Author/s:
Hewage, TB;Ilager, S;Read, MR;Buyya, R

Title:
Aging-aware CPU Core Management for Embodied Carbon Amortization in Cloud LLM Inference

Date:
2025-06-16

Citation:
Hewage, T. B., Ilager, S., Read, M. R. & Buyya, R. (2025). Aging-aware CPU Core Management for Embodied Carbon Amortization in Cloud LLM Inference. E-Energy '25: Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems, pp.43-55. Association for Computing Machinery. https://doi.org/10.1145/3679240.3734608.

Persistent Link:
https://hdl.handle.net/11343/361928

Aging-aware CPU Core Management for Embodied Carbon Amortization in Cloud LLM Inference

Tharindu B. Hewage
University of Melbourne
Melbourne, Australia
tsaryakarahe@student.unimelb.edu.au

Shashikant Ilager
University of Amsterdam
Amsterdam, Netherlands
s.s.ilager@uva.nl

Maria Rodriguez Read
University of Melbourne
Melbourne, Australia
maria.read@unimelb.edu.au

Rajkumar Buyya
University of Melbourne
Melbourne, Australia
rbuyya@unimelb.edu.au

Abstract

Broad adoption of Large Language Models (LLM) demands rapid expansions of cloud LLM inference clusters, leading to the accumulation of embodied carbon—the emissions from manufacturing and supplying IT assets—that mostly concentrate on inference server CPU. This paper delves into the challenges of sustainable growth of cloud LLM inference, emphasizing extended amortization of CPU embodied over an increased lifespan. Given the reliability risks of silicon aging, we propose an aging-aware CPU core management technique to delay CPU aging effects, allowing the cluster operator to safely increase CPU life. Our technique exploits CPU underutilization patterns that we uncover in cloud LLM inference by halting aging in unused cores and even-outing aging in active cores via selective deep idling and aging-aware inference task allocation. Through extensive simulations using real-world Azure inference traces and an extended LLM cluster simulator from Microsoft, we show superior performance of our technique over existing methods with an estimated 37.67% reduction in yearly embodied carbon emissions through p99 performance of managing CPU aging effects, a 77% reduction in CPU underutilization, and less than 10% impact to the inference service quality.

CCS Concepts

• **Computer systems organization → Cloud computing**; • **Social and professional topics → Sustainability**; • **Theory of computation → Online algorithms**.

Keywords

Sustainable AI, Embodied Carbon Reduction, CPU Aging, LLM Inference

ACM Reference Format:
Tharindu B. Hewage, Shashikant Ilager, Maria Rodriguez Read, and Rajkumar Buyya. 2025. Aging-aware CPU Core Management for Embodied Carbon Amortization in Cloud LLM Inference. In *The 16th ACM International Conference on Future and Sustainable Energy Systems (E-ENERGY '25), June 17–20, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3679240.3734608

ACM ISBN 979-8-4007-1125-1/25/06
https://doi.org/10.1145/3679240.3734608

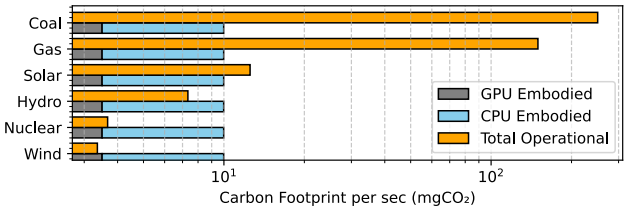


Figure 1: The carbon footprint of an A100x4 GPU server running per second inference application when powered by energy sources with different carbon intensity [26].

1 Introduction

The proliferation of applications driven by cloud-based generative Large Language Model (LLM) inference is seen across diverse domains, such as conversational agents [35], education [8], and coding assistance [16]. As the popularity of such applications scales their user base to billions [43], cloud service providers continue to expand LLM inference clusters towards the GigaWatt scale to support the growing demand [11]. Recently, Meta announced its plans to build a brand new data center targeting AI workloads [34], and xAI plans to expand its AI cluster from 100K to 1 million GPUs [12].

LLM Inference clusters deploy and serve pre-trained LLMs [36]. In inference, a user request (i.e., prompt query) is split into a set of input tokens. Input tokens are then fed to the model (i.e., forward pass) to generate the first output token and an intermediate context called KV-cache. KV-cache and the first token are then fed for the second forward pass, and the process is repeated until a stopping condition is met [36]. In return, a single request can incur many forward passes. To reduce the latency in that, clusters employ parallel model computation via GPU accelerators [31]. Due to memory constraints, each server typically utilizes several GPUs to support modern LLMs with billions of parameters [40]. In return, LLM inference becomes both memory and compute-intensive [46]. When serving LLMs at scale, inference clusters employ many inference optimization techniques to better utilize underlying resources, such as phase splitting [36] and iteration-level scheduling [52]. As

a result, LLM inference at scale results in a complex set of CPU tasks (i.e., inference tasks), such as facilitating steps of optimization techniques [36, 52], request scheduling [46], and tokenization [26].

The growth of LLM inference clusters also increases cloud infrastructure carbon footprint [7, 11, 34]. It combines two aspects: direct carbon emissions owing to energy sources (i.e., operational) and indirect carbon emissions resulting from business activities such as manufacturing, shipping, and recycling IT assets (i.e., embodied) [9, 53]. Today, as cloud service providers invest in renewable energy generation to meet net-zero emission goals [10, 34], renewable energy sources with lesser carbon intensity [29] continue to penetrate power grids [7], diminishing the effect of operational carbon over the embodied. Microsoft, a hyper-scale cloud provider, reported that over the past four years, its operational carbon was reduced by 6.3 percent while embodied increased by 30.9 percent [5]. In LLM inference clusters, most of its embodied carbon accounts for CPU components, including the die and mainboard [26]. Therefore, optimizing CPU embodied becomes paramount for the sustainable growth of LLM inference clusters. Figure 1 illustrates that. With lesser carbon-intensive renewable energy sources, the CPU embodied becomes the dominant carbon aspect in inference servers.

We study the problem of optimizing the CPU embodied in LLM inference clusters. Inference clusters amortize the CPU embodied over its lifetime. Therefore, extending CPU life further amortizes its embodied carbon. CPU life extensions are typically achieved through extending its hardware refresh cycle [18]. CPU hardware refresh cycle replaces CPUs with newer hardware generations. Its aim is to gain performance-per-watt improvements [18] and avoid reliability risks of silicon aging [17, 18, 44]. However, CPU performance gains in that are minimal for inference clusters. This is because CPU tasks in inference clusters carry out GPU-accelerated LLM inference, and these inference tasks mostly benefit from single-core performance, which has plateaued in recent years [45]. As a result, the sole aim of maintaining a standard hardware refresh cycle is to avoid silicon aging. In this context, extending the CPU hardware refresh cycle requires efficient management of the CPU to delay the effects of silicon aging. It is worth noting that this is not the case for GPU, for which the embodied carbon footprint is smaller and performance gains of newer hardware generation are significant [26]. Many works exploring silicon aging management in CPUs employ efficient aging-aware workload management [1, 17, 27, 54]. They leverage task scheduling among CPU cores to even out core aging and, in return, slow down the aging rate of the overall CPU [17, 27, 44, 54]. However, leveraging the opportunities present in CPU usage patterns of cloud LLM inference is yet to be explored.

To this end, we propose an aging-aware CPU core management technique to extend the CPU life in inference clusters. In return, cluster embodied carbon is further amortized over the increased lifespan. We design our technique for CPU usage patterns in cloud LLM inference. Using production inference traces, we uncover that LLM inference clusters mostly underutilize CPU cores with occasional usage bursts. To exploit that, we design a dynamic **working set** of cores where the cores in the set remain active while others deep idle [48]. We then design online algorithms that (1) identify and adjust the *working set* based on usage bursts and (2) assign inference tasks inside the *working set* to even out aging across cores. Collectively, our approach achieves age-halting and reduced underutilization of cores. The *working set*, however, can lead inference tasks to oversubscribe the CPU if not scaled in time. The online algorithms we propose are also designed to mitigate that.

We implement our approach by extending splitwise-sim, a high-fidelity LLM cluster simulator from Microsoft [36]. We use production LLM inference traces generated with data collected from LLM inference services in Azure [36] and use state-of-the-art CPU core management techniques as baselines. Results for our experimental cluster show an estimated 37.67% reduction in yearly embodied carbon emissions through p99 performance of managing CPU aging effects and reduction of CPU core underutilization by 77%, all while maintaining CPU oversubscription below 10%. The **key contributions** of our work are as follows:

(1) An investigation into the role of the CPU in state-of-art LLM inference clusters and uncovering CPU underutilization patterns using production traces.
(2) Propose a new technique for aging-aware CPU core management using dynamic age-halting of deep idling CPU cores.
(3) Implement the proposed technique in a simulated environment and conduct extensive experiments using production inference traces.
(4) An evaluation of our proposed technique against state-of-the-art CPU core management baselines, focusing on its efficiency in managing CPU core aging, reducing yearly embodied carbon, and controlling task-related CPU oversubscription.

2 Background and Motivation

In this section, we provide background on embodied carbon amortization in cloud servers and optimizing it by extending CPU life. We then outline our investigations on applying that in cloud LLM inference clusters. We highlight key takeaways from where we draw our motivations for this paper.

2.1 Background: Embodied Carbon Amortization in Cloud Servers through CPU Age Management

In managing the carbon footprint of cloud data centers, Green House Gas (GHG) protocol, a global standard formed to manage GHG emissions [37], defines three scopes. Scopes 1 and 2 represent the operational carbon, typically owing to the carbon intensity of the data center's energy sources. Scope 3 represents embodied carbon: carbon emissions that indirectly result from the manufacturing and shipping of servers and other IT assets that have already been built and installed in data centers [9]. Unlike operational carbon, which can be optimized by adopting less carbon-intensive energy sources, embodied carbon needs to be amortized over the asset's lifespan. Here, amortization is a way to account for embodied carbon. For example, if a server with a 4-year operational lifetime causes 1000 $kgCO_2eq$ of Scope 3 emissions, then amortization accounts for 250 $kgCO_2eq$ of embodied carbon emissions per year.

Recent studies of embodied carbon optimization outline three tenets of environmental design: reduce, reuse, and recycle [18]. Out of that, this paper focuses on recycling, more specifically enabling

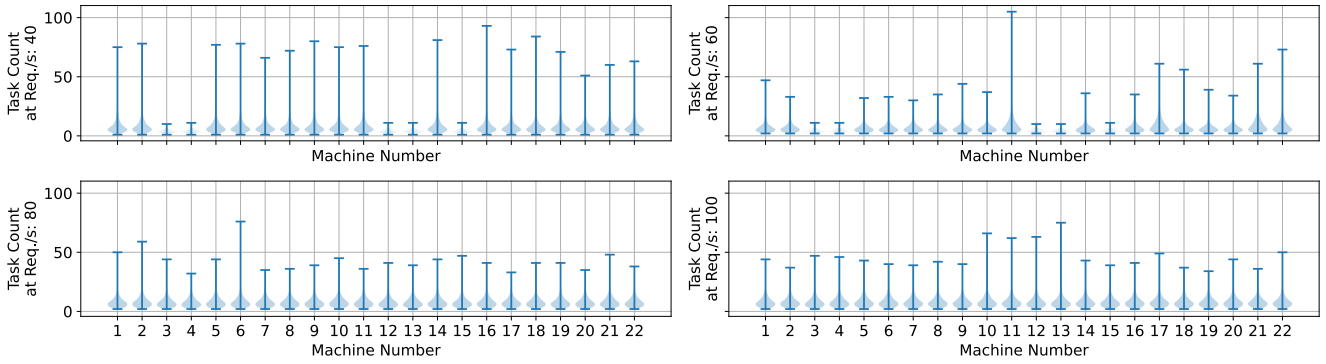


Figure 2: Distributions of running inference tasks in an LLM inference cluster of 22 H100 machines.

a second life of the CPU by improving its reliability to extend the lifetime [18]. The primary reason for CPU reliability degradation is the silicon aging of its transistors beyond the rated life [44]. Many works studying silicon aging in CPU [17, 39, 44] model with **Negative Bias Temperature Instability (NBTI)**. NBTI is an aging mechanism that affects PMOS transistors in the CPU [1]. It is caused by the stress of workload execution. During workload execution, transistors in the CPU continue to switch, applying stress on transistors and releasing them back. When stress is applied, NBTI shifts the transistor's threshold voltage (ΔV_{th}) but leaves a residual shift when the stress is removed. That incurs a slight increase in the ΔV_{th}, accumulating over time. As a result, critical path delay in the circuit increases, reducing the maximum operating frequency of the CPU (i.e., **CPU Aging**). Since NBTI aging results from workload execution, workload management techniques can mitigate that for improved CPU reliability [17, 27, 39, 44]. In return, embodied carbon is further amortized through a second life of the CPU.

2.2 Motivation: Impact of CPUs' on Embodied Carbon in LLM Inference Clusters

Servers in LLM inference clusters serve generative AI requests via low-latency model computation through GPU accelerators. Therefore, most of the inference server's thermal design power (TDP) comes from the GPU. In return, GPU dominates the server's operational carbon footprint [26]. Nevertheless, the electrical grids powering inference clusters continue to integrate low-emission renewable energy sources [20]. As a result, the operational carbon intensity of inference servers continues to diminish, whilst embodied carbon accounts for the majority of the server's carbon footprint (see Figure 1).

The embodied carbon of an inference server consists of two categories: CPU embodied and GPU embodied. GPU is packaged into an independent hardware unit and installed using a standard interface such as PCIe. In return, it is loosely coupled from the server and straightforward to replace [23]. In contrast, the CPU is installed as a tightly coupled component with many other associated components, such as the CPU chassis, mainboard, power delivery components, etc [22, 25, 26]. Moreover, CPU components have strict compatibility requirements. For instance, the CPU die must be compatible with the mainboard's socket type and its chipset generation [21, 22].

Thus, replacing individual CPU components has become increasingly unsustainable in cloud platforms due to multiple reasons. Firstly, maintaining a continuous supply of spare components with fine-grained compatibility requirements is difficult to procure due to the poor availability of repair parts [6, 28]. Secondly, recent technology trends in data centers, such as utilizing liquid cooling, significantly increase the time and effort required to repair servers at the component level [28]. Therefore, upon a CPU component failure, cloud providers tend to replace the CPU and the associated components as a whole [42]. In this context, the CPU embodied in an inference server is attributed to the embodied carbon footprint of the CPU and its associated components [25, 26].

Refined carbon modeling studies in LLM inference clusters show a substantial impact from CPU embodied towards the cluster's embodied carbon footprint [25]. In an Azure T4 inference server, the GPU embodied calculates only 41.8 kgCO2eq emissions for its total lifetime. However, the CPU embodied accounts for 278.3 kgCO2eq, while 54% of this is attributable to the CPU die, the CPU chassis, and the mainboard alone [26]. Therefore, cloud providers are challenged with lowering the rate of accumulation of CPU embodied and achieving extended amortization of procured CPU embodied.

Inference clusters acquire CPU embodied faster than it can amortize through its hardware refresh life cycle. A hardware refresh cycle aims to gain performance improvements of newer CPU hardware generations and avoid reliability concerns of aging CPUs [18]. Recent studies show that LLM inference clusters may not gain significant performance benefits from newer CPU hardware generations. CPUs in inference clusters execute tasks (i.e., inference tasks) facilitating the inference workflows, such as phase splitting, request scheduling, batching, and tokenization [26, 36]. These typically benefit from single-core performance, yet the yearly single-core performance of newer CPU hardware generations has been mostly the same [45]. That leaves avoiding reliability concerns of CPU aging as the sole benefit of the CPU hardware refresh cycle.

It's important to note that the reliability concerns of aging are significant in the CPU, whereas its associated components, such as the mainboard, pose minimal risks. For instance, long-term failure

analysis of servers shows mainboard failure rates have dropped over time [42]. Further, many associated components, such as storage and memory, already implement error correction mechanisms for aging hardware, such as Error Correction Codes (ECC) [13]. In contrast, erroneous computations of unreliable aging CPUs are extremely difficult to correct through preventive mechanisms and have become increasingly frequent in hyper-scale cluster deployments [13, 19].

As discussed in Section 2.1, an efficient aging-aware workload management technique can mitigate CPU reliability concerns, which also translates to an extension to the CPU hardware refresh cycle, allowing the cluster to further amortize its CPU embodied.

Takeaways: *Extended amortization of CPU embodied significantly reduces the carbon footprint of inference clusters. However, it is constrained by the reliability concerns of CPU aging. Hence, there is an opportunity for an effective aging-aware CPU management technique to optimize that.*

CPU Utilization Patterns in LLM Inference Clusters: To design an efficient aging-aware CPU management technique, it is important to understand CPU utilization patterns in cloud LLM inference. For that, we monitor and analyze CPU utilization patterns in a high-fidelity simulated LLM inference cluster environment that infers real workload traces.

We use an LLM cluster simulator from Microsoft [36] and extend it to model CPU in cloud LLM inference. We then replay production inference traces from Azure and observe cluster CPU utilization. In that, we allocate each CPU task to a dedicated core. Our cluster settings closely match that in production. In Section 5, We discuss our experimental setup in detail. Figure 2 illustrates our results. Each subplot maps to a different throughput level and shows the distribution of concurrent inference tasks executed in each cluster machine. The x-axis denotes the machine number, and the y-axis denotes the inference task count. We make two key observations in that.

- ***O1:*** *Cores are mostly underutilized, as indicated by the lower mean values in the violin plots.*
- ***O2:*** *There are occasional bursts of running tasks as indicated by the maximum values of the violin plots, which justifies having CPUs with higher core counts.*

In our simulation, our key finding is that underutilized CPU cores are available to the machine operating system to schedule system tasks apart from the inference serving platform. Therefore, these cores can actively execute system tasks in a time-shared manner [51]. In return, all cores can actively execute instructions regardless of being allocated to an inference task, thus continuing to age due to the transistor stress of workload execution. In this context, we identify an opportunity to halt aging in the underutilized cores. We ***hypothesize to reduce the available cores to match the number of running tasks of the inference platform***. In return, we can put the remaining cores to deep idle, which turns off the clock and power gates the CPU cores [50, 51], stopping the transistor switching and halting the core aging. However, doing so can introduce a new set of challenges. As shown in Figure 2, the number of concurrent inference tasks can dynamically change. Therefore, limiting the available cores can lead inference tasks to oversubscribe the CPU unless the number of available cores is scaled in time. Moreover, a reduced set of available cores can introduce core affinity, which can increase the failure risks of individual CPU cores due to uneven core aging [54].

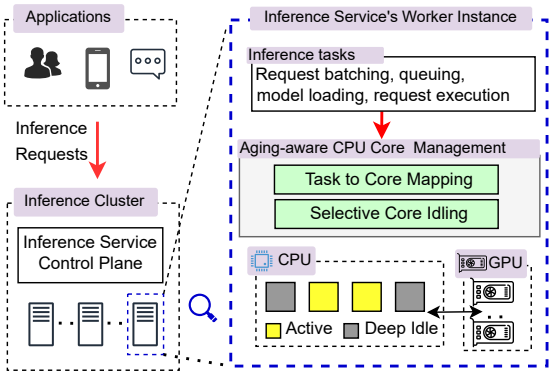


Figure 3: High-Level system diagram of aging-aware CPU core management in LLM inference clusters.

Takeaways: *Underutilized cores in cloud LLM inference provide the opportunity to halt CPU aging through deep idling of unused cores. However, it presents new challenges, including timely switching of core idle states to reduce CPU oversubscription and efficient use of the available cores to avoid uneven core aging.*

3 System Model and Problem Formulation

This section presents the system model, its components, and the formulation of the problem.

3.1 System Model

Figure 3 provides a high-level view of our system model. We model a high-performance LLM inference cluster deployment with inference-optimized servers. It executes cloud LLM inference services, and we assume it does not co-locate other types of workloads. Our model is based on similar production settings in practice where an inference cluster is reserved for LLM inference services [36]. Firstly, the inference requests from end-user applications reach the cluster's inference service. Each request is then scheduled to servers by the cluster-level scheduler. Inference servers run virtualized worker instances, which conduct request batching, queuing, model loading, and finally execute the request leveraging an inference backend, such as vLLM [2]. Inference backend efficiently utilizes CPU and GPU resources and may leverage high-bandwidth InfiniBand interconnections between GPUs, such as sharing intermediate KV-cache in phase splitting [36]. Our system architecture matches that of production deployments, such as the NVIDIA triton server inference architecture on Kubernetes [2].

In return, the server-level worker instance of the inference service executes many CPU tasks (i.e., inference tasks). For each inference task, we allocate a dedicated CPU core. If the cores are insufficient, we assume that inference tasks oversubscribe the CPU. We introduce a new component to conduct aging-aware CPU core management. It oversees assigning cores to inference tasks and controlling the idle states of CPU cores. A CPU core in our system

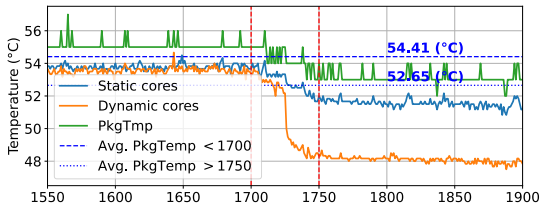


Figure 4: Changes in operating temperature when half of the cores are set to deep idle in an Intel Xeon CPU. If awake, cores are 100% utilized.

model can switch between either active or deep idle states [48]. Being in the active state gradually ages the CPU cores. In contrast, deep idling halts cores from aging. However, cores that deep idle become unavailable for inference task execution. Cores in the active state are available for task execution, yet allocating a task accelerates core aging. In Section 3.2, we provide in-depth details about the silicon aging behavior with our aging model. CPU cores in the host are isolated and pinned to worker instance CPU cores, such that task mapping and idle state changes directly reflect on the physical core. Overall, the combination of inference task execution and deep idling control the CPU aging rate, where a lower value further amortizes its embodied carbon through the increased lifespan [54].

3.2 Aging Model

We model the aging of CPU cores due to the execution of system and inference tasks. In that regard, Negative-bias Temperature Instability (NBTI) is a major aging mechanism [1, 17] for the CPU. In this work, we model NBTI-induced core aging. Similar to previous works [1], we use a reaction-diffusion-based aging model to calculate CPU core frequency degradation due to NBTI-induced aging.

$$f(t) = f_0 \times (1 - \frac{\Delta V_{th}}{V_{dd} - V_{th}}) \tag{1}$$

where $f(t)$ is the frequency at time t, and f_0 is the initial frequency of the core. Previous works show that f_0 can deviate from the nominal value due to variations in the manufacturing process [17, 38]. To accommodate that, we use the following model to calculate f_0.

$$f_0 = K' \min_{k,l \in S_{CP}} \left(\frac{1}{p_{kl}}\right).$$

where K' is a technology-dependent constant, and S_{CP} represents the sections of the core containing critical paths. In order to calculate f_0, we first divide the chip area into an $N_{\text{chip}} \times N_{\text{chip}}$ grid and assume that critical paths are contained entirely within the grid cells. Then, we assign each grid cell with a Gaussian random variable (p_{kl}). In order to calculate the spatial correlation between the random variables, we use the following formula [38].

$$\rho_{ij,kl} = e^{-\alpha \sqrt{(i-k)^2+(j-l)^2}} \quad \forall\, i, j, k, l \in [1, N_{\text{chip}}].$$

where α decides how quickly spatial correlations die out. For our experiments, we set N_{chip} to 10 and K' to 1. Then, we set the mean of random variables by solving for a scenario where if a core does not exhibit process variation, f_0 should equal the nominal value. We set the remaining parameters similar to the calculation of a previous work [38].

Table 1: Temperature modeling for different core states.

Idle-state	C-state [48]	Inference Task	Temp.
Active	C0	Allocated	$54°C$
Active	C0	Unallocated	$51.08°C$
Deep Idle	C6	N/A	$48°C$

After f_0, we calculate ΔV_{th} in Equation 1, which is the shift in the threshold voltage. CPU cores in our system can undergo time intervals in different idle states. In order to calculate ΔV_{th} in those, we calculate ΔV_{th} using the following recursive equation [32].

$$\Delta V_{\text{th}}(t_p) = ADF_p \left[\left(\frac{\Delta V_{\text{th}}(t_{p-1})}{ADF_p} \right)^{\frac{1}{n}} + \tau_p \right]^n$$

where $V_{\text{th}}(t_p)$ is the value at p^{th} time interval, and τ_p is the length of the p^{th} time interval. ADF is a time-independent factor for each time interval, which we calculate using the following equation.

$$ADF(T, V_{\text{dd}}, Y) = K \cdot \exp\left(-\frac{E_0}{k_B T}\right) \cdot \exp\left(\frac{B\, V_{\text{dd}}}{t_{\text{ox}}\, k_B T}\right) \cdot Y^n \tag{2}$$

where Y is the stress from the executing task. We assume each task in our system incurs the worst case by setting it to 1.0. T is the operating temperature of the CPU core. In order to create a realistic temperature model, we conduct an experiment by running a high utilization task in a server-grade CPU and switching its cores between active and deep idle states. During the experiment, we monitor the changes in core temperatures. Figure 4 illustrates our observations. Table 1 denotes the temperature model we derive from that. We set the rest of the parameters in equation 2 as follows. K is a fitting parameter. To calculate its value, we use CPU aging data from a previous work, which states that for 22nm CPU technology, the worst-case frequency reduction due to aging for a lifetime of 10 years can reach 30% [1]. We set values of our model to match this scenario and solve the ΔV_{th} equation to find the value for K. All parameters that were not explicitly mentioned are set similarly to a previous work matching for the 22nm CPU technology [1].

3.3 Embodied Carbon Model

Our carbon modeling is based on our findings in section 2.2. We model the embodied carbon of the CPU and its associated components in LLM inference servers to accurately measure the carbon amortization optimizations of our proposed technique. We use recent fine-grained carbon modeling [26], encompassing different configurations and components of multi-GPU inference servers, such as the peripheral components for cooling and power delivery. With that, the yearly CPU embodied carbon footprint of an LLM inference server ($CF^{\text{emb,cpu}}$) is modeled as,

$$CF^{\text{emb,cpu}} = \frac{1}{LT}\left(N_r K_r + \sum_k CF_k\right)$$

where $k \in$ {CPU die, CPU Chassis, Mainboard, DRAM, PDN, etc.}, CF_k is the carbon footprint of associated component k, LT is the lifetime in years, and $N_r K_r$ is the packing carbon footprint.

In section 6, we use our yearly CPU embodied carbon model to quantify the resulting carbon savings of the proposed technique. In that, we use concrete values of CPU embodied carbon footprint calculated using our carbon model for the Azure T4 inference server with 16GB GDDR6 denoted by the VM flavor Standard_NC4as_T4_v3 [4, 26].

3.4 Problem Formulation

This paper considers a cluster of LLM inference servers managed by an inference service. It serves inference requests arriving from cloud users.

At the server level, the GPU-accelerated inference requests result in a dynamic number of concurrent CPU tasks (i.e., inference tasks). They are handled by a multi-core CPU. Due to manufacturing process variations, each CPU core exhibits a maximum frequency value that deviates from the nominal value, degrading over time with workload execution due to core aging. Depending on the task allocation and management of core idle states, the rate of frequency degradation among the cores can differ. Over time, the multi-core CPU exhibits a distribution of degraded frequencies among its cores, increasing their failure risks [54]. In addition, the service quality of serving inference tasks can be impacted if active cores are insufficient to facilitate the running inference tasks. In this context, we formulate our problem as follows.

Our multi-core CPU contains an N number of CPU cores. We denote the number of deep idling cores at time t with $N_{idle}(t)$, the number of executing tasks with $T(t)$, and the frequency of the core i with $f_{core_i}(t)$. For the duration of ΔT, the following equation calculates the reduction of core frequency due to core aging ($f_{red_i}(\Delta T)$) for the i^{th} core.

$$f_{red_i}(\Delta T) = f_{\text{core},i}(t) - f_{\text{core},i}(t + \Delta T) \quad i \in N$$

Similarly, the following equation calculates the variance in the CPU core frequency distribution ($f_{\text{var}}(\Delta T)$).

$$f_{\text{var}}(\Delta T) = \text{Var}(F) \quad \text{where } F = \left\{ f_{\text{core},i}\left(t + \Delta T\right) : i \in N \right\}$$

Finally, the following equation quantifies service quality impact from inference serving due to core deep idling ($T_{oversub}(\Delta T)$).

$$T_{oversub}(\Delta T) = \int_{t}^{t+\Delta T} u(T(t) - (N - N_{idle}(t))) \times (T(t) - (N - N_{idle}(t)))\, dt$$

Where u is the unit step function. The goal of our problem is to extend amortizing CPU embodied carbon through increasing its operating lifespan by mitigating CPU aging effects. For that, both per-core and uneven aging effects across cores must be reduced. Moreover, the impact on the service quality of inference tasks must also be reduced. With that, we state our problem,

$$\begin{aligned} \text{Min.} \quad & f_{red_i}(\Delta T) \quad \forall i \\ \text{Min.} \quad & f_{\text{var}}(\Delta T) \\ \text{Min.} \quad & T_{oversub}(\Delta T) \end{aligned}$$

4 Extended Embodied Carbon Amortization through Aging-aware CPU Core Management in LLM Inference Clusters

In this section, we provide the design and inner workings of our proposed aging-aware CPU core management technique. Our design is based on our findings in Section 2.2. We evaluate the performance of the proposed technique in Section 6, showcasing its potential to reduce yearly embodied carbon emissions of inference clusters.

Figure 3 illustrates the design of the proposed technique. We optimize CPU core aging at the server level to extend CPU lifetime, matching the CPU GPU asymmetric lifetime. To implement our proposed *aging-aware core management*, we introduce two main mechanisms: *(1) Task to Core Mapping* and *(2) Selective Core Idling*. *Task to Core Mapping* runs an algorithm to decide the mapping of each inference task to a CPU core. It aims to mitigate uneven aging among available CPU cores. Whereas *Selective Core Idling* runs an algorithm to determine a *working set* of cores to match the current inference throughput. It halts the aging of cores in the non-working set by setting them to deep idle. Apart from age halting, it further complements uneven core aging by selecting cores to deep idle in an aging-aware manner.

Together, the proposed technique reduces the overall aging rate of the CPU in two aspects. Both *Task-to-Core Mapping* and *Selective Core Idling* **even-out aging** across cores, preventing early effects of premature aging in specific cores. *Selective Core Idling* **halts aging** in cores when the inference throughput provides opportunities to do so. It delays aging effects in cores.

4.1 Task-to-Core Mapping

The primary goal of *Task-to-Core Mapping* is to reduce the age variance among CPU cores. To achieve that, it distributes the stress of inference tasks favoring lesser-aged cores. As a result, older cores age slower, delaying overall aging effects.

Algorithm 1 outlines the proposed algorithm for *Task-to-Core Mapping*. It takes the set of active cores (i.e., *working set*) as the input and selects a core to run an inference task. Therefore, each new inference task executes the algorithm 1 once. To reduce the execution time, we design the algorithm to leverage an age estimation approach for its selection logic rather than obtaining CPU micro-architectural attributes to calculate an accurate value. To achieve that, each core in the input *working set* provides two additional attributes: task-assigned status and its idle history. Using a core's idle history, we calculate an estimation for its age. We maintain a core's last eight idle durations, similar to that of the Linux governor algorithm [48]. At execution, we create placeholders for both the selected core and its idle score (line 1). Then, we iteratively evaluate each core in the *working set*. We calculate an idle score for each core that has not been assigned a task yet (line 7). Idle score accumulate all idle durations in the provided history. Here, the insight is that if a core mostly remained idle, its aging rate is lower than that of a less idle core. We then use the idle score to conduct a relative comparison among the cores to filter the core with the most idle score (line 8). As a result, the core with the least estimated age is selected to execute the next inference task.

Overall, the algorithm estimates the age of each core using a rolling idle duration window and distributes the stress of executing

Algorithm 1 Proposed Task-to-Core mapping algorithm.

Require: *cpu_cores*: The *working set* of cores. Each has:
- `task`: Inference task assigned (or None if no task),
- `last_idle_durations`: Recent idle durations.

Ensure: The selected core to run the inference task.

```
1:  selected_core ← None
2:  selected_idle_score ← 0.0
3:  for all core in cpu_cores do
4:      if core.task ≠ None then
5:          continue
6:      end if
7:      idle_score ← ∑(core.last_idle_durations)
8:      if (selected_core = None)
        or (idle_score > selected_idle_score) then
9:          selected_core ← core
10:         selected_idle_score ← idle_score
11:     end if
12: end for
13: return selected_core
```

inference tasks in a least-aged-first manner. In return, the aging effects of cores take a prolonged time to appear, i.e., slowing down the CPU aging rate. Our age estimation approach avoids the overhead of calculating an accurate aging value. Since *Task-to-Core Mapping* is executed quite frequently in the cloud environments, a minimum execution overhead ensures reduced latency impact on inference request serving.

4.2 Selective Core Idling

In addition to the reduction of the aging rate of *Task-to-Core Mapping*, we provide a core-level optimization mechanism called *Selective Core Idling*, which halts aging. It contributes to the overall aging reduction in the CPU by dynamically halting core aging whenever the inference task execution is able to tolerate that.

The main idea behind *Selective Core Idling* is to leverage the unused CPU cores in cloud LLM inference for deep idling (see Section 2.2). We do that by dynamically adjusting the size of the *working set* of cores and using the remaining cores for deep idling. The key challenge here is to match the size of the *working set* to the number of running inference tasks. If the *working set* is smaller, then inference tasks begin to oversubscribe the CPU, whereas a larger *working set* leaves a portion of unused cores in the active state, which otherwise would have been utilized for deep idling. To address that, we design an algorithm for *Selective Core Idling*. The main part of our algorithm is a module called a reaction function. The reaction function decides the algorithm's sensitivity when adjusting the size of the *working set*. We periodically execute the *Selective Core Idling* algorithm to adjust the *working set* to match the inference throughput.

Algorithm 2 outlines the proposed algorithm for *Selective Core Idling*. It takes two inputs: the set of available cores and the number of inference tasks that are oversubscribing the CPU. Once executed, it adjusts the *working set* by selectively setting the idle states of available cores to either deep idle or active. Firstly, the algorithm processes the set of available cores to obtain the number of total

Algorithm 2 Proposed selective core idling algorithm.

Require:
- cpu_cores: List of available cores
- oversub_tasks: Number of CPU oversubscribing tasks

Ensure: Adjusted core idle states: deep idle or active.

```
1:  N ← get_total_core_count(cpu_cores)
2:  active_cores ← get_active_core_count(cpu_cores)
3:  normal_tasks ← get_assigned_task_count(cpu_cores)
4:  C_SLP_t ← N − active_cores
5:  T_t ← normal_tasks + oversub_tasks
6:  T_t ← min(N, T_t)
7:  e_t ← (N − C_SLP_t − T_t)
8:  e_t_prd ← e_t
9:  e_t_prd ← e_t_prd / N
10: if e_t_prd ≥ 0 then
11:     F(e_t_prd) ← tan(0.785 · e_t_prd)
12: else
13:     F(e_t_prd) ← arctan(1.55 · e_t_prd)
14: end if
15: e_t_corr ← N × F(e_t_prd)
16: e_t_corr ← int(e_t_corr)
17: δ_cores ← | e_t_corr|
18: if e_t_corr > 0 then
19:     put_cores_idle(δ_cores, cpu_cores)
20: else if e_t_corr < 0 then
21:     put_cores_active(δ_cores, cpu_cores)
22: end if
```

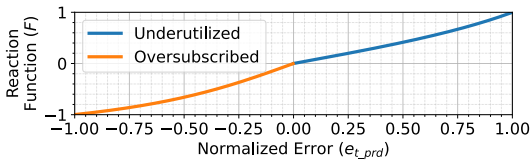


Figure 5: Behavior of the piecewise Reaction Function (F) for utilization of the CPU.

cores, the number of cores that are in the active state, and the number of inference tasks that are allocated with a dedicated core (line 1). The algorithm calculates the number of cores that are currently deep idling (line 4), as well as the total number of tasks. Here, we cap the total number of tasks at the total number of CPU cores (line 6). This is done to obtain a normalized error term (e_{t_prd}), which we are calculating next (line 9). The error term indicates the severity of CPU oversubscription of the inference tasks. We then use the error term as the input to the part of our algorithm that carries out the reaction function (line 10 to line 14). The output of the reaction function is then scaled back (line 15) and used to determine the cores to set either active or deep idle. When putting cores to deep idle, we do that in the order of most aged first. When putting cores to active, we do it in the order of least aged first. That way, we complement even-out core aging of *Task to Core Mapping* when deep idling the cores.

Reaction Function: The reaction function is designed to be independent of the number of CPU cores. It takes the normalized error term (e_{t_prd}) in Algorithm 2 (line 9) as the input. It returns a normalized output between −1 and +1. Here, a positive output indicates CPU underutilization, requiring the setting of cores from active to deep idle. A negative output indicates CPU oversubscription, requiring the setting of cores from deep idle to active. In the inference cluster, CPU oversubscription impacts the inference latency of user requests, which is a short-term effect requiring immediate action. In contrast, CPU underutilization leads to the aging of cores, which is a long-term effect since aging is a slow process. To balance this trade-off, we design the reaction function to react slower for CPU underutilization and faster for CPU oversubscription. Figure 5 illustrates the behavior of our reaction function. Algorithm 2 denotes the equation and the values we used for that (line 10 to line 14).

Table 2: Tasks modeled as inference tasks in the extended splitwise-sim [36] simulator.

Task Name	Class/Function
`finish_flow`	`Executor.finish_flow`
`finish_request`	`Executor.finish_request`
`finish_task`	`Executor.finish_task`
`submit`	`Executor.submit`
`submit_chain`	`Executor.submit_chain`
`submit_flow`	`Executor.submit_flow`
`submit_task`	`Executor.submit_task`
`alloc_memory`	`Instance.alloc_memory`
`free_memory`	`Instance.free_memory`
`start_iteration`	`ORCAInstance.start_iteration`
`flow_completion`	`Link.flow_completion`

5 Implementation

We implement our proposed technique in a simulated environment. We use splitwise-sim [36], an event-driven, high-fidelity LLM cluster simulator from Microsoft. It employs Splitwise, a state-of-the-art phase splitting LLM serving technique. Compared to vanilla request level scheduling, Splitwise increases CPU load due to the facilitation of additional tasks of phase splitting. As a result, our simulation environment enables exposing our technique to realistic CPU stress levels present in state-of-the-art cloud LLM inference clusters.

First, we extend the simulator to model the inference tasks that run on the CPU. Table 2 outlines the tasks we modeled. We model the CPU load of the executor component that facilitates the inference workflow, worker instance tasks that handle memory and iterative-level scheduling, and the tasks of interconnects. For that, we merge each class function with the APIs of a new processor subclass we implemented for the CPU. Inside the CPU class, we manage the state of CPU cores. Using the models we outlined in Section 3, we maintain core temperatures, idle states, and the shift in threshold voltage. Each class function call outlined in Table 2 invokes the `assign_core_to_cpu_task` API of the CPU class. It then provides inputs and invokes the Algorithm 1. In return, we determine a CPU core to cater for the function call. We update the idle states and temperature of the core to match the task execution. Further, we update the shift in the threshold voltage and the resulting operating frequency of the core. The execution time of the calling function in the simulator is then adjusted according to the operating frequency. In parallel to that, we periodically invoke the `adjust_sleeping_cores` API of the CPU class to conduct *Selective Core Idling*. It retrieves the system state and executes the Algorithm 2. In return, the algorithm sets the idle state of a number of CPU cores to either active or deep idle. Since the periodic execution of algorithm 2 does not add overhead to inference request latency, we use it as an opportunity to accurately calculate degraded core frequency due to aging. We assume that data is provided by the core-level aging sensors [17] with an additional overhead.

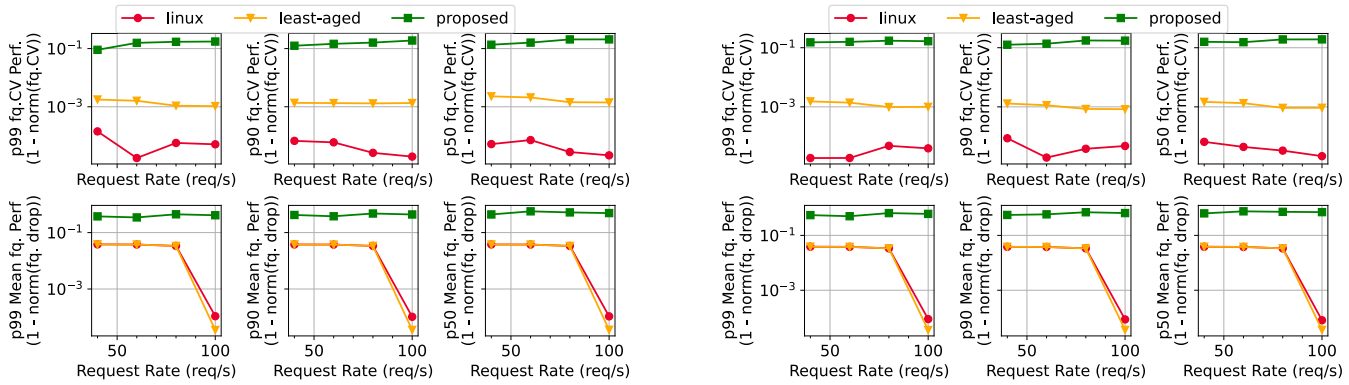
It is important to note that, in practice, the performance of our proposed technique and its implementation could depend on the underlying CPU hardware, its features, and our algorithm configurations. For instance, the selective core idling in our proposed technique is supported across CPU vendors via the CPU idle states feature [49]. Since their specific hardware implementations could differ [30, 51], the effectiveness of CPU age halting will be determined by the number of CPU components that get deactivated at the deepest sleep state [51]. In addition, the deeper sleep states introduce longer transition latencies (due to longer wake times of idle cores) [49], however, due to latencies of microsecond-scale [51], it typically incurs minimum impact to inference tasks, depending on the inference task throughput and CPU oversubscription. While selective core idling and task-to-core mapping mechanisms do not incur coordination latency in between due to their independent execution, they can exhibit latency overhead within their workflows. For instance, the implementation of the task-to-core mapping mechanism could introduce a latency overhead in algorithm execution. We reduce the impact of that through low-complexity algorithm design. Accordingly, selective core idling could introduce a communication latency overhead when retrieving data from core-level aging sensors. We reduce this impact by executing its algorithm periodically rather than executing it for each inference task scheduling event.

6 Performance Evaluation

In this section, we evaluate the performance of our proposed CPU core management for amortizing embodied carbon in LLM clusters. We provide our experimental design and setup, compare our results with state-of-the-art baselines, and analyze them in detail.

6.1 Experiment Design and Setup

We conduct evaluation experiments in the simulated environment, which we modeled and implemented with an LLM cluster simulator from Microsoft. We describe the inner details of our implementation in Section 5. We model a cluster of 22 GPU-optimized Nvidia H100 machines with 5 prompt instances and 17 token instances of phase splitting, which is an iso-throughput, power-optimized cluster design for cloud LLM inference [36]. We use it to create a realistic cloud inference cluster. Each server in the cluster runs a worker instance. For the worker instance CPU, we use CPU core counts of 40 and 80 to match public VM offerings in Azure for Nvidia H100 machines [3].



(a) Comparison of 40 VM cores

(b) Comparison of 80 VM cores

Figure 6: Comparison of managing aging effects in CPU.

6.1.1 Baselines: As discussed in section 2.1, our problem context requires extended embodied carbon amortization through an efficient CPU age management technique. However, existing works focusing on CPU age management in the cloud settings for LLM inference are limited. In that regard, we use two state-of-the-art baselines. We use ***linux*** to compare the performance of our technique with state-of-the-art inference serving systems [36], and we use ***least-aged*** to compare the performance of our technique with state-of-the-art CPU age management techniques designed for cloud settings that cater CPU tasks similar to inference tasks used in our system model. We further detail our baselines as follows:

linux: It represents existing state-of-the-art LLM inference servers that leave default Linux schedulers to manage the core-level task scheduling. To implement that, we use CPU data from an LLM inference server captured while executing inference requests [47]. Using the data, we build a probabilistic model to generate inference tasks to CPU core mappings.

least-aged *[54]:* It is an aging-aware task-serving idea proposed for cloud servers. Unlike most works, *least-aged* proposes the idea of assigning tasks away from aged cores using executed work as an aging estimate without requiring frequent CPU profiling. Although *least-aged* was designed for cloud CPU tasks in general, the task characteristics it uses for aging apply to the inference tasks that we model.

6.1.2 Workloads: We use LLM inference traces generated by Microsoft using real Azure inference data [36]. Each request in the trace is characterized by the number of input tokens and the number of output tokes generated. It does not provide the actual query that was used in the public cloud environment due to privacy requirements. For the performance metrics that we outline next, the actual query does not make an impact; rather, the execution times resulted from processing the input tokens.

6.1.3 Metrics: To measure the effects of CPU aging, we use the coefficient of variation (CV) of the distribution of frequencies among CPU cores in each inference server after the experiment. We then calculate the percentile values across the cluster. The resulting **frequency CVs** reflect how well the technique could even out the aging effects among the cores in the cluster machines. To measure the application impact, we calculate the distribution of the number of idle CPU cores in inference servers during the experiments. The resulting data reflect the impact of **CPU oversubscription** across the cluster servers.

6.2 Results and Analysis

We carry out the performance evaluation as follows. We sample a set of initial core frequencies for each inference server CPU according to the process variation model described in Section 3.2. We then replay LLM inference traces on the cluster. We conduct repeated experiments for different throughput levels of LLM inference traces for each baseline and for our proposed technique. At the end of the experiments, we calculate the degradation of initial CPU core frequencies through our metrics to evaluate how each baseline and our proposed technique managed cores across the cluster to reduce CPU aging effects and minimize the application impact. We further estimate the reduction of the cluster's yearly embodied carbon emissions resulting from CPU aging management.

Management of core aging effects: Based on our aging model described in Section 3.2, the initial frequency of each CPU core deviates from the nominal value due to process variation. Due to the execution of inference tasks, initial frequencies degrade over time, resulting in a frequency distribution across CPU cores. Figure 6 illustrates the results of that. Its subplot, figure 6a, illustrates the performance of managing the coefficient of variation (CV) of the core frequency distribution. The performance value decreases when the frequency CV increases, and vice versa. It also illustrates the performance of managing the mean frequency degradation. The performance value decreases when the mean frequency degradation increases and vice versa. Both performances are for the VM with 40 cores. Figure 6b illustrates the same for the VM core count of 80. All plots share the x-axis, which is the throughput of inference traces.

The results show that in both VM core types, the reduction of frequency variance among cores with the *least-aged* is better than in *linux*. It shows the effectiveness of assigning inference tasks

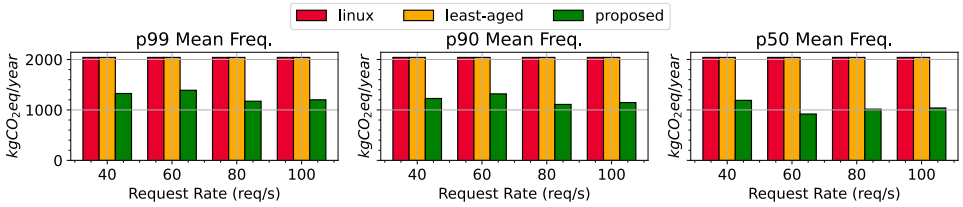
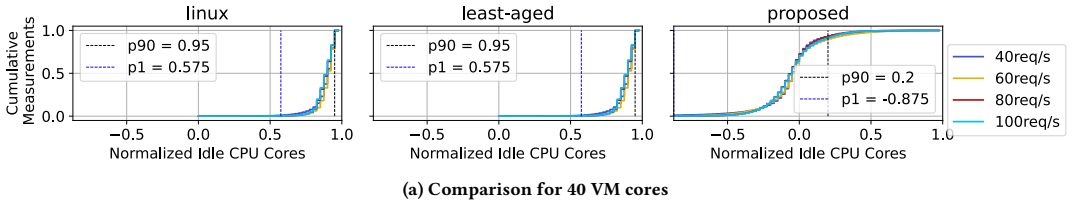
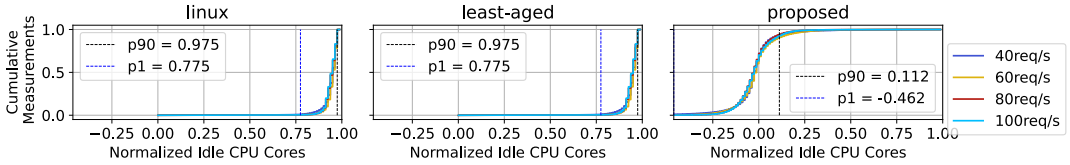


Figure 7: Comparison of estimated yearly CPU embodied carbon reduction in the cluster through management of CPU aging effects.



(a) Comparison for 40 VM cores



(b) Comparison for 80 VM cores

Figure 8: Comparison of utilization of available cores for running tasks. The x-axis in each plot denotes normalized idle CPU cores, where a negative value indicates CPU oversubscription and a positive value indicates CPU underutilization.

away from the aged cores to improve aging imbalance across cores in the *least-aged*. However, the proposed technique significantly outperforms both baselines in that. It showcases the superiority of core age even-out behavior across both *Task-to-core mapping* and *Selective Core Idling* mechanisms in the proposed technique. The results of managing mean frequency degradation performance show that both baselines exhibit quite similar performances. In Figures 6a and 6b, their frequency performances are consistent, only deviating slightly at the request rate of 100. In contrast, the proposed technique surpasses baselines across all evaluated request rates, showcasing the superior performance of its age halting. Discussed performance patterns are consistent across both VM sizes.

Reduction of yearly CPU-embodied carbon emissions: Delayed CPU aging effects allow cloud operators to extend CPU lifespan by increasing the hardware refresh lifecycle. We apply the same with our results of managing core aging effects. We take the hardware refresh cycle of a typical Linux-based LLM inference server as 3 years and its CPU embodied carbon during this lifespan as 278.3 $kgCO_2eq$ [26]. We then compare the reduction of the mean core frequency of other techniques to *linux* and estimate an increase in lifecycle extension using a linear model. Using the carbon and lifespan expansion data, we calculate yearly embodied carbon emissions for baselines and the proposed technique. Figure 7 presents our results. It shows the cluster's yearly CPU-embodied carbon emissions for the age management performance of different throughput levels.

The results show that yearly CPU-embodied carbon savings for the *least-aged* are minimal when compared to *linux*. This is due to the similar performance of the mean frequency degradation that we discussed previously. In contrast, a cluster managed with our proposed technique shows significant carbon savings in CPU embodied. When estimated with p99 mean frequency performance, our proposed method reduces yearly CPU embodied emissions in our experimental inference cluster by 37.67%. It further increases to 49.01% for p50 mean frequency performance. The observed results are due to the superiority of the mean frequency performance of our proposed technique. It showcases achieving CPU embodied carbon reduction through effective age management. Note that the advantage of carbon reductions with *least-aged* over *linux* may improve with the experiment duration. However, the goal of our experiments is to evaluate the advantage of our proposed technique, which we show within the evaluated experiments.

Application impact of aging-aware core management: Figure 8 illustrates the results of idle core availability in the cluster servers during inference task execution. The x-axis in all figures shows normalized idle CPU cores, in which a positive value indicates core underutilization and a negative value indicates core oversubscription, whereas the y-axis denotes the measurement distribution.

Table 3: Comparison of relevant works with our proposed technique.

Work	Even-Out Core Aging	Process Variation Aware	Avoid CPU Profiling	Dynamic Age-halting
Facelift'08 [44]	✓	✓		
Hyat'15 [17]	✓	✓		
Tamer'21 [39]	✓			
Shoulao'23 [27]	✓	✓		
Zhao et al.'23 [54]	✓		✓	
Our Proposed	✓	✓	✓	✓

The results show that both baselines do not incur core oversubscription but underutilize cores, yielding positive values of idle CPU cores. p1 to p90 percentiles in both baselines reside closer to 1.0, with a higher VM count increasing the closeness. In contrast, the proposed technique outperforms both baselines in CPU underutilization. Its p90 percentile is at least 77.8% better in both VM core counts. However, its negative p1 percentile indicates that the proposed technique results in CPU oversubscription. The severity of that improves with the VM core count, showing a smaller P1 value. We observe consistent idle core distributions across different inference throughput rates. Additionally, results show that p1 of the proposed technique is at least less than -0.1, which means the proposed technique maintains the CPU oversubscription below 10%.

In summary, we observe the core aging even-out behavior of our proposed technique surpassing baselines in reducing frequency CVs. Alongside, age-halting behavior in our technique showcases its superiority in delaying mean frequency degradation. Both frequency CV and mean frequency performance are metrics of CPU aging effects in our system model. We then estimate yearly CPU embodied emissions in the experimental inference cluster based on the mean frequency performance. Results highlight the efficacy of our proposed method to reduce CPU embodied through managing its aging effects. In return, our proposed method shows CPU oversubscription, which can impact the service quality of the inference tasks. Yet, results show that our proposed technique is able to maintain its severity.

7 Related Work

The environmental impact of embodied carbon in growing LLM inference clusters has caught attention in recent years [7, 10, 14, 26, 33]. As an early research area, these works model embodied carbon in LLM inference and advocate for potential directions to reduce that [26, 33, 53]. Further, some outline CPU GPU asymmetric optimization opportunities of heterogeneous energy, performance, and inference application patterns [26, 53] and accounting carbon footprint for a given inference request on specific hardware settings [15]. In contrast, some works propose system-level techniques to actively optimize embodied carbon. These include controlling LLM token generation [24] and disaggregation of specific computing onto older hardware [41].

Building on studies of embodied carbon optimization in CPU GPU asymmetric lifetimes, we explore system-level solutions for fine-grain CPU-aging management by leveraging request-level patterns in cloud LLM inference clusters.

A plethora of works investigates mitigating CPU aging effects through workload management [17, 27, 39, 44, 54]. Their predominant approach is even-outing tasks across the cores to reduce uneven aging. These include utilizing CPU profiling [17, 27, 39, 44] and addressing manufacturing process variations in CPU [17, 27, 44]. However, not many consider the efficacy of the proposed techniques in cloud settings. For example, conducting CPU profiling in clouds with large server fleets is difficult. Nevertheless, few recent works consider cloud-efficient techniques, such as workload management at the resource management level, to reduce the severe exercising of specific cores [54]. In addition to even-outing aging, age halting is an efficient approach to slow down CPU aging. Age halting has been used in the literature to leverage dark silicon in CPUs for age management [17]. However, their age-halting is static since the age-halting adjustments are only done after a relatively longer epoch.

In contrast, we study CPU age management for cloud LLM inference. Further, our study encompasses both even-outing aging and dynamic age-halting.

8 Conclusions and Future Work

Given the growing imperative for sustainable growth of cloud LLM inference clusters, our work focuses on mitigating the accumulation of embodied carbon that mostly concentrates on inference server CPU. We propose an aging-aware CPU core management technique, showcasing its potential to extend the cluster's embodied carbon amortization through increasing CPU lifespan. Exploiting CPU underutilization patterns that we uncover, the proposed technique not only even-out silicon aging across cores but also harnesses the opportunities of age halting using core deep idling. Our empirical simulations demonstrate the superiority of the proposed technique over existing methods in reducing the cluster's yearly embodied emissions with a minimum impact on the inference service quality. Our technique enables LLM inference to reduce embodied carbon through the CPU and improve performance with the GPU via the CPU GPU's asymmetric lifetime.

In future work, we plan to implement the proposed technique with cloud resource management middleware and leverage runtime core telemetry data to improve the core aging estimation. Further, selective core idling in the proposed technique leverages CPU usage patterns of LLM inference. It can be advanced to deployments that co-locate other services as a promising direction for future research. Similarly, another important direction is to investigate the aging heterogeneity across different server CPUs, complementing each node's existing core aging management.

References

[1] Mohsen Ansari, Sepideh Safari, Amir Yeganeh-Khaksar, Roozbeh Siyadatzadeh, Pourya Gohari-Nazari, Heba Khdr, Muhammad Shafique, Jörg Henkel, and Alireza Ejlali. 2023. ATLAS: Aging-Aware Task Replication for Multicore Safety-Critical Systems. In *Proceedings of the 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 223–234.

[2] AWS. 2025. Deploying Multiple Large Language Models with NVIDIA Triton Server and vLLM. Retrieved May 4, 2025 from https://awslabs.github.io/data-on-eks/docs/gen-ai/inference/GPUs/vLLM-NVIDIATritonServer

[3] Azure. 2024. NCads H100 v5-series. Retrieved May 4, 2025 from https://learn.microsoft.com/en-us/azure/virtual-machines/ncads-h100-v5

[4] Azure. 2024. NCasT4_v3 sizes series. Retrieved May 4, 2025 from https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/gpu-accelerated/ncast4v3-series?tabs=sizebasic

[5] Sally Beatty. 2024. Microsoft builds first datacenters with wood to slash carbon emissions. Retrieved May 4, 2025 from https://news.microsoft.com/source/features/sustainability/microsoft-builds-first-datacenters-with-wood-to-slash-carbon-emissions/

[6] Daniel S. Berger, Fiodar Kazhamiaka, Esha Choukse, Íñigo Goiri, Celine Irvene, Pulkit Misra, Alok Kumbhare, Rodrigo Fonseca, and Ricardo Bianchini. 2023. Research Avenues Towards Net-Zero Cloud Platforms. In *Proceedings of the 1st Workshop on NetZero Carbon Computing (NetZero)*.

[7] Ricardo Bianchini, Christian Belady, and Anand Sivasubramaniam. 2024. Data Center Power and Energy Management: Past, Present, and Future. *IEEE Micro* 44, 5 (2024), 30–36.

[8] Jairus Bowne. 2024. Using Large Language Models in Learning and Teaching. Retrieved May 4, 2025 from https://biomedicalsciences.unimelb.edu.au/study/dlh/assets/documents/large-language-models-in-education/llms-in-education

[9] Microsoft Corporation. 2022. The role of embodied carbon in cloud emissions. Retrieved May 4, 2025 from https://go.microsoft.com/fwlink/p/?linkid=2233506

[10] Crusoe. 2023. How Together And Crusoe Are Reducing The Carbon Impact Of Generative AI. Retrieved May 4, 2025 from https://crusoe.ai/blog/crusoe-together-reducing-carbon-impact-of-generative-ai/

[11] Crusoe. 2024. Crusoe to Build Initial 200 MW AI Data Center With Plans to Expand at 1.2 GW Lancium Clean Campus. Retrieved May 4, 2025 from https://crusoe.ai/newsroom/crusoe-200mw-ai-data-center/

[12] Sheila Dang. 2024. Musk's xAI plans massive expansion of AI supercomputer in Memphis. Retrieved May 4, 2025 from https://www.reuters.com/technology/artificial-intelligence/musks-xai-plans-massive-expansion-ai-supercomputer-memphis-2024-12-04/

[13] Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. 2021. Silent Data Corruptions at Scale. *arXiv preprint arXiv:2102.11245* (2021).

[14] Ahmad Faiz, Sotaro Kaneda, Ruhan Wang, Rita Chukwunyere Osi, Prateek Sharma, Fan Chen, and Lei Jiang. 2024. LLMCarbon: Modeling the End-to-End Carbon Footprint of Large Language Models. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

[15] Zhenxiao Fu, Fan Chen, Shan Zhou, Haitong Li, and Lei Jiang. 2024. LLMCO2: Advancing Accurate Carbon Footprint Prediction for LLM Inferences. *arXiv preprint arXiv:2410.02950* (2024).

[16] GitHub. 2021. Introducing GitHub Copilot: your AI pair programmer. Retrieved May 4, 2025 from https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

[17] Dennis Gnad, Muhammad Shafique, Florian Kriebel, Semeen Rehman, Duo Sun, and Jörg Henkel. 2015. Hayat: harnessing dark silicon and variability for aging deceleration and balancing. In *Proceedings of the 52nd Annual Design Automation Conference*. Article 180.

[18] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 784–799.

[19] Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. 2021. Cores that don't count. In *Proceedings of the 21st Workshop on Hot Topics in Operating Systems (HotOS 21)*. 9–16.

[20] IEA. 2023. International Energy Agency's report on Low-emissions sources of electricity. Retrieved May 4, 2025 from https://www.iea.org/reports/low-emissions-sources-of-electricity

[21] Intel. 2024. How to Find Compatible Motherboards for the Intel® Xeon® Processor Family? Retrieved May 4, 2025 from https://www.intel.com/content/www/us/en/support/articles/000057630/processors.html

[22] Shixin Ji, Zhuoping Yang, Xingzhen Chen, Stephen Cahoon, Jingtong Hu, Yiyu Shi, Alex K. Jones, and Peipei Zhou. 2024. SCARIF: Towards Carbon Modeling of Cloud Servers with Accelerators. In *Proceedings of 2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 496–501.

[23] Lenovo. 2025. ThinkSystem SR650 Maintenance Manual. Retrieved May 4, 2025 from https://pubs.lenovo.com/sr650/sr650_maintenance_manual.pdf

[24] Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. 2024. Sprout: Green Generative AI with Carbon-Efficient LLM Inference. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 21799–21813.

[25] Yueying Li, Zhanqiu Hu, Esha Choukse, Rodrigo Fonseca, G. Edward Suh, and Udit Gupta. 2025. EcoServe: Designing Carbon-Aware AI Inference Systems. *arXiv preprint arXiv:2502.05043* (2025).

[26] Yueying Lisa Li, Omer Graif, and Udit Gupta. 2024. Towards Carbon-efficient LLM Life Cycle. In *Proceedings of the 3rd Workshop on Sustainable Computer Systems*.

[27] Arthur F. Lorenzon, Guilherme Korol, Marcelo Brandalero, and Antonio Carlos Schneider Beck. 2023. Harnessing the Effects of Process Variability to Mitigate Aging in Cloud Servers. In *Proceedings of the 2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 1–6.

[28] Jialun Lyu, Marisa You, Celine Irvene, Mark Jung, Tyler Narmore, Jacob Shapiro, Luke Marshall, Savyasachi Samal, Ioannis Manousakis, Lisa Hsu, Preetha Subbarayalu, Ashish Raniwala, Brijesh Warrier, Ricardo Bianchini, Bianca Schroeder, and Daniel S. Berger. 2023. Hyrax: Fail-in-Place Server Operation in Cloud Platforms. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 287–304.

[29] Diptyaroop Maji, Noman Bashir, David Irwin, Prashant Shenoy, and Ramesh K. Sitaraman. 2024. Untangling Carbon-free Energy Attribution and Carbon Intensity Estimation for Carbon-aware Computing. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems*. 580–588.

[30] Kevin Mayo, Sylvester Rajasekaran, Igor Pasichnyk, and Michael Senizaiz. 2018. High Performance Computing (HPC) Tuning Guide. Retrieved May 4, 2025 from https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/tuning-guides/58479_amd-epyc-9005-tg-hpc.pdf

[31] Rick Merritt. 2023. Why GPUs Are Great for AI. Retrieved May 4, 2025 from https://blogs.nvidia.com/blog/why-gpus-are-great-for-ai/

[32] Iraj Moghaddasi, Arash Fouman, Mostafa E. Salehi, and Mehdi Kargahi. 2019. Instruction-Level NBTI Stress Estimation and Its Application in Runtime Aging Prediction for Embedded Processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38 (2019), 1427–1437.

[33] Sophia Nguyen, Beihao Zhou, Yi Ding, and Sihang Liu. 2025. Towards Sustainable Large Language Model Serving. *SIGENERGY Energy Inform. Rev.* 4, 5 (2025), 134–140.

[34] LA Office of the governor. 2024. Landry Announces Meta Selects North Louisiana as Site of $10 Billion Artificial Intelligence Optimized Data Center. Retrieved May 4, 2025 from https://gov.louisiana.gov/news/4697

[35] OpenAI. 2022. Introducing ChatGPT. Retrieved May 4, 2025 from https://openai.com/index/chatgpt/

[36] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 118–132.

[37] GHG Protocol. 2024. Greenhouse Gas Protocol. Retrieved May 4, 2025 from https://ghgprotocol.org/about-us

[38] Bharathwaj Raghunathan, Yatish Turakhia, Siddharth Garg, and Diana Marculescu. 2013. Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors. In *Proceedings of the 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 39–44.

[39] Faezeh Sadat Saadatmand, Nezam Rohbani, Farshad Baharvand, and Hamed Farbeh. 2021. TAMER: an adaptive task allocation method for aging reduction in multi-core embedded real-time systems. *The Journal of Supercomputing* 77 (2021), 1939–1957.

[40] Philipp Schmid, Omar Sanseviero, Pedro Cuenca, and Lewis Tunstall. 2023. Llama 2 is here - get it on Hugging Face. Retrieved May 4, 2025 from https://huggingface.co/blog/llama2

[41] Tianyao Shi, Yanran Wu, Sihang Liu, and Yi Ding. 2024. GreenLLM: Disaggregating Large Language Model Serving on Heterogeneous GPUs for Lower Carbon Emissions. *arXiv preprint arXiv:2412.20322* (2024).

[42] Fumiyoshi Shoji, Shuji Matsui, Mitsuo Okamoto, Fumichika Sueyasu, Toshiyuki Tsukamoto, Atsuya Uno, and Keiji Yamamoto. 2015. Long term failure analysis of 10 peta-scale supercomputer. *HPC in Asia Poster, ISC* (2015).

[43] The Financial Times. 2024. OpenAI targets 1bn users in next phase of growth. Retrieved May 4, 2025 from https://www.ft.com/content/e91cb018-873c-4388-84c0-46e9f82146b4

[44] Abhishek Tiwari and Josep Torrellas. 2008. Facelift: Hiding and slowing down aging in multicores. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*. 129–140.

[45] Amanda Tomlinson and George Porter. 2023. Something Old, Something New: Extending the Life of CPUs in Datacenters. *SIGENERGY Energy Informatics Review* 3, 3 (2023), 59–63.

[46] Shashank Verma and Neal Vaidya. 2023. Mastering LLM Techniques: Inference Optimization. Retrieved May 4, 2025 from https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

[47] Grant Wilkins, Srinivasan Keshav, and Richard Mortier. 2024. Hybrid Heterogeneous Clusters Can Lower the Energy Consumption of LLM Inference Workloads. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems*. 506–513.

[48] Rafael J. Wysocki. 2018. CPU Idle Time Management. Retrieved May 4, 2025 from https://www.kernel.org/doc/html/v5.4/admin-guide/pm/cpuidle.html

[49] Rafael J. Wysocki. 2018. CPU Idle Time Management. Retrieved May 4, 2025 from https://docs.kernel.org/admin-guide/pm/cpuidle.html

[50] Jawad Haj Yahya, Jeremie S. Kim, A. Giray Yağlıkçı, Jisung Park, Efraim Rotem, Yanos Sazeides, and Onur Mutlu. 2022. DarkGates: A Hybrid Power-Gating Architecture to Mitigate the Performance Impact of Dark-Silicon in High Performance Processors. In *Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1170–1183.

[51] Jawad Haj Yahya, Haris Volos, Davide B. Bartolini, Georgia Antoniou, Jeremie S. Kim, Zhe Wang, Kleovoulos Kalaitzidis, Tom Rollet, Zhirui Chen, Ye Geng, Onur Mutlu, and Yiannakis Sazeides. 2022. AgileWatts: An Energy-Efficient CPU Core Idle-State Architecture for Latency-Sensitive Server Applications. In *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 835–850.

[52] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 521–538.

[53] Xiaoyang Zhang, Yijie Yang, and Dan Wang. 2024. Spatial-Temporal Embodied Carbon Models for the Embodied Carbon Accounting of Computer Systems. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems*. 464–471.

[54] Jiechen Zhao, Katie Lim, Thomas Anderson, and Natalie Enright Jerger. 2023. The Case of Unsustainable CPU Affinity. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems (HotCarbon 23)*. Article 1.