a-chip (SoCs), a varying amount of computations can be executed locally on edge devices, as shown in Fig. 1(a), removing the network overhead [38], [39]. .

Edge data center (Edge DC) refers to small-scale data center co-located with cellular base station (BS) in edge network [37]. Edge applications can offload computations to edge DC with tolerable data transmission overhead (from 5 ms to 20 ms depending on its location) [40], as transmission data (e.g., input/output of computations) only needs to pass through the edge network, as shown in Fig. 1(b). However, due to space and scalability limitations, the servers in an edge DC usually have less computing capabilities than those in a hyperscale DC [40].

Data center (DC) refers to the large-scale data center usually operated by industrial companies or cloud service providers [41]–[43]. DCs usually benefit from economies of scale for large workloads and co-locations by exploiting a large number of highly efficient co-processors, such as GPUs. However, offloading computations to a DC usually incurs higher network overheads than that to an edge DC, as the transmission data needs to pass through not only the edge network but also multiple hops of routers in the core network, as shown in Fig. 1(c).

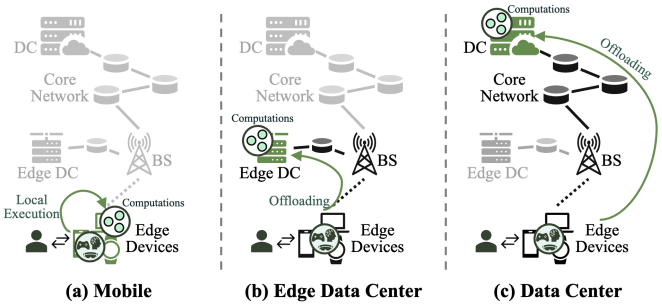

Fig. 1. Examples of execution scaling in edge computing.

Given billions of mobile users, carbon footprint (CF) of edge computing can be significant. In the next subsection. we explain the major sources of the CF in edge computing.

B. Carbon Footprint of Edge Computing

The CF of each edge computing component is composed of the operational (OP_{CF}) and embodied (E_{CF}) emissions [10], [12], [13]. The former is the emissions from the energy consumption operating the component, and the latter is the emissions for manufacturing its hardware (e.g., processor, memory, storage, etc.). As in [7], [14], the embodied emissions can be discounted based on the application runtime, *T*, over the overall lifetime of the system, *LT*, and the number of resources used by the application (e.g., the number of DC HWs allocated to the application), *N*, over that of total available resources, N_{total}. Typically, lifetime of servers and mobile devices is 3-5 years and 2-3 years, respectively [13].

$$CF = OP_{CF} + E_{CF} \times \frac{T}{LT} \times \frac{N}{N_{total}} \quad (1)$$

The OP_{CF} of a component is calculated as the product of the energy consumed by the component and the carbon intensity (CI) of the energy [13].

$$OP_{CF} = Energy \times CI \quad (2)$$

The E_{CF} can be obtained by referring to the environmental product reports of industrial companies generated by life cycle analysis (LCA) tools [44], [45], or by using the architectural carbon modeling tools [10], [12], [13]. The LCA tools quantify E_{CF} of HW components using coarse-grained information (e.g., economic cost of electronics and materials). On the other hand, the architectural tools calculate E_{CF} based on a variety of parameters (e.g., SoC area, energy per area, yield, raw materials, etc.), or its relative ratio to OP_{CF}.

In edge computing, as shown in Fig. 1, different components can be used by each user depending on which computing

component is selected as the execution (or offloading) target [37]–[39]. For example, if the user device is selected as the execution target for a user, the device may only be used for executing the computations, while the rest of the components are used for the other users (or remain idle). On the other hand, if one of the remote servers (in either an edge DC or DC) is selected as the execution target, the server and the associated network components (e.g., network interfaces of the mobile device and DC, base station, and/or core routers) may be used as the computing and communicating components, respectively, shared across the allocated users. Here, the total carbon emission of the selected execution target for a user can be calculated based on the emissions of the associated 1) computing (CF_{comp}), 2) communicating (CF_{comm}), and 3) idle (CF_{idle}) components, as in (3), where U represents the number of users co-sharing the components.

$$CF = \frac{CF_{comp}}{U_{comp}} + \frac{CF_{comm}}{U_{comm}} + \frac{CF_{idle}}{U_{idle}} \tag{3}$$

Given significant CF of edge computing, many prior works have tried to reduce emissions by focusing on minimizing operational emissions of datacenter [19]–[21] or improving energy efficiency of mobile applications [34], [35]. Unfortunately, the prior works often fail to fully address the overall CF of edge computing, which is significantly affected by various features. In the next section, we explain these features one-by-one.

III. Challenges of Carbon Optimization

In this section, we introduce various features that affect the overall CF of edge computing, which makes the carbon optimization challenging.

A. Heterogeneous Carbon Intensity

The operational carbon efficiency of each edge computing component depends on the CI of energy sources powering the component [7], [8], [21]. Renewable energy sources (i.e., wind and solar) typically have lower CI than the other energy sources. Thus, to improve the carbon efficiency of edge computing, it is crucial to run computations on a *green component* which is fueled by the renewable energy sources.

Unfortunately, generation of the renewable energy is not always available at any single location all the time [7], [8], [21]. For example, solar energy is only available with sunlight at any location, whereas wind energy is available intermittently throughout a day. The availability of these energy sources can even be affected by the location of the power grid and the climate (e.g., solar radiation, clouds, etc.). As a result, the overall CI of each power grid significantly fluctuates within a day [7], [8], [21], and the power grids in different locations have different CI, as shown in Fig. 2. This may affect the CI of the components powered by each grid, making it challenging to find the *green components*.

To this end, the CI of each component is determined by *when*, *where*, and *how* the component is charged or powered. For client devices, the CI is determined by *when* and *how* the users charge the battery [46]. On the other hand, for the server systems in a DC, the CI is determined by *which* energy source is used to power the DC and the *location* of the grid powering the DC [7], [21]. Consequently, to identify the *green component* for the execution of the computations, it is important to consider the heterogeneous CI.

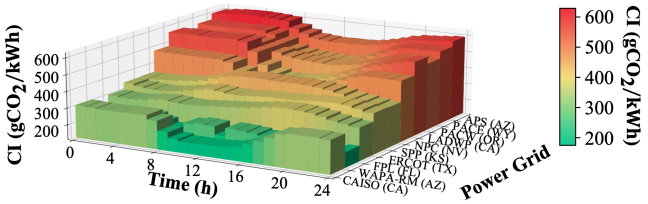

Fig. 2. Time-dependent carbon intensity of power grids in different locations of US. The x-axis shows the time, y-axis shows the power grids in different locations, z-axis and color represent the CI.

B. Amortization of Embodied Emission

Although it is not possible for edge applications to directly adjust the embodied emissions of the components, the embodied emissions can be discounted based on the application runtime over the lifetime of the components, and the number of resources used by the application over that of the total available resources [7], [14]. For example, when the computations of an application are shifted from many user devices to a DC, the operational emissions can be saved by relying on more renewable energy available at the DC. However, such computation shifts may increase the embodied emissions if more servers are allocated to handle the increased requests (i.e., rebound effect).

From the global perspective, if we distribute computations of an application to various targets, we can exploit less-powerful devices to execute the fragmented computations [16]. This extends the upgrade cycle of the devices, translating to less embodied CF overall. We can also reduce the number of servers allocated to an application exploiting user devices. The servers then can be provisioned to the other applications [7]. This potentially reduces installations of the new servers, translating to less embodied CF. According to our analysis, if the lifetime of the components is extended by 1-3 years, the total CF can be reduced by 44.2%, on average, for various edge applications. Consequently, when we determine the execution target of computations, we need to carefully consider the trade-off between the operational *and* embodied emissions.

C. Inherent Uncertainty

Execution in edge-cloud environment inherently contains uncertainty due to its stochastic nature [37]. For example, the CI of components can fluctuate due to changes in renewable energy generation. In addition, execution time and energy consumption can be significantly affected by resource interference from co-located workloads and network variability. Such uncertainty can affect the CF of components, as it is directly related to both CI and execution time.

1) Carbon Intensity Fluctuation: In the real world, the generation of renewable energy (e.g., solar, wind, etc.) in power grids can fluctuate substantially due to climate changes

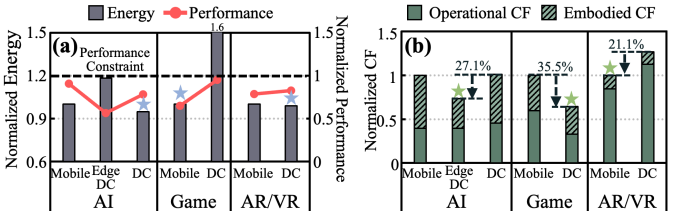

Fig. 3. Carbon optimization design space across edge-cloud infrastructure. (a) shows the normalized energy and performance of execution targets available for various workloads and (b) shows their normalized CF. Note the energy and CF are normalized to those of Mobile in each workload, while the performance is normalized to the performance constraint.

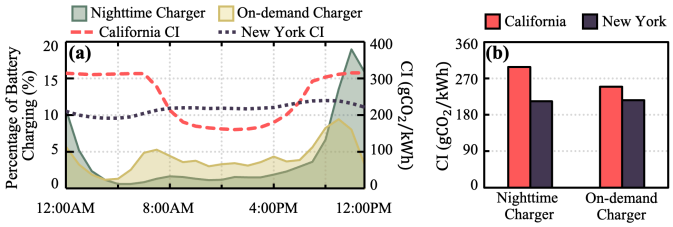

Fig. 4. (a) Battery charging patterns of nighttime charger (green area) and on-demand charger (yellow area) along with the hourly CI of California and New York, and (b) the overall CI of the two battery charging scenarios in different locations.

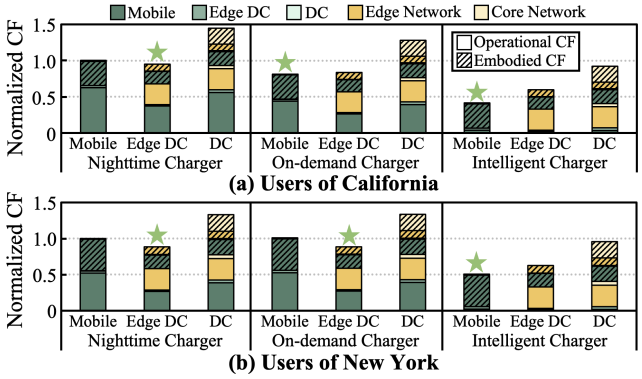

Fig. 5. CF of execution targets running SqueezeNet for users with different charging scenarios in (a) California and (b) New York. CF is normalized to that of Mobile in Nighttime Charger.

(e.g., clouds, turbulence, solar radiation, etc.) [7], [8], [21]. Furthermore, the generation of other energy sources can fluctuate during the generation process [47]. As the CI of a power grid is determined by the amount of energy generated by each energy source, the CI can also fluctuate frequently and unexpectedly, along with the varying energy generations. Given the linear relationship between the CI and the amount of operational CF, uncertainty in CI can affect the operational emissions of each component.

2) Runtime Variance: In a realistic execution environment, there can be several co-located workloads not only on DCs [48], but also on the user devices [49]. The resource interference from the co-located workloads can affect the efficiency of computing components substantially [34], [50]. Network variability stemming from varying wireless signal strength in edge network and traffic in the core network can also affect the communication efficiency [51], [52]. Thus, it is also crucial to consider the stochastic runtime variance for edge computing.

IV. Carbon Design Space for Edge Computing

In this section, we explore the design space of carbon-aware scheduling for edge computing. The design space includes the scheduling decisions (i.e., allocations of a user's requests across edge-cloud infrastructure) in various environments. We explore three state-of-the-art edge workloads: artificial intelligence (AI), game, and augmented/virtual reality (AR/VR). Details of the workloads and edge-cloud execution setup are explained in Section VI.

A. Carbon Design Space Overview

Fig. 3 shows (a) latency, energy consumption, and (b) carbon footprint (CF) results of edge applications. The x-axis shows the available execution targets for each workload.

As shown in Fig. 3, the energy-optimal execution target (blue star) is different from carbon-optimal one (green star) — the CF gap between the carbon-optimal target and energy-optimal one is 27.1%, 35.5%, and 21.1%, for AI, Game, and AR/VR workloads, respectively. There are two reasons: 1) the operational CF (solid area in Fig. 3(b)) is different from the energy consumption (Fig. 3(a)) due to the heterogeneous carbon intensity (CI) and 2) the amortization of embodied CF (shaded area in Fig. 3(b)) contributes to the total CF.

In Fig. 3, the carbon-optimal execution target also varies with the computation-communication ratio of the workloads. For example, in case of AR/VR workload, the carbon optimal execution target is Mobile due to the smaller computation-to-communication ratio. On the other hand, in case of AI and Game workloads, the CF of DCs is the lowest, due to the larger computation-to-communication ratio which can benefit from higher computing capabilities of DCs.

Takeaway 1: Carbon optimization is distinct from energy optimization. The carbon-optimal target depends on the workload characteristics.

B. Heterogeneous Carbon Intensity

In this subsection, we further explore the impact of heterogeneous CI on the operational carbon emissions of each component. To explore the realistic CI of the components, we employ the hourly energy generation data of all the US power grids [53], [54] along with the statistical energy charging/powering models of the components.

Carbon intensity of mobile: In case of the user devices, there can be two different battery charging scenarios [55]: 1) nighttime charger who charges the battery at night and 2) on-demand charger who charges the battery on demand throughout a day. To consider the impact of the battery charging scenarios, we employ empirical models of users' battery charging behaviors [56], [57] (i.e., hourly battery charge rate trace over the week). Fig. 4(a) shows the two battery charging

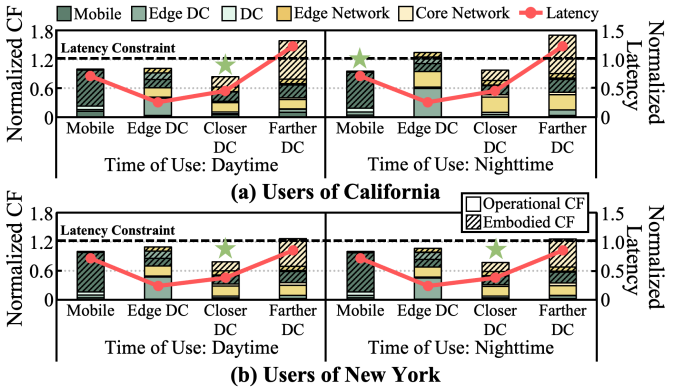

Fig. 6. CF and latency of execution targets running ResNet for users with different time in (a) California and (b) New York. CF is normalized to that of Mobile in each time of use.

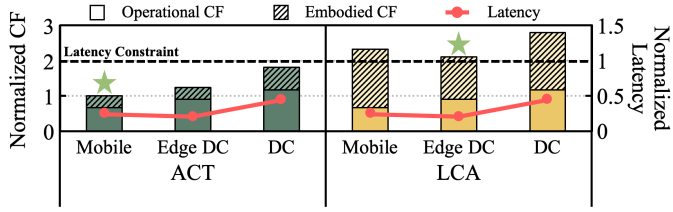

Fig. 7. CF and latency of execution targets running MobileNet with two different embodied CF modeling tools. CF is normalized to that of Mobile with ACT.

scenarios along with the hourly CI of California and New York. As shown in Fig. 4(b), the CI of the two scenarios can vary depending on the location of the users. In addition to the two scenarios, we also consider intelligent chargers who charge the battery when the renewable energy is available at the close grid to explore their carbon impact.

Fig. 5 shows the CF of an AI workload (SqueezeNet) with different charging scenarios for (a) mobile users in California and (b) those in New York. The x-axis shows the three execution targets with different battery charging scenarios.

Typically, in case of the Nighttime Charger, the CI of Mobile is usually high since the battery is charged during the night when less amount of renewable (i.e., solar energy) is available. When the charging behavior changes from nighttime charger to intelligent charger, the CI of Mobile significantly decreases, saving the overall CF by up to 61.2%. In this case, the carbon-optimal target shifts to Mobile.

As compared between Fig. 5(a) and (b), even when the users in California and New York have the same charging behavior, the carbon-optimal target differs — this is because the amount of renewable energy generation depends on the location.

Takeaway 2: The carbon-optimal target varies with the charging behavior of users and their location. Intelligent carbon-aware battery charging for client devices reduces CF by up to 61.2%.

Carbon intensity of DCs: In case of DCs, the CI can vary depending on the location and time. To consider this impact, we use the location information of cloud services' DCs [58], [59], and different time of use (daytime and nighttime). Fig. 6 shows the CF and latency of ResNet with different time of use for users in different locations.

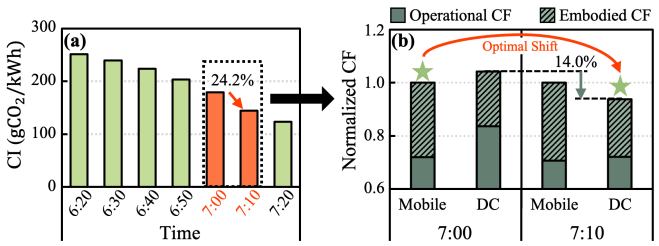

Fig. 8. Impact of CI fluctuations on the scheduling decisions. (a) shows the CI of the CAISO and (b) shows the CF of execution targets running 1st-Person Shooting Game at different time. CF is normalized to that of Mobile at 7:00.

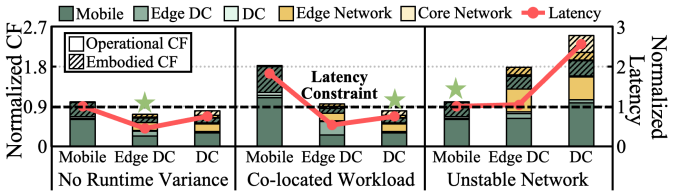

Fig. 9. CF and latency of execution targets running Inception with runtime variance. CF is normalized to that of Mobile with no variance.

As shown in Fig. 6, CF of the execution targets varies depending on the time when users use the application — the carbon-optimal target shifts from closer DC in daytime to Mobile in nighttime for users in California. As compared between Fig.6(a) and (b), the carbon-optimal target also depends on the location of users.

Given the time- and location-dependent CI, it is also possible to consider DCs in different locations — e.g., a farther DC with plenty of renewable energy [60]–[62] — as the execution target. In Fig.6, although farther DCs can have lower CI, offloading to the farther DCs often violates the latency constraint due to the increased network latency.

Takeaway 3: Scheduling across the components needs to carefully consider the geographical trade-off between CI and latency.

C. Embodied Carbon Emission

In this subsection, we explore the impact of embodied CF on the carbon-aware scheduling, by using two embodied CF modeling tools, i.e., ACT [13] and LCA [11][1]. Fig. 7 shows the latency and CF (with two different modeling tools) of execution targets for MobileNet. In Fig. 7, ACT indicates Mobile as the carbon-optimal target whereas LCA indicates Edge DC as the optimal one. This discrepancy comes from different patterns of latency (which linearly affect the discounted embodied CF) and operational energy of the targets. Given the higher estimation of embodied CF of LCA (compared to ACT), this result implies rising embodied CF should be further considered when designing green systems.

[1]Among the various embodied CF modeling tools, we use ACT and LCA since they estimate lower and upper bound embodied CF, respectively.

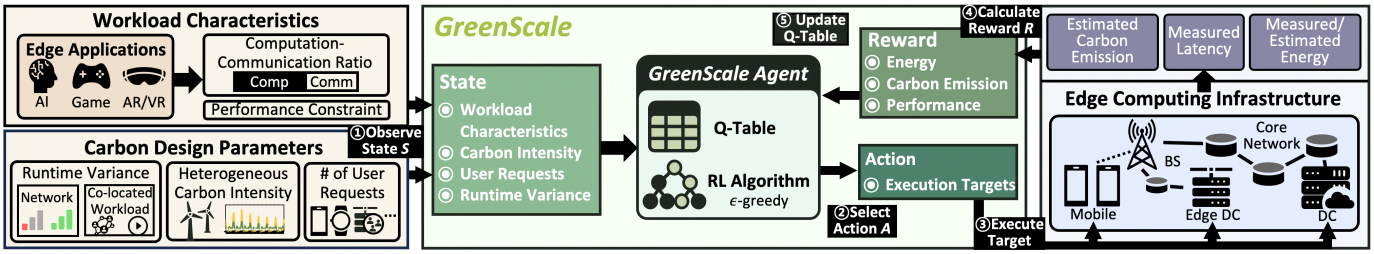

Fig. 10. GreenScale design overview.

Takeaway 4: Giving rising embodied CF of the components, amortization of embodied emission should be carefully considered when designing green systems.

D. Inherent Uncertainty

In this subsection, we explore the impact of inherent uncertainty on the scheduling decisions based on the measurements.

1) Carbon Intensity Fluctuation: To explore the impact of CI fluctuations, we collect the CI of grids at minute level intervals [63]. Fig. 8 shows (a) the CI of the grid in California (i.e., CAISO), and (b) CF results for 1st-Person Shooting Game at different time. In Fig. 8(a), the CI decreases by up to 24.2% within 10 minutes, due to a sudden increase of renewable energy generation. This decreases the CF of DC by 14.0%, shifting the carbon-optimal execution target for the Game application, as shown in Fig. 8(b).

2) Runtime Variance: For the interference from co-located workloads, we measure the computation latency on each component while running other workloads simultaneously [34], [48]. We also measure the network latency under an unstable network [51], [64], [65]. Fig. 9 shows the CF and latency of Inception when there is no runtime variance, when there exist co-located workloads, and when the network is unstable.

When there is no runtime variance, the carbon-optimal execution target is Edge DC. However, when there exist co-located workloads, the carbon-optimal target shifts to DC. This is because the adverse impact of co-located workloads depends on the computation and memory capabilities of each component. On the other hand, when either the edge network or core network is unstable, the carbon-optimal target shifts to Mobile due to the increased network overhead.

Takeaway 5: Inherent uncertainty, common for edge computing, can impact the total CF of systems meaningfully, shifting the carbon-optimal execution target.

V. GreenScale

In this section, we propose a carbon-aware scheduler, *GreenScale*, for edge computing. The objective of the scheduler can be formulated as in (4)-(6), where x is the execution target, $\mathcal{X}$ is the set of available execution targets (i.e., mobile, edge DC, hyperscale DC), and τ is the performance constraint, choosing the carbon-optimal execution target for edge applications in different environments while satisfying the performance constraint.

$$x^* = \arg\min_{x \in \mathcal{X}} \mathrm{CF}_x \quad \text{subject to} \quad T_x \leq \tau \tag{4}$$

$$\mathrm{CF}_x = \sum_{i \in \{\mathrm{comp,comm,idle}\}} \frac{\mathrm{CF}_i}{U_i}, \quad x \in \mathcal{X} \tag{5}$$

$$\mathrm{CF}_i = \mathrm{OP}_{\mathrm{CF}} + \mathrm{E}_{\mathrm{CF}} \times \frac{T}{LT} \times \frac{N}{N_{\mathrm{total}}} \tag{6}$$

To achieve the objective (minimizing not only the CF of the local device but also the global CF by considering the number of allocated HWs), we use a light-weight reinforcement learning (RL) algorithm for *GreenScale*. RL has the following advantages [66], compared to heuristic-based [60], [67], [68] and supervised learning-based [69], [70] techniques for scheduling: **(1) Adaptability -** as discussed in Section III and IV, carbon-aware scheduler needs to adapt to varying features (e.g., user- and environment-dependent features in carbon-aware design space). RL has an advantage for learning and acting to the varying features, **(2) Online learning -** unlike an offline learning-based techniques, RL continuously adapt its decision-making policy online, using system-level feedback and specialize to the current workload, system configuration, and user- and environment-dependent features, and **(3) Extensibility -** unlike heuristic-based techniques, RL can be easily extended to different types of workloads and devices with online learning.

A. Design Overview

In this subsection, we explain the design overview of *GreenScale*. Fig. 10 shows the overview of *GreenScale*. For each execution of the computations for an edge application, *GreenScale* observes the current execution state (①), including workload characteristics, carbon intensity (CI), user requests, and runtime variance. For the observed state, *GreenScale* selects an action (i.e., execution target for the computations) (②) which is expected to minimize both operational *and* embodied carbon emissions meeting the performance requirements. This selection is based on a lookup table (i.e., Q-table) that contains the accumulated rewards of previous selections. *GreenScale* then executes the computations on the selected target (③), while observing the results (i.e., latency, energy, and carbon emissions). Based on the results, *GreenScale* calculates the reward (④), which indicates how much the target improves carbon efficiency and meets the performance requirements. Finally, *GreenScale* updates the Q-table with the calculated reward (⑤).

TABLE I
STATE-RELATED FEATURES

State		Description	Discrete values
Workload-related feature	$S_{Workload}$	Computation-to-communication ratio (i.e., FLOPs / transmission data bytes)	Small (<10K), Medium (<50K), Large (<100K), Larger (>=100K)
Carbon intensity	S_{CI_Mobile}	CI of mobile device	Small (<50), Medium (<200), Large (<400), Larger (>=400)
	$S_{CI\_Edge\_DC}$	CI of edge DC	Small (<50), Medium (<200), Large (<400), Larger (>=400)
	$S_{CI\_Edge\_Network}$	CI of edge network	Small (<50), Medium (<200), Large (<400), Larger (>=400)
	$S_{CI\_Core\_Network}$	CI of core network	Small (<50), Medium (<200), Large (<400), Larger (>=400)
	S_{CI_DC}	CI of hyperscale DC	Small (<50), Medium (<200), Large (<400), Larger (>=400)
Embodied emission	$S_{U\_Edge\_DC}$	Number of user requests in edge DC	Small (<32), Medium (<256), Large (>=256)
	S_{U_DC}	Number of user requests in DC	Small (<64), Medium (<256), Large (<1024) Larger (>=1024)
Runtime variance	$S_{Co\_CPU\_M}$	CPU utilization of co-running apps on mobile	None (0%), Small (<25%), Medium (<75%), Large (100%)
	$S_{Co\_MEM\_M}$	Memory usage of co-running apps on mobile	None (0%), Small (<25%), Medium (<75%), Large (100%)
	$S_{Co\_CPU\_E}$	CPU utilization of co-running apps on edge DC	None (0%), Small (<25%), Medium (<75%), Large (100%)
	$S_{Co\_MEM\_E}$	Memory usage of co-running apps on edge DC	None (0%), Small (<25%), Medium (<75%), Large (100%)
	$S_{Co\_CPU\_DC}$	CPU utilization of co-running apps on DC	None (0%), Small (<25%), Medium (<75%), Large (100%)
	$S_{Co\_MEM\_DC}$	Memory usage of co-running apps on DC	None (0%), Small (<25%), Medium (<75%), Large (100%)
	$S_{Edge\_Network}$	Bandwidth of edge network	Regular (>40Mbps), bad (<=40Mbps)
	$S_{Core\_Network}$	Bandwidth of core network	Regular (>40Mbps), bad (<=40Mbps)

B. GreenScale RL Design

In this subsection, we explain the design of *GreenScale* RL algorithm. Given the resource-constrained edge execution environment, it is crucial to deploy the RL algorithm with low runtime overhead (for both training and inference). Among the various RL algorithms [66], Q-learning has an advantage for low runtime overhead, as it finds the best action with a lookup table. Hence, we use Q-learning for *GreenScale*.

For accurate prediction with the Q-learning, it is crucial to model the carbon design space into the state-action-reward space. We tailor-design the *state* and *action* based on our observations in the carbon-aware design space (Section IV). In addition, we define the *reward* with the measured latency, energy, and operational and embodied carbon emissions.

State: We define states that are critical to carbon-aware scheduling. Table I summarizes the states.

As we explored in Section IV-A, the carbon-optimal target depends on workload characteristics (i.e., computation-to-communication ratio). Hence, we define $S_{Workload}$ with the number of floating point operations per execution and the size of transmission data per execution.

In addition, as we explored in Section IV-B, the carbon-optimal target significantly depends on the CI of the involved infrastructure components. For the mobile device, the CI is dependent on the battery charging pattern[2] and the location of the user. Thus, we define S_{CI_Mobile} as the function of the two parameters. For the rest of the components (i.e., edge DC and DC, and edge and core network), the CI is dependent on the location and the time of use. Thus, we define $S_{CI\_Edge\_DC}$, S_{CI_DC}, $S_{CI\_Edge\_Network}$, and $S_{CI\_Core\_Network}$, as the functions of the location and time. Note it is possible to obtain the CI information by using carbon APIs (e.g., WattTime [53], electricityMap [71]) for the power grids, and/or carbon APIs of the cloud services [72], [73].

As we explored in Section IV-C, the carbon-optimal target can also depend on the amortization of embodied emission. Assuming that the number of server resources allocated to an application (which affects the embodied CF) is adjusted by the provisioning techniques (e.g., AWS Provisioning and Orchestration, Azure AD Provisioning, etc.) depending on the number of user requests and costs, we define $S_{U\_Edge\_DC}$ and S_{U_DC} to represent the number of user requests in the edge and hyperscale DC, respectively — this potentially affects the allocations of user requests, changing the number of server resources and avoiding rebound effect on DCs [21].

As we explored in Section IV-D, the efficiency of the targets highly depends on the existence of co-running workloads. Since the intensity of resource interference from the co-running workloads varies with their resource utilization[3], we identify S_{Co_CPU} and S_{Co_MEM}, which represent the CPU and memory utilization of co-running workloads, respectively, for each target. We also model the network instability with $S_{Edge\_Network}$ and $S_{Core\_Network}$ to represent the network bandwidth of edge and core networks, respectively.

To convert the continuous features into discrete values for constructing the Q-table, we apply a clustering algorithm to each feature [34], [76]; it determines the optimal number of clusters for the given data. The last column of Table I summarizes discrete values for each state — note these values can be adjusted by the clustering algorithm depending on the specifications of infrastructure components.

Action: We define the actions as the available execution targets (i.e., mobile device or servers in different locations) in the edge-cloud infrastructure. The set of actions can be augmented by considering other control knobs, such as dynamic voltage and frequency scaling and batch executions. For example, as long as the performance constraint is satisfied, it is possible to reduce the frequency of processors reducing the operational emissions. It is also possible to improve the throughput of an execution target with batching.

Reward: To model the optimization objective with the reward, we encode three rewards: R_{perf}, R_{energy}, and R_{CF}. R_{perf} is the measured performance for a selected action. R_{energy} and R_{CF} are the estimated energy consumption of the selected action and its carbon emissions, respectively — note, since

[2]In off-the-shelf smartphones, it is possible to track the battery charging behavior of the user, by using the battery monitoring APIs (e.g., BatteryManager API in Android OS and BatteryState API in iOS).

[3]It is possible to monitor the resource utilization of the virtualized server instances using resource monitoring tools (e.g., stats of docker [74], CloudWatch of AWS [75], etc.). If the resource monitoring is not available, *GreenScale* can estimate the intensity of resource interference based on a ratio of the currently measure latency to the average latency.

TABLE II
NN INFERENCE WORKLOADS

Category	NN	FLOPs	Params	Input/ Output (KB)	Perf. Req. (ms)
Vision	MobileNet	0.3G	3.5M	150.5	33.3
	SqueezeNet	0.8G	1.2M	150.5	
	ResNet	4.1G	25.6M	150.5	
	MobileNet-SSD	0.8G	6.8M	270	
	Inception	5.7G	23.8M	268.2	
Text	BERT	25.3G	17.5M	1	100

the energy estimation is based on the measured latency, the mean absolute percentage error of the energy estimation is 7.3%. To ensure *GreenScale* selects an execution target that minimizes global carbon emission (and local energy consumption) while satisfying the performance constraint, the reward *R* is calculated as in (7), where α and β are the weights of performance and energy[4], respectively.

$$\begin{aligned} &if \quad R_{perf} < \ Performance \ Constraint, \\ &\qquad R = -R_{CF} + \alpha R_{perf} - \beta R_{energy} \\ &else \\ &\qquad R = -R_{CF} - \beta R_{energy} \end{aligned} \tag{7}$$

The reward is calculated depending on whether the performance constraint is satisfied or not. In (7), R_{energy} and R_{CF} are multiplied by -1 to increase the reward for lower values.

C. Algorithm Details

In this subsection, we provide a detailed explanation of *GreenScale's* Q-learning algorithm. To avoid local optima, *GreenScale* deals with the exploitation versus exploration dilemma, by employing the epsilon-greedy algorithm — the epsilon-greedy algorithm chooses the action with the highest reward or a uniformly random action based on an exploration probability.

In Q-learning, the value function *Q(S,A)* takes state *S* and action *A* as parameters. At the beginning, the Q-learning algorithm initializes the *Q(S,A)* in the Q-table with random values. At runtime, the algorithm observes *S* for each workload execution by checking the workload characteristics, carbon intensity, user requests, and runtime variance. For the observed *S*, it generates a random value and compare it with ϵ (i.e., 0.1 by referring to previous works in this domain [34], [76]). If the random value is smaller than ϵ, the algorithm randomly chooses *A*. Otherwise, it chooses *A* with the largest *Q(S,A)*.

The algorithm then executes the workloads on the target defined by *A*. During the execution, the algorithm measures R_{perf} and estimates R_{energy} and R_{CF}. Based on these values, the algorithm calculates the reward *R*, as in (7). It then observes new state *S'* and chooses *A'* which has the largest *Q(S',A')*. The algorithm updates the *Q(S,A)* as in (8), where γ and μ are hyperparameters representing the learning rate and the discount factor, respectively — we set γ and μ as 0.9 and 0.1, respectively, based on a sensitivity test.

$$Q(S,A) = Q(S,A) + \gamma[R + \mu Q(S',A') - Q(S,A)] \tag{8}$$

After the learning is completed (i.e., the largest *Q(S,A)* value for each state *S* is converged), the Q-table is used to select *A* which maximizes *Q(S,A)* for the observed *S*.

D. Transfer Learning

The training of *GreenScale* needs to be performed on-device to learn the user- and device-dependent features. Given the resource-constrained edge execution environment, it is important to reduce the training overhead. To expedite the on-device training minimizing its overhead, developers can pre-train the Q-table, and transfer the pre-trained model to the devices [77] (results are presented in Section VII-B).

VI. EVALUATION METHODOLOGY

A. Workloads

To evaluate *GreenScale*, we use three state-of-the-art workloads: artificial intelligence (AI), game, and augmented/virtual reality (AR/VR) summarized in Table II, III, and IV. We explore AI workloads as they are widely used in many of recent intelligent services [78]. In addition, we explore game applications as they dominate 63.5% of the application market [79]. We also explore AR/VR applications, as they have gained recent traction in both consumer and research communities [80] thanks to the advances in efficient computing technologies, high-speed networks, and specialized hardware.

For the AI workloads, we run the inference of neural networks (NNs) which are widely used in mobile intelligent services [81]. The TFLite runtime [82] implemented in an Android application is used to run inference on mobile. For DCs, TensorFlow runtime [83] is used for inference execution. Here, the migration overhead is measured with the transmission latency and energy for the input/output.

For the game workloads, we run three types of games which are widely used by mobile users [84]. For mobile, we run the Android game applications to measure the latency, FPS, and energy. For DC, we consider the cloud gaming service scenario [84], [85] where the backend computations run in the cloud based on frontend user interactions gathered on the smartphone display. We run NVIDIA Geforce Now [85] on mobile for measuring the performance and energy of the frontend including the transmission overhead. We also measure the energy consumption of the backend by running the desktop games on the server.

For the AR/VR workloads, we use four workloads from ILLIXR [86] and XRBench [87]. AR/VR workloads have three sub tasks: 1) Input which reads inputs from sensors, 2) Perception which understands the current surrounding environment, 3) Visual/Audio which combines the virtual information with the physical world, and renders the final frame. We use the OpenXR-based ILLIXR framework [86] to run the per-frame

[4]We set 0.1 for α and β based on the sensitivity analysis — this exhibits the best CF while satisfying the performance constraint. In general, the increase of the α generally reduces the performance of the workloads (by using the maximum frequency) at the expense of the energy and CF. On the other hand, the increase of β generally improves the energy and CF with DVFS but hurts QoS.

TABLE III
GAME WORKLOADS

Category	Name	Input/ Output (MB)	FPS Req.	Latency Req. (ms)
1st-Person Shooting Game	Fortnite	3.2	60	100
3rd-Person Role Playing Game	Genshin Impact	3.0	60	500
Omnipresent Strategy	Team Fight Tactics	1.9	60	1,000

TABLE IV
AR/VR WORKLOADS

Category	Name	Tasks	Input/ Output (KB)	Perf. Req. (ms)
VR	3D World	1) Input 2) Perception 3) Visual/ Audio	540.5	33.3
	3D Material			
	3D Cartoon			
AR	AR Demo			

TABLE V
EDGE-CLOUD INFRASTRUCTURE

Component			Measurement -based	Model -based
Edge Computing Infrastructure	Mobile Devices	Latency	✓	
		Energy	✓	
	Edge Network	Latency	✓	
		Energy		✓ [96]
	Edge DC	Latency	✓	
		Energy	✓	✓ [94]
	Core Network	Latency	✓	
		Energy		✓ [97]
	DC	Latency	✓	
		Energy	✓	✓ [94]
Carbon Intensity	Power Grid		✓	
	Mobile Battery			✓ [57]
	Uncertainty			✓ [98], [99]
Embodied CF	LCA			✓ [44]
	Architectural Model			✓ [13]
Runtime Variance	On-device Interference	Mobile	✓	
		Server	✓	
	Network Variability	Edge Network	✓	✓ [51]
		Core Network	✓	✓ [65]

tasks on either mobile devices or servers. Here, the migration overhead includes the transmission latency and energy for input/output.

To ensure that different targets run the same input for the interactive workloads (i.e., game and AR/VR workloads), we pre-encoded a series of inputs [80], [88].

We determine the latency target of each workload by referring to the user study and QoS analysis works for each workload. For the AI workloads, we use 30 FPS and 100 ms for the vision NNs and text NN, respectively, by referring to [89]. For the game workloads, we use two performance requirements: FPS for the smoothness and end-to-end latency for the responsiveness. We use different requirement values for different classes of games by referring to a prior study on game performance [90] (Table III). For the AR/VR workloads, we use the 30FPS of end-to-end latency requirement by referring to ILLIXR [86] and XRBench [87].

B. Edge-Cloud Infrastructure

We base our experiments on the edge-computing infrastructure: mobile, edge DC, DC, edge network, and core network. Table V summarizes how we build our study.

We use an off-the-shelf Android smartphone for the AI and game workloads, Pixel 3 [11], which is equipped with a Snapdragon 845 SoC [91], 4GB RAM, and 128GB storage. For the AR/VR workloads, we use an NVIDIA Jetson device equipped with NVIDIA Volta GPU, 64GB RAM, and 32GB storage — the workloads do not run on Android devices. We measure the latency of the workloads directly on the devices, while measuring their power consumption using an external power measurement device [92].

For edge DC and hyperscale DC, we use the AWS instances of p3.2xlarge (equipped with Xeon E5-2686 v4, NVIDIA Tesla V100, 64GB RAM, and 4TB SSD) and p4d.24xlarge (equipped with Xeon P-8275CL, 8 NVIDIA A100 GPUs, 1152GB RAM, and 8TB SSD), respectively, which have similar specifications to the ones used in [93]. We also measure the latency of the workloads directly on the instances, while estimating the power consumption based on a frequency- and utilization-based power model[5]. Since the game workloads do not run on Linux, we use a Windows server (equipped with AMD Ryzen 7 5800X, Geforce RTX 3080 Ti, 64GB RAM, and 1TB SSD) having similar specifications to the ones used in NVIDIA Geforce Now [85]. Note we assume the average PUE of 1.6 and 1.1 for edge DCs [95] and DCs [2], respectively.

For BSs of edge networks, we use the macro-scale BS whose TDP is 1000W [96]. We calculate the single-user power consumption based on the number of users who can be served by the macro-scale BS [96]. We measure the signal strength-dependent wireless network latency on the mobile device [34], while adjusting the signal strength by its locations [51]. For the core network, we use the TDP and bandwidth of Cisco routers [97]. We also measure the distance-dependent propagation latency along with the number of associated core routers by connecting the mobile devices with the instances in different locations in the US.

The measured latency and power consumption of the devices are used for calculating their operational emissions. In addition, the specifications of the devices are used to calculate the embodied emissions with LCA and ACT [13], [44].

C. GreenScale Evaluation

To evaluate the effectiveness of *GreenScale*, we compare it with three baselines on our edge-cloud infrastructure with an average number of mobile app users [100]: Mobile, which always runs the workloads on the mobile device; DC (Best), which runs workloads on the most carbon efficient DC; and Opt, an oracular design that always runs the workloads on the carbon-optimal target. We also compare *GreenScale* with a state-of-the-art energy-aware edge-cloud scheduler [34].

To validate the adaptability of *GreenScale* to heterogeneous carbon intensity, we use the hourly energy generation data of

[5]We construct the power model by measuring the power consumption of local servers which have the similar specifications with the virtualized instances — similar practice has been used in previous works [94].

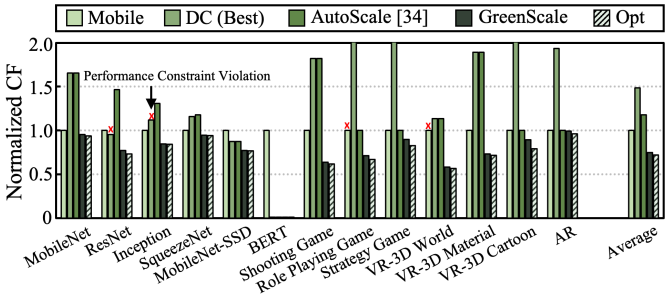

Fig. 11. CF result of *GreenScale* compared to the baselines. CF is normalized to that of Mobile in each workload, and x above the bar indicates that the selected execution target does not satisfy the performance constraint.

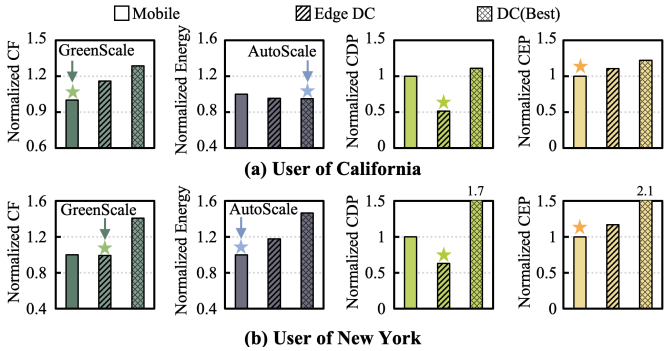

Fig. 12. CF, energy, CDP and CEP results of execution targets available for (a) a user in California and (b) a user in New York. Each metric is normalized to that of Mobile and the stars above the bars indicate the optimal execution target identified by each metric. GreenScale selectes the carbon optimal execution target whereas AutoScale [34] selects the energy optimal execution target.

all the US power grids [53], [54] along with the empirical energy charging/power models of the components [56], [57] (i.e., hourly battery charge rate trace over the week for the two scenarios). We train *GreenScale* with the 2020 hourly energy generation data and test it with the 2021 data.

We also emulate the realistic runtime variance. In case of resource interference from co-running workloads, we inject its impact (measured degradation depending on the resource utilization of co-running workloads) according to the mobile application (i.e., web browser and music player) usage [34] and typical server utilization patterns [48], [50]. In addition, since the wireless network signal strength is usually modeled with Gaussian distribution [51], we randomly inject the impact of signal strength variability following the Gaussian distribution — we use our measurements (i.e., measured edge network latency depending on signal strength and empirically collected daily signal strength variations) for mean and standard deviation of the distribution. On the other hand, the wired network variability is inherently included in the measured distance-dependent latency [101].

For evaluating the effectiveness of transfer learning, we pre-train the Q-table for each workload by using the rest of the workloads, assuming that the developer can pre-train the Q-table with existing workloads and deploy it [34], [76].

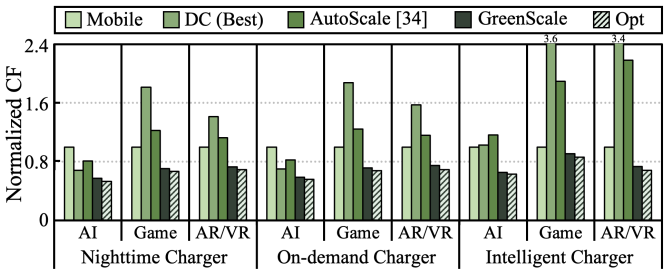

Fig. 13. CF result of *GreenScale* compared to the baselines with different batter charging models. CF is normalized to that of Mobile in each workload.

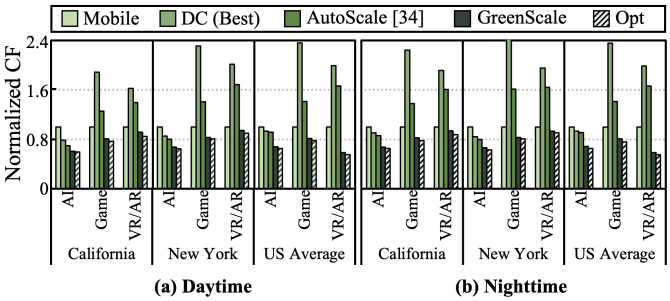

Fig. 14. CF result of *GreenScale* compared to the baselines with users from different location at (a) daytime and (b) nighttime. CF is normalized to that of Mobile in each workload.

VII. Evaluation Results

Fig. 11 shows the average carbon footprint (CF) normalized to Edge when there is no runtime variance — x above the bars indicates the performance constraint violation. Overall, *GreenScale* improves the average CF by 25.8% and 46.9%, compared to Mobile and DC (Best), respectively, satisfying the performance constraint. Across diverse workloads, *GreenScale* accurately predicts the carbon-optimal execution target. As a result, *GreenScale* achieves almost the same CF as Opt; the difference is only 2.8%. *GreenScale* also improves the average CF over the state-of-the-art scheduler, AutoScale [34], by 35.2%. This is because carbon optimization is distinct from conventional energy optimization, due to the time- and location-dependent carbon intensity and the amortization of embodied CF.

Fig. 12 shows the CF, energy, CDP (Carbon Delay Product), and CEP (Carbon Energy Product) of various execution targets available for (a) a user in California and (b) a user in New York running an AI Workload. As shows in Fig. 12, GreenScale selects the most carbon efficient execution targets, whereas AutoScale selects the most energy efficient (yet carbon inefficient) execution target. As a result, GreenScale significantly improves the CF of the execution, compared to AutoScale. In Fig. 12, CDP does not seem to be a good indicator for the carbon optimization, as it gives weight to the latency — as long as the latency constraint is satisfied, the actual latency value is not much critical though. On the other hand, CEP can be used to balance between the overall CF and the energy consumption of the local device.

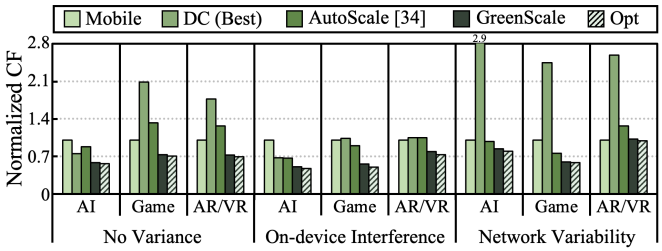

Fig. 15. CF result of *GreenScale* compared to the baselines with runtime variance. CF is normalized to that of Mobile in each workload.

TABLE VI
ABLATION STUDY RESULTS

GreenScale Features	Accuracy
W/o Runtime Variance Features	59.9%
W/o Embodied Emission Features	62.1%
W/o Carbon Intensity Features	68.2%
W/o Workload Features	79.2%
All Features (Ours)	**96.8%**

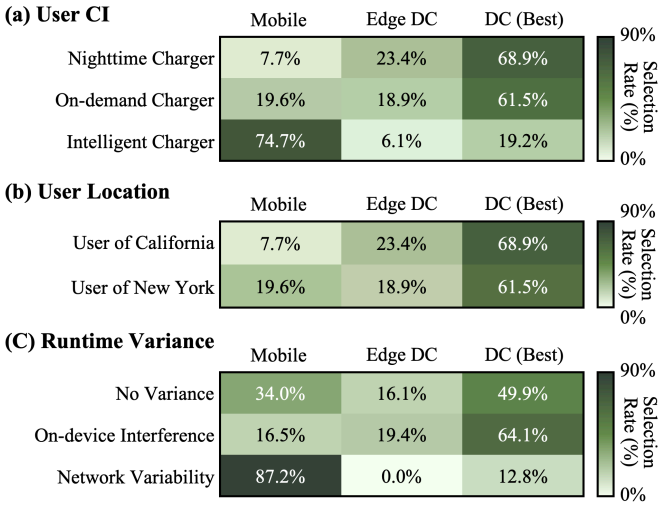

Fig. 16. Execution target selections of *GreenScale* with (a) different battery charging models, (b) users from different locations, and (c) runtime variance. Note that the color represents the selection rate.

A. Adaptability Analysis

In this subsection, we demonstrate the *GreenScale's* adaptability to various execution environments.

Adaptability to Mobile Heterogeneity: Fig. 13 shows the average CF (y-axis) normalized to Mobile for different battery charging scenarios (x-axis). As *GreenScale* explicitly considers the heterogeneous carbon intensity (CI), it finds the carbon-optimal execution target for all types of scenarios. As a result, it improves their average CF by 33.4%, 49.5%, and 41.7%, compared to Mobile, DC (Best) and AutoScale [34], respectively — it shows a 5.6% difference with Opt.

Adaptability to Location and Time of Use: Fig. 14 shows the average CF (y-axis) normalized to Mobile for the users in different states (x-axis) in (a) daytime and (b) nighttime. Since *GreenScale* can adapt to the location- and time-dependent CI, it accurately selects the carbon-optimal target regardless of users' location and time of use, improving the average CF by 23.3%, 48.4%, and 36.6%, compared to Mobile, DC (Best), and AutoScale [34], respectively.

Adaptability to runtime variance: Fig. 15 shows the average CF normalized to Mobile in the presence of runtime variance (which happens more frequently than CI changes). Since *GreenScale* can adapt to the runtime variance along with the heterogeneous CI and the embodied CF, it improves the average CF by 29.4% and 49.8%, compared to Mobile and DC (Best), respectively, meeting the performance constraint. *GreenScale* also shows 29.1% better CF than AutoScale [34], on average — this result implies AutoScale, which optimizes the energy efficiency adapting to runtime variance, is not sufficient for carbon optimization.

B. Accuracy and Overhead Analysis

In this subsection, we provide the accuracy and overhead analysis results.

Accuracy: To analyze the prediction accuracy of *GreenScale*, we compare its decision to the optimal one. *GreenScale* achieves 97.1% and 96.8% accuracy even in the presence of varying carbon intensity and runtime variance, respectively.

Fig. 16 shows the detailed execution target selections of GreenScale depending on the execution time. GreenScale accurately selects the most carbon-efficient execution target, effectively adapting to the execution environment — the selection difference between GreenScale and Opt is less than 0.2%. For example, GreenScale successfully adapts to the charging behaviors of users, as shows in Fig. 16(a) — it mostly selects DCs for nighttime chargers, while dominantly selecting the mobile device for intelligent chargers. GreenScale also adapts to the user locations, as shown in Fig. 16(b) — the CI of the execution targets can significantly vary across locations. GreenScale's selections also vary depending on the runtime variance as shown in Fig. 16(c) — it mostly selects DCs when there is on-device interference, whereas dominantly selecting the mobile device when there is network variability. Note, even when GreenScale selects sub-optimal target, it does not degrade the overall CF compared to Opt. This is because the CF of the selected sub-optimal target is still similar to that of the optimal one — the CF difference is only less than 1%.

To further analyze the impact of each *GreenScale* feature on its accuracy, we conduct the ablation study. Table VI shows the ablation study results. As shown in Table VI, excluding each feature significantly degrades the *GreenScale* accuracy, demonstrating the importance of the features in the optimization of carbon-aware scheduling.

Training overhead: When training the Q-table, the reward converges after 100-110 workload runs, on average (Fig. 17). Before convergence, *GreenScale* exhibits 21.7% lower average carbon emission than Opt — it still shows 19.8% lower carbon emission compared to Edge, on average. The training overhead can be alleviated with the transfer learning (Section V-D) — the reward converges more rapidly reducing the average training time overhead by 40.3%.

Runtime overhead: To validate the viability of the mobile deployment of *GreenScale*, we implement it as a part of an

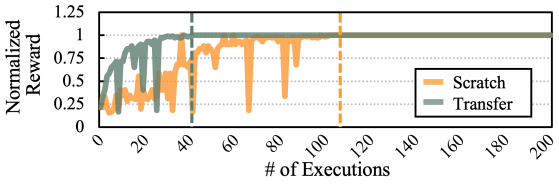

Fig. 17. Comparison of reward between Scratch (orange) and Transfer (green) over the number of executions. The dashed line marks the number of executions when the reward first reaches its maximum value.

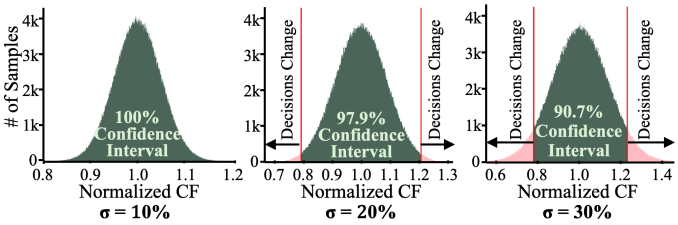

Fig. 18. Uncertainty analysis of *GreenScale* with different degrees of σ.

Android application. The runtime overhead of RL algorithm in *GreenScale* is, on average, 25.4μs for training, excluding the time for workload execution. It corresponds to the 1.5% of execution time per frame. In addition, after the training, the overhead of the RL inference execution is reduced to 7.3μs which accounts for 0.4% of the frame time. The memory requirement of *GreenScale* is 0.4 MB, which corresponds to 0.01% of the 3GB DRAM capacity of a typical mobile device.

C. Uncertainty Analysis

Although we follow the practice used in many prior works, inputs of the *GreenScale* have an inherent uncertainty [10]. The sources of uncertainty include the measurement noise [102], uncertainty in estimation models (for DC power [94] and embodied CF [13]), and environmental variance. To verify the effectiveness of the *GreenScale* even in the presence of such uncertainty, we conduct the uncertainty analysis with Monte Carlo approach [103], [104]. Given that the sources of uncertainty are modeled by the Gaussian distribution [102], we generate a sufficient number of random numbers (1,000,000) for each input with the Gaussian distribution of $N(\mu, \sigma^2)$, where μ and σ are the mean and the standard deviation. We use our measurements and estimations for μ, while varying σ from 10% to 30% of μ.

Figure 18 shows the uncertainty analysis results with different degrees of the uncertainty (i.e., σ). As shown in Figure 18, the execution target selected by *GreenScale* does not change when the σ is less than and equal to 10% of μ. This means that the conclusions in Section VII remain the same. Even with the 20% and 30% σ, the execution target does not change within the 97.9% and 90.7% confidence intervals, respectively. Note, even when the target is changed, *GreenScale* still reduces the CF by 28.6% compared to [34], on average — this is because even severe uncertainty does not change the overall CF trends.

VIII. Related Work

Carbon modeling tools: Given the increasing CF of ICT, academia and industry have proposed a number of tools to quantify carbon emissions across the hardware life cycles. LCA tools quantify the CF of components using coarse-grained information (e.g., economic cost of electronics and materials, etc.) [44], [45]. Although product environmental reports published by industry have been based on the LCA tools [11], [105], they have not been used for comparative analyses between hardware components to guide design space exploration. Complementing the above tools, architectural modeling tools were proposed to consider the direct CF from hardware manufacturing and operational use of systems [10], [12], [13]. ACT calculates the embodied CF based on a variety of parameters (e.g., SoC area, energy per area, raw materials, etc.) [13]. Given that such parameters have inherent uncertainty, FOCAL estimates the embodied CF based on its relative ratio of the operational CF [10]. By exploiting the above tools, this work quantifies the CF of edge computing, and first explores the optimization space of carbon-aware scheduling considering its unique features.

Optimization for edge computing: User-centric optimizations have primarily focused on minimizing the energy consumption of the device satisfying the user-specified performance constraint, by considering hardware heterogeneity on-device [31]–[33] and that across the edge-cloud computing infrastructure [34], [35]. However, as we demonstrate in Section IV, carbon optimization is distinct from energy optimization due to varying CI and embodied CF.

On the other hand, DC-centric optimizations have tried to maximize the operational efficiency of DC infrastructure by splitting the computations across edge-scale and hyperscale servers [40], [69], [70], meeting the service level agreement (SLA). Several techniques further consider the varying renewable energy availability at DCs. For instance, [60]–[62] tried to switch computations between DCs in different places considering their CI, whereas [19]–[21] temporally shift batch jobs considering the renewable energy availability within a DC. However, the above techniques have not considered the embodied CF and runtime variance in edge computing. [7], [22] first considered embodied CF when scheduling computations across DCs. However, they do not consider local devices as the scheduling decisions even when they are the most carbon-efficient execution target. In addition, they also do not consider the runtime variance in edge computing.

More recently, as the sustainability of computing has been receiving more attentions, several works have tried to further optimize the CF in edge computing. [36] tried to enable efficient data collecting and select the carbon-efficient execution target in edge computing by considering the location and time dependent CI. [106] tried to improve the accuracy of battery model using a reinforcement learning algorithm for better exploiting edge DCs. However, the recent techniques also do not consider heterogeneous charging behaviors of edge devices, embodied emissions, and runtime variance, which also has a significant impact on the overall CF of edge computing.

IX. Conclusion

Given billions of mobile users, it is crucial to design *green applications* which can improve the carbon-efficiency of edge computing. In this paper, we propose *GreenScale* which accurately selects the carbon-optimal execution target for edge computing with a lightweight reinforcement learning algorithm. Through the in-depth carbon characterization of state-of-the-art applications across edge-cloud infrastructure, we demonstrate that the carbon optimal scheduling decision depends on various features, such as *time* and *location*-dependent carbon intensity, the amortization of embodied carbon emissions, and runtime variance. In our evaluation, *GreenScale* improves the CF of the edge applications by 35.2% compared to a state-of-the-art edge-cloud scheduler. We believe *GreenScale* can be a viable solution to enable sustainable executions of the *green applications* at the edge.

X. Acknowledgments

This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korea government (MSIT) under RS-2023-0021271, the Institute of Information & communications Technology Planning & Evaluation (IITP) Grant funded by MSIT under RS-2024-00398353, the IITP-ITRC(Information Technology Research Center) Grant funded by MSIT under IITP-2025-RS-2023-00260091, the IITP-ICT Creative Consilience Program Grant funded by the MSIT under IITP-2025-RS-2020-II201819, and the National Science Foundation (NSF) under CCF-2324860. *(Corresponding author: Young Geun Kim)*

References

[1] L. Belkhir and A. Elmeligi, "Assessing ict global emissions footprint: Trends to 2040 & recommendations," *Journal of Cleaner Production*, vol. 177, pp. 448–463, 2018.

[2] Google, "Data centers - efficiency," https://google.com/about/datacenters/efficiency.

[3] Meta, "Software, servers, systems, sensors, and science: Facebook's recipe for hyperefficient data centers," https://tech.fb.com/engineering/2020/01/hyperefficient-data-centers/.

[4] AWS, "Sustainability - carbon-free energy," https://sustainability.aboutamazon.com/climate-solutions/carbon-free-energy?energyType=true.

[5] Meta, "Sustainability - energy," https://sustainability.fb.com/energy/.

[6] Google, "Data centers - 24/7 carbon-free energy by 2030," https://www.google.com/about/datacenters/cleanenergy/.

[7] B. Acun, B. Lee, K. Maeng, M. Chakkaravarthy, U. Gupta, D. Brooks, and C.-J. Wu, "A holistic approach for designing carbon aware datacenters," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[8] 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 *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[9] 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 *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 854–867.

[10] L. Eeckhout, "Focal: A first-order carbon model to assess processor sustainability," in *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024, pp. 401–415.

[11] Google, "Google pixel 3 product environmental report," https://sustainability.google/reports/pixel3-productenvironmentalreport/.

[12] D. Kline Jr, 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.

[13] 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 and architectural carbon modeling tool," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2022.

[14] G. S. Foundation, "Software carbon intensity specification," https://github.com/Green-Software-Foundation/sci/blob/main/Software_Carbon_Intensity/Software_Carbon_Intensity_Specification.md.

[15] S. Kannan and U. Kremer, "Towards application centric carbon emission management," in *2nd Workshop on Sustainable Computer Systems Design and Implementation (HotCarbon 2023)*, 2023.

[16] J. Switcher, G. Marcano, R. Kastner, and P. Pannuto, "Junkyard computing: Repurposing discarded smartphones to minimize carbon," in *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023, pp. 400–412.

[17] E. Brunvand, D. Kline, and A. K. Jones, "Dark silicon considered harmful: A case for truly green computing," in *Proceedings of International Green and Sustainable Computing Conference (IGSC)*, 2018.

[18] M. Elgamal, D. Carmean, E. Ansari, O. Zed, R. Peri, S. Manne, U. Gupta, G.-Y. Wei, D. Brooks, G. Hills, and C.-J. Wu, "Carbon-efficient design optimization for computing systems," in *Workshop on Sustainable Computer Systems (HotCarbon)*, 2023.

[19] 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 *Proceedings of International Middleware Conference (Middleware)*, 2021, pp. 260–272.

[20] W. A. Hanafy, Q. Liang, N. Bashir, D. Irwin, and P. Shenoy, "Carbonscaler: Leveraging cloud workload elasticity for optimizing carbon-efficiency," in *Proceedings of International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2023.

[21] W. A. Hanafy, Q. Liang, N. Bashir, A. Souza, D. Irwin, and P. Shenoy, "Going green for less green: Optimizing the cost of reducing cloud carbon emissions," in *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024, pp. 479–496.

[22] Y. Jiang, R. B. Roy, B. Li, and D. Tiwari, "Ecolife: Carbon-aware serverless function scheduling for sustainable computing," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–15.

[23] Ericsson, "Mobililty reports," https://www.ericsson.com/en/reports-and-papers/mobility-report/reports.

[24] Statista, "Forecast number of mobile users worldwide from 2020 to 2025," https://www.statista.com/statistics/218984/number-of-global-mobile-users-since-2010/.

[25] J. Malmodin, N. Lövehagen, P. Bergmark, and D. Lundén, "Ict sector electricity consumption and greenhouse gas emissions–2020 outcome," *Telecommunications Policy*, vol. 48, no. 3, p. 102701, 2024.

[26] Z. Tu, H. Cao, E. Lagerspetz, Y. Fan, H. Flores, S. Tarkoma, P. Nurmi, and Y. Li, "Demographics of mobile app usage: Long-term analysis of mobile app usage," *CCF Transactions on Pervasive Computing and Interaction*, vol. 3, pp. 235–252, 2021.

[27] Statista, "Most popular apple app store categories as of 3rd quarter 2022, by share of available apps," https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/.

[28] ——, "Time spent with nonvoice activities on mobile phones every day in the united states from 2019 to 2024." https://www.statista.com/statistics/1045353/mobile-device-daily-usage-time-in-the-us/.

[29] Market.us., "Ai in mobile apps market: Revolutionizing user experience and innovation," https://www.linkedin.com/pulse/ai-mobile-apps-market-revolutionizing-user-experience-innovation-mn99f.

[30] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[31] M. Han, J. Hyun, S. Park, J. Park, and W. Baek, "Mosaic: Heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2019, pp. 165–177.

[32] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "ulayer: Low latency on-device inference using cooperative single-layer acceleration

and processor-friendly quantization," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2019.
[33] S. Wang, A. Pathania, and T. Mitra, "Neural network inference on mobile socs," *IEEE Design & Test*, 2020.
[34] Y. G. Kim and C.-J. Wu, "Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1082–1096.
[35] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017, pp. 615–629.
[36] Z. Song, M. Xie, J. Luo, T. Gong, and W. Chen, "A carbon-aware framework for energy-efficient data acquisition and task offloading in sustainable aiot ecosystems," *IEEE Internet of Things Journal*, 2024.
[37] W. She, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
[38] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.
[39] A. Hazra, A. Kalita, and M. Gurusamy, "Meeting the requirements of internet of things: The promise of edge computing," *IEEE Internet of Things Journal*, vol. 11, no. 5, pp. 7474–7498, 2024.
[40] A. Ali-Eldin, B. Wang, and P. Shenoy, "The hidden cost of the edge: A performance comparison of edge and cloud latencies," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.
[41] Amazon, "Aws," https://aws.amazon.com.
[42] Microsoft, "Azure: Cloud computing services," https://azure.microsoft.com.
[43] Google, "Cloud," https://cloud.google.com/.
[44] C. M. U. G. D. Institute, "Economic input-ouput life cycle assessment (eio-lca)," http://www.eiolca.net.
[45] Sphera, "Life cycle assessment software," https://gabi.sphera.com/america/index/.
[46] D. Patterson, J. M. Gilbert, M. Gruteser, E. Robles, K. Sekar, Y. Wei, and T. Zhu, "Energy and emissions of machine learning on smartphones vs. the cloud," *Communications of the ACM*, vol. 67, pp. 86–97, 2024.
[47] Q. Geng and C. Liang, "Research on the method of suppressing power fluctuation of hydro generator unit based on vortex elimination technology," *Energy Reports*, vol. 7, pp. 1038–1046, 2021.
[48] C. Jiang, Y. Qiu, W. Shi, Z. Ge, J. Wang, S. Chen, C. Cérin, Z. Ren, G. Xu, and J. Lin, "Characterizing co-located workloads in alibaba cloud datacenters," *IEEE Transactions on Cloud Computing*, vol. 10, pp. 2381–2397, 2022.
[49] D. Shingari, A. Arunkumar, B. Gaudette, S. Vrudhula, and C.-J. Wu, "Dora: Optimizing smartphone energy efficiency and web browser performance under interference," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018, pp. 64–75.
[50] S. Chen, C. Delimitrou, and J. F. Martinez, "Parties: Qos-aware resource partitioning for multiple interactive services," in *Proceedings of International Conference on Architecturla Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.
[51] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2013, pp. 29–40.
[52] A. Sridharan *et al.*, "On the impact of aggregation on the performance of traffic aware routing," *Teletraffic Science and Engineering*, vol. 4, pp. 111–123, 2001.
[53] Watttime, "Grid emissions intensity by electric grid," https://watttime.org/explorer/.
[54] EIA, "Hourly electric grid monitor," https://www.eia.gov/electricity/gridmonitor/dashboard/electric_overview/US48/US48.
[55] A. Kerai, "2023 cell phone usage statistics: Mornings are for notifications," https://www.reviews.org/mobile/cell-phone-addiction/.
[56] D. Ferreira, A. K. Dey, and V. Kostakos, "Understanding human-smartphone concerns: A study of battery life," *Pervasive Computing*, pp. 19–33, 2011.
[57] E. A. Oliver and S. Keshav, "An empirical approach to smartphone energy level prediction," 2011, pp. 345–354.
[58] Google, "Distributed cloud," https://cloud.google.com/distributed-cloud.
[59] Amazon, "Aws global infrastructure," https://aws.amazon.com/about-aws/global-infrastructure/?nc1=h_ls.
[60] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. H. Andrew, "Geographical load balancing with renewables," *ACM SIGMETRICS Performance Evaluation Review*, vol. 39, no. 3, pp. 62–66.
[61] R. Bianchini, "Leveraging renewable energy in data centers: Present and future," in *Proceedings of International Symposium on High-Performance Parallel and Distributed Computing*, 2012, pp. 135–136.
[62] S. Subramanya, T. Guo, P. Sharma, D. Irwin, and P. Shenoy, "Spoton: A batch computing service for the spot market," in *Proceedings of ACM Symposium on Cloud Computing (SoCC)*, 2015, pp. 329–341.
[63] C. ISO, "Today's outlook," https://www.caiso.com/todays-outlook/supply.
[64] M. Iorio, F. Risso, and C. Casetti, "When latency matters: Measurements and lessons learned," *ACM SIGCOMM Computer Communication Review*, vol. 51, no. 4, pp. 2–13, 2021.
[65] U. Bauknecht and T. Enderle, "An investigation on core network latency," in *2020 30th International Telecommunication Networks and Applications Conference (ITNAC)*, 2020.
[66] S. Pagani, S. Manoj, A. Jantsch, and J. Henkel, "Machine learning for power, energy, and thermal management on multicore processors: A survey," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 101–116, 2020.
[67] Y. Zhang, X. Chen, Y. Chen, Z. Li, and J. Huang, "Cost efficient scheduling for delay-sensitive tasks in edge computing system," in *Proceedings of IEEE International Conference on Service Computing (SCC)*, 2018.
[68] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "Keids: Kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem," *IEEE Internet of Thins Journal*, vol. 7, no. 5, 2020.
[69] R. Mo, F. Dai, Q. Liu, W. Dou, and X. Xu, "Multi-objective cross-layer resource scheduling for internet of things in edge-cloud computing," in *Proceedings of IEEE International Conference on Cloud Computing (CLOUD)*, 2020.
[70] S. Tuli, G. Casale, and N. R. Jennings, "Mcds: Ai augmented workflow scheduling in mobile edge cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, 2022.
[71] electricityMaps, https://www.elertricitymaps.org.
[72] Microsoft, "Calculating my carbon footprint," https://www.microsoft.com/en-us/sustainability/emissions-impact-dashboard.
[73] AWS, "Customer carbon footprint tool," https://aws.amazon.com/aws-cost-management/aws-customer-carbon-footprint-tool/.
[74] Docker, "Docker container stats," https://docs.docker.com/reference/cli/docker/container/stats/.
[75] AWS, "Amazon cloudwatch," https://aws.amazon.com/cloudwatch.
[76] Y. Choi, S. Park, and H. Cha, "Optimizing energy efficiency of browsers in energy-aware scheduling-enabled mobile devices," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 2019.
[77] Y. Wang, Y. Liu, W. Chen, Z.-M. Ma, and T.-Y. Liu, "Target transfer q-learning and its convergence analysis," *Neurocomputing*, vol. 392, pp. 11–22.
[78] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at facebook: Understanding inference at the edge," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 331–344.
[79] Statista, "App - worldwide," https://www.statista.com/outlook/dmo/app/worldwide.
[80] S. Zhao, H. Zhang, C. S. Mishra, S. Bhuyan, Z. Ying, M. T. Kandemir, A. Sivasubramaniam, and C. Das, "Holoar: On-the-fly optimization of 3d holographic processing for augmented reality," in *Proceedings of International Symposium on Microarchitecture (MICRO)*, 2021.
[81] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, G.-Y. Wei, and C.-J. Wu, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, 2020.
[82] TFLite, "Ml for mobile and edge devices," https://www.tensorflow.org/lite.
[83] Tensorflow, "An end-to-end open source machine learning platform," https://www.tensorflow.org/.
[84] T. Liu, S. He, S. Huang, D. Tsang, L. Tang, J. Mars, and W. Wang, "A benchmarking framework for interactive 3d applications in the cloud,"

in *Proceedings of International Symposium on Microarchitecture (MICRO)*, 2020.
[85] NVIDIA, "Geforce now," https://www.nvidia.com/en-us/geforce-now/.
[86] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair *et al.*, "Illixr: Enabling end-to-end extended reality research," in *Proceedings of IEEE International symposium on Workload characterization (IISWC)*, 2021.
[87] H. Kwon, K. Nair, S. Seo, J. Yik, D. Mohapatra, D. Zhan, J. Song, P. Capak, P. Zhang, P. Vajda, C. Banbury, M. Mazumder, L. Lai, A. Sirasao, T. Krishna, H. Khaitan, V. Chandra, and V. J. Reddi, "Xrbench: An extended reality (xr) machine learning benchmark suite for the metaverse," in *International Conference on Machine Learning and Systems (MLSys)*, 2023.
[88] Y. G. Kim, M. Kim, and S. W. Chung, "Enhancing energy efficiency of multimedia applications in heterogeneous mobile multi-core processors," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1878–1889, 2017.
[89] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Mcikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeria, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," *arXiv:1911.02549*, 2019.
[90] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, pp. 40–45, 2006.
[91] Qualcomm, "Snapdragon 845 mobile platform," https://www.qualcomm.com/products/snapdragon-845-mobile-platform.
[92] Monsoon, "High voltage power monitor," https://www.msoon.com/high-voltage-power-monitor.
[93] MLCommons, https://mlcommons.org/.
[94] X. Zhang, Z. Shen, B. Xia, Z. Liu, and Y. Li, "Estimating power consumption of containers and virtual machines in data centers," in *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER)*, 2020.
[95] X. Zhao, Y. Lu, Z. Li, J. Tan, Y. Feng, and Y. Tao, "Explicitly consider server-attached fans for thermal modeling in edge data centers," in *Proceedings of ACM International Conference on Future Energy Systems (e-Energy)*, 2020.
[96] A. Gupta, I. Jain, and D. Bharadia, "Multiple smaller base stations are greener than a single powerful one: Densification of wireless cellular networks," in *1st Workshop on Sustainable Computer Systems Design and Implementation (HotCarbon)*, 2022.
[97] Cisco, https://www.cisco.com/c/en/us/products/collateral/routers/8000-series-routers/datasheet-c78-742571.html.
[98] F. Y. Ettoumi *et al.*, "Statistical analysis of solar measurements in algeria using beta distributions," *Renewable Energy*, vol. 26, no. 1, pp. 47–67, 2002.
[99] A. N. Celik, "A statistical analysis of wind power density based on the weibull and rayleigh models at the southern region of turkey," *Renewable energy*, vol. 29, no. 4, pp. 593–604, 2004.
[100] buildfire, "Mobile app download statistics & usage statistics (2023)," https://buildfire.com/app-statistics/.
[101] S. A. Stoev, G. Michailidis, and J. Vaughan, "Global modeling and prediction of computer network traffic," *arXiv:1005.4337v1*.
[102] H. Castrup, "Distributions for uncertainty analysis," in *Proceedings of International Dimensional Worklshop (IDW)*, 2001.
[103] R. Hanhan, E. Garzon, Z. Jahshan, A. Teman, M. Lanuzza, and L. Yavits, "Edam: Edit distance tolerant approximate matching content addressable memory," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2022.
[104] V. Tsoutsouras, O. Kaparounakis, B. Bilgin, C. Samarakoon, J. Meech, J. Heck, and P. Stanley-Marbell, "The laplace microarchitecture for tracking data uncertainty and its implementation in a risc-v processor," in *Proceedings of International Symposium on Microarchitecture (MICRO)*, 2021.
[105] Dell, "Life cycle assessment of dell r740," https://www.delltechnologies.com/asset/en-us/products/servers/technical-support/Full_LCA_Dell_R740.pdf.
[106] H. Liao, G. Tang, D. Guo, Y. Wang, and R. Cao, "Rethinking low-carbon edge computing system design with renewable energy sharing," in *Proceedings of the 53rd International Conference on Parallel Processing*, 2024, pp. 950–960.

Yonglak Son received his BS degree in software engineering from Soongsil University. He is currently a integrated Ph.D student in the Division of Computer and Communication Engineering, Korea University. His research interests include energy efficient design and efficient systems for machine learning.

Udit Gupta is currently an assistant professor in electrical and computer engineering at Cornell Tech, Roosevelt Island, NY, 10044, USA, and a visiting research scientist at Meta AI Research. His research interests include improving the performance, efficiency, and sustainability of computer systems and architectures by co-designing solutions across the computing stack.

Andrew McCrabb received the bachelor's degree in electrical engineering at Auburn University, in Auburn, Alabama, and the MS degree in computer science and engineering from the University of Michigan, Ann Arbor, Michigan, in 2016 and 2019, respectively. He is currently working toward the PhD degree in the Computer Science and Engineering department at the University of Michigan. His research interests include hardware acceleration, graph analytics, big data analysis, artificial intelligence, semi-structured data, and near-memory computing. He received the Towner Prize for Distinguished Academic Achievement by the University of Michigan, Ann Arbor, Michigan, in 2020.

Young Geun Kim is currently an assistant professor in the Department of Computer Science and Engineering, Korea University. His research focus lies in the domain of computer system architecture with particular emphasis on energy efficient and low-power systems. His research has pivoted into designing efficient systems for machine learning execution at the edge. He received his BS and Ph.D. degrees in the Department of Computer Science from Korea University in 2014 and 2018, respectively.

Valeria Bertacco received the Laurea degree in computer engineering from the University of Padova, Italy, and the MS and PhD degrees in electrical engineering from Stanford University, Stanford, California, in 2003. She was at Synopsys, Inc., Mountain View, California, before joining the University of Michigan, Ann Arbor, Michigan, where she is currently Arthur F. Thurnau professor of computer science and engineering and vice provost for engaged learning. She is also the director of Applications Driving Architectures (ADA) Center, a JUMP center co-sponsored by SRC and DARPA. Her research interests include computer design, with emphasis on specialized architecture solutions and design viability, in particular reliability, validation and hardware-security assurance. She has received the IEEE CEDA Early Career Award, served as chair of the DAC program committee, and track-chair of the DATE program committee.

David Brooks received the B.S. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 1997, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, USA, in 1999 and 2001, respectively. He is currently the Haley Family Professor of computer science with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. His current research interests include resilient and power-efficient computer hardware and software design for high-performance and embedded systems. Dr. Brooks was a recipient of several honors and awards, including the ACM, Maurice Wilkes Award and ISCA Influential Paper Award.

Carole-Jean Wu is an Adjunct Professor at Arizona State University and is currently a Research Director at Meta AI Research. Her research lies in the domain of computer systems. Her recent research focuses on designing systems for machine learning execution at-scale and on tackling system challenges to enable efficient AI execution in a responsible way. Wu received a B.Sc. degree from Cornell University, and M.A. and Ph.D. degrees from Princeton University.