Special Session: End-To-End Carbon Footprint Assessment and Modeling of Deep Learning

Ahmad Faiz Lei Jiang Fan Chen
Department of Intelligent Systems Engineering, Indiana University, Bloomington, IN, USA
{afaiz, jiang60, fc7}@iu.edu

***Abstract*—Large language models (LLMs) have been widely adopted across various applications. However, their intensive computation and energy use have raised concerns about environmental sustainability. Our open-source tool, *LLMCarbon*, is the first comprehensive model for estimating the carbon footprint of LLMs before training. This paper reviews the *LLMCarbon* model, details its hardware efficiency model, presents a case study on training carbon footprints, and discusses recent research inspired by *LLMCarbon* and future directions.**

***Index Terms*—Large Language Models, Sustainability**

I. Introduction

Recent advancements in deep learning, epitomized by the proliferation of large language models (LLMs) [1]–[19], have garnered significant attention from both industry and academia, leading to widespread adoption across various domains. However, their intensive compute and corresponding energy consumption have raised concerns about their negative impact on environmental sustainability. Therefore, it is imperative to assess their carbon footprint before commencing the resource-intensive training process. Specifically, the carbon footprint of an LLM comprises of two main components: the operational footprint ($CO2_{oper}$) from runtime hardware energy consumption and the embodied footprint ($CO2_{emb}$) from hardware manufacturing, transportation, and disposal. Predicting $CO2_{oper}$ requires considering LLM model parameters, hardware efficiency, and data center energy efficiency. Calculating $CO2_{emb}$ involves chip area and Carbon Per unit Area (CPA), based on semiconductor fabrication parameters such as yield, manufacturing energy consumption, chemical emissions, and raw material sourcing. The total carbon footprint ($CO2_{eq}$) is the sum of $CO2_{oper}$ and $CO2_{emb}$. Additionally, a comprehensive assessment should include test loss, training duration, and inference latency to fully evaluate LLM performance and environmental impact.

There has long been a lack of comprehensive LLM carbon footprint estimators. Existing tools either report operational emissions post-training [20], or are designed for convolutional neural network [21]. Our open-source tool, *LLMCarbon* [22], is the first end-to-end carbon footprint projection model for both dense and sparse LLMs, and it has garnered significant attention since its release. This paper first reviews the *LLMCarbon* model, then explains the critical hardware efficiency model within *LLMCarbon*. We also present a case study on scaling training carbon footprints and discuss recent work inspired by *LLMCarbon* and future research directions.

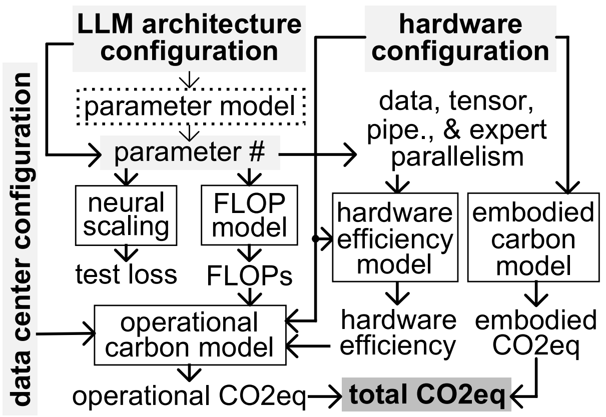


Fig. 1. Overview of LLMCarbon [22]. It calculates the total carbon footprint by processing LLM configuration parameters, hardware configurations, and data center specifications through specialized modules.

II. Overview of LLMCarbon

As shown in Figure 1, the inputs to LLMCarbon encompass the LLM algorithmic model architecture, hardware configuration, and data center specifications. These inputs undergo processing through a series of modules to generate the final LLM carbon footprint output, as detailed below.

First, the parameter model calculates the total parameter count from the LLM configuration or directly accepts its value as input. LLMCarbon uses the Chinchilla Neural Scaling Law [7] to estimate test loss across models, sizes, and datasets. The FLOP model calculates the total floating-point operations (FLOPs) required using the parameter count and dataset size. For sparse MoE models, the base dense model's parameters are used for FLOP calculation. Based on the parameter count, LLMCarbon determines the required number of compute devices and optimal parallelism configurations [23], [24] .

Second, LLMCarbon incorporates a hardware efficiency model to estimate LLM processing efficiency, which depends on tensor, pipeline, data, and expert parallelism. The number of devices n required for optimal efficiency is calculated from the parallelization degrees as $n = t \cdot p \cdot d$ where t, p, d are the tensor, pipeline, and data parallel degrees. Optimal parallel settings are derived from [23] for dense models and from [24] for sparse mixture-of-experts (MoE) models. LLMCarbon uses a regression model to estimate efficiency based on the number of compute devices, parallel settings, and total parameters.

Finally, incorporating data center details, hardware efficiency, and FLOP count, LLMCarbon computes the total operational carbon footprint. The embodied carbon footprint is calculated by summing the emissions of individual chip components. The total carbon footprint is the sum of operational and embodied footprints. LLMCarbon results differ from Google's published LLM carbon footprints by $\leq 8.2\%$.

III. Hardware efficiency model in LLMCarbon

In this section, we elaborate on the critical hardware efficiency model in LLMCarbon, providing results in addition to those presented in [22].

A. Hardware Efficiency vs. Model Parallelism Size

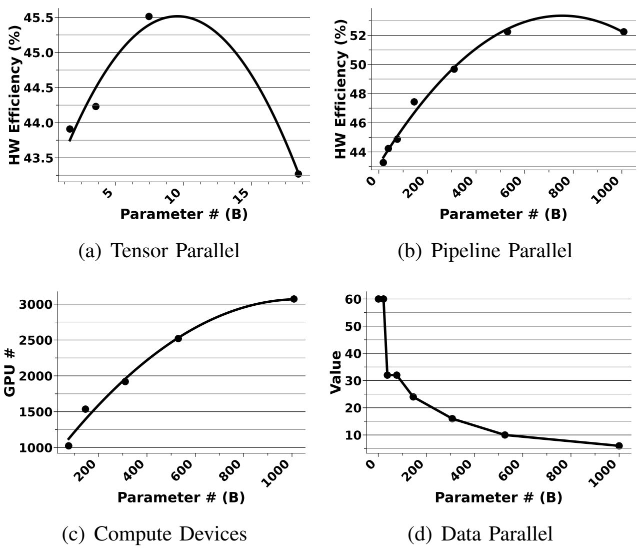
Figure 2(a) illustrates hardware efficiency with increasing tensor parallel size in a single multi-GPU server. Within a single multi-GPU server node, hardware efficiency is primarily determined by the size of matrix multiplications (GEMMs) and the communication cost over high-bandwidth GPU interconnects. To avoid the high cost of all-reduce communication across inter-server links, the tensor model parallel size is increased to z when using a z device server [23]. For single-node models, the data parallel size is set to 32. Tensor model parallelism splits the matrix multiplications of the MLP and self-attention, requiring two all-reduce communications per layer for each forward and backward pass. As tensor parallelism increases, the size of GEMMs is reduced, lowering compute efficiency while increasing all-reduce costs between tensor parallel replicas.

Figure 2(b) shows hardware efficiency scaling super-linearly with increasing parameter size using larger pipeline parallel sizes. More specifically, further scaling of model sizes across multi-GPU nodes is achieved using pipeline parallelism. With the tensor parallel degree fixed at z, scaling involves adjusting the pipeline degree and the number of GPUs. Larger model sizes improve GPU compute utilization due to larger GEMMs without significantly increasing communication costs. Pipeline parallelism requires point-to-point communication between GPUs on two servers, which can bottleneck performance. Combining tensor and pipeline parallelism facilitates parallel communication between the same tensor ranks, significantly reducing cross-node communication costs.

B. Hardware Efficiency vs. Number of Compute Devices

Figure 2(c) illustrates the scaling of compute devices as a function of total model parameters using an A100 80GB GPU. Efficiency for other GPU types is scaled by their relative performance factors compared to the A100 benchmarks. LLMCarbon adopts settings from [23] to estimate the number of compute devices, n, required for optimal throughput, where n for optimal efficiency (eff_n) is calculated as $n = t \cdot p \cdot d$. If the number of compute devices deviates from n, hardware efficiency decreases. The efficiency with *re* devices (eff_{re}) can be calculated using Equation 1, where $\gamma_0 \sim \gamma_2$ are fitting constants, eff_n represents the highest achievable efficiency, and n denotes the number of devices required to achieve eff_n.

$$\mathit{eff}_{re} = \begin{cases} \gamma_0 \cdot \frac{re}{n} \cdot \mathit{eff}_n & re < n \\ \gamma_1 \cdot \frac{n}{re} \cdot \mathit{eff}_n + \gamma_2 & re > n \end{cases} \qquad (1)$$



(a) Tensor Parallel (b) Pipeline Parallel

(c) Compute Devices (d) Data Parallel

Fig. 2. Hardware efficiency model in LLMCarbon. (a) Efficiency scaling with increasing tensor size within a single multi-GPU server; (b) Efficiency scaling between inter-server nodes with pipeline parallelism using point-to-point communication; (c) Estimated GPUs required for optimal throughput with increasing model size; (d) Data parallel size based on total compute devices and model parallel size.

For a new LLM, the optimal parallel degrees are derived from the estimated number of compute devices and model parallel size. The final data parallel size for each model parallel replica is calculated as $d = \frac{t \cdot p}{n}$. Figure 2(d) illustrates the calculated data parallel size when scaling large models that require cross-node communications. Increasing d to match n may not be feasible for all models, as the training memory footprint could exceed the capacity of a single accelerator. In such cases, the number of devices used will be less than the estimated optimal number, resulting in decreased hardware efficiency, which can be estimated using Equation 1.

C. Carbon Footprint Throughout the LLM Lifecycle

Various Hardware Components. LLMCarbon estimates the total training time for LLM processing by considering GPU efficiency, the number of devices, and total FLOPs required. A hardware unit comprises various components, including CPUs, LLM computing devices, memory modules, SSDs, and other peripherals. The total energy consumed during training is calculated by multiplying each component's peak power usage by the training duration and summing these values. This approach estimates the energy directly consumed by LLM processing, excluding any data center overhead.

Data Center Configuration. LLMCarbon incorporates auxiliary energy overhead of a data center by multiplying the total hardware energy consumption by the data center's Power Usage Effectiveness (PUE) [25], which is the ratio of the total energy consumption of the data center to the energy consumed solely by LLM computing hardware. The final operational carbon emission is determined by multiplying the data center's energy consumption by its carbon intensity, a comprehensive metric that evaluates the environmental impact of the data center's energy use by considering the proportion of renewable, carbon-free energy utilized.

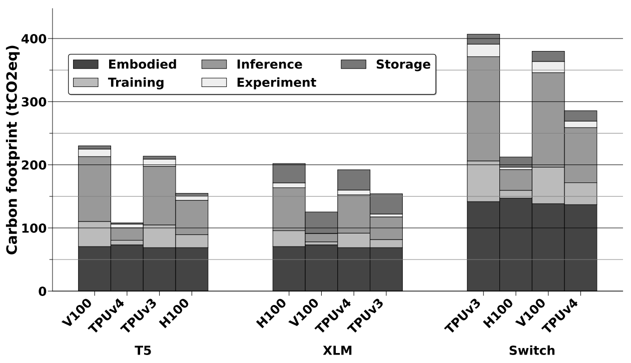


Fig. 3. Total carbon footprint of dense and sparse MoE LLM models trained on various computing devices.

Total Carbon Footprint. LLMCarbon adopts the lifecycle ratio—covering training, inference, experimentation, and storage—from [20]. Figure 3 presents the carbon footprint of three LLMs trained with various hardware devices, with hardware efficiency ranging from 39% to 19.7%. As shown, the embodied carbon from hardware manufacturing significantly contributes to 24% to 35% of the overall carbon footprint. Moreover, the embodied carbon is relatively consistent across models, primarily due to components like SSDs and DRAM. In contrast, operational carbon can vary significantly between devices for the same model. Notably, the operational carbon footprint of LLMs is reduced by 71% and 41% when using ML-specific accelerators like the H100 and TPUv4 respectively. Additionally, optimal parallel settings identified by LLMCarbon result in a significant reduction of 16% to 39% in operational carbon emissions for the three LLMs. As the gap between operational and embodied carbon emissions widens in the near future [20], using power-efficient ML-specific hardware along with optimal parallel settings to achieve high hardware efficiency offers a practical approach to mitigate the operational carbon overhead in an LLM's lifecycle.

IV. Case Study: Carbon Footprint Scaling

In this section, we conduct a case study using LLMCarbon to estimate the test loss and total training carbon footprint for mainstream LLM models, as shown in Figure 4. We provide a detailed explanation of these experiments, which serve as both a critical goal and a practical application. This enables industry LLM providers and average users to estimate the carbon footprint of LLMs before training, allowing them to make informed trade-offs and adjust their choices accordingly.

LLM Models and Configuration. LLMCarbon studies the test loss versus training carbon footprint for existing dense and sparse MoE LLM models. The dense models include T5 [1], GPT3 [2], XLM [3], Noor [4], PaLM [5], Gopher [6], Chinchilla [7], LaMDA [8], Jurassic-1 [9], MT-NLG [10], Bloom [11], YaLM [12], and GLM [13]. The sparse models considered are GShard [14], Switch [15], GLaM [16], FBMoE [17], ST-MoE [18], and PR-MoE [19]. We assume that these LLMs, each with parameter size P, are trained on the same dataset size D, on the same hardware infrastructure with V100 GPUs (node size of 8 GPUs), and in the same data center with a PUE of 1.1 and a carbon intensity of $0.431 CO2_{eq}/KWh$. Optimal parallel settings and the number of compute devices are assumed for training the listed models.

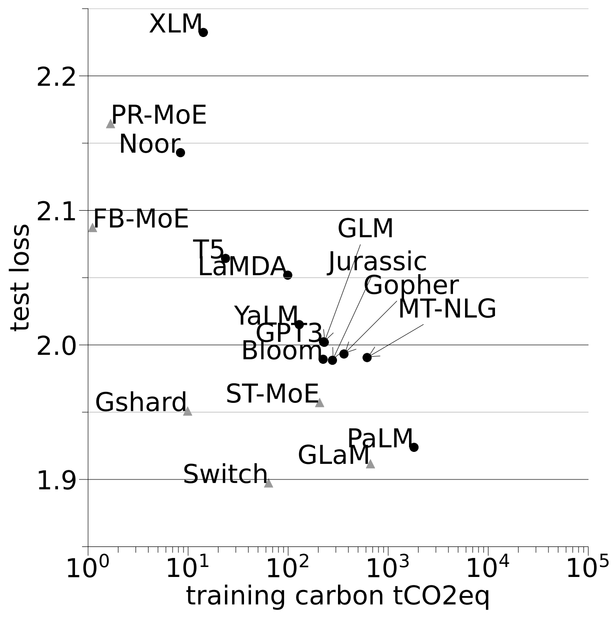


Fig. 4. Training carbon footprint scaling of dense and sparse MoE models with their corresponding estimated test loss.

Test Loss Estimation. To compute the expected test loss, LLMCarbon uses Chinchilla neural scaling laws [7], with fitting constants α=0.34, β=0.28, A=406.4, B=410.7, and E=1.69 for Equation 2. The scaling law indicates that the test loss L consists of an irreducible term E and a reducible term that diminishes with the scaling of parameters P and dataset size D. For a MoE LLM with size P, the expected test loss mirrors that of a dense counterpart with parameter size $P/8$ [19] Consequently, for MoE LLMs, the parameter size P is effectively reduced to $P/8$ in Equation 2.

$$L(P, D) = \frac{A}{P^{\alpha}} + \frac{B}{D^{\beta}} + E \tag{2}$$

Hardware Configuration. The number of compute devices used to train the new LLMs is given by $n = t.p.d$ where t, p, d are the optimal sizes of tensor, pipeline, and data parallel respectively. LLMCarbon employs polynomial regression models, as detailed in Section III, to estimate hardware efficiency based on the total parameter size, which determines the degrees of model and data parallelism. For models that fit on a single GPU server node, hardware efficiency using tensor parallelism is estimated from 2(a). For models exceeding a single GPU server's capacity, hardware efficiency using pipeline parallelism is derived from Figure 2(b). The total number of V100 GPUs required is determined from Figure 2(c), and the final data parallel ranks are calculated as $d = \frac{n}{t \cdot p}$.

Insights. Neural Scaling Law [7] indicates that an LLM with more parameters trained on more tokens can achieve a lower test loss. However, this improvement requires increased computational resources, leading to a larger carbon footprint. The choice between dense and sparse MoE LLMs significantly impacts the trade-off between test loss and training carbon footprint. MoE LLMs are more efficient, as their Pareto front lies closer to the origin in the test loss-carbon footprint plane,

TABLE I
EMBODIED CARBON COMPARISON BETWEEN [22] AND [27].

	Min.	20th percentile	Median	80th percentile	Max.
LLMCarbon [22]			0.63		
STEC-CS [27]	0.61	0.62	0.64	0.66	0.67
Difference (%)	3.17%	1.59%	1.59%	4.76%	6.35%

meaning they can achieve lower test loss with the same carbon footprint compared to dense models. This superior performance of MoE LLMs is due to their architecture, which allows for more efficient parameter utilization and reduced test loss with the same operational footprint.

V. DISCUSSION AND FUTURE WORK

In this section, we discuss several subsequent works [26]–[28] since the release of LLMCarbon and explore potential research directions inspired by these studies. Integrating these insights moving forward can enhance the accuracy and inclusively of LLM carbon footprint modeling and management.

Improve Hardware Modeling. LLMCarbon's estimation of the operational carbon footprint relies on training data from high-end hardware used in LLM training. The model's hardware efficiency is based on data from power-intensive GPUs. However, LLMCarbon's throughput and training time estimations do not account for hardware efficiency during the loading of model weights and preliminary computations. Recent work, OpenCarbonEval [26] introduces a dynamic throughput modeling approach to more accurately capture hardware fluctuations during the training process.

Embodied Carbon Modeling. LLMCarbon employs a bottom-up modeling approach to calculate total embodied carbon by assessing the footprint of chips within specific hardware units, using data from annual life-cycle analysis reports provided by hardware manufacturers, detailing carbon emissions for various product classes (e.g., 16nm TSMC CPU). In contrast, recent work [27] highlights the variability in energy consumption during hardware manufacturing, influenced by factors such as location and season. A Spatial-Temporal Embodied Carbon Model (STEC) is proposed to capture energy consumption data at granular levels, including day, season, and year. Table I illustrates the differences in embodied emissions for the Meta XLM between STEC and LLMCarbon. STEC measures the minimum, 20th percentile, median, 80th percentile, and maximum embodied carbon emissions for hardware manufacturing by optimizing location and time period. The 1.59% difference between STEC and LLMCarbon is due to limited spatial-temporal options in high-end LLM training hardware manufacturing.

Beyond Sustainability. While LLMCarbon focuses on the energy costs and potential environmental impacts of LLMs, recent work [28] examines the cost and energy usage from a collective control perspective. This study includes a qualitative analysis of the ethical implications and future directions towards equitable access to LLMs.

ACKNOWLEDGMENT

This work was supported in part by NSF CCF-2105972 and NSF CAREER AWARD CNS-2143120.

REFERENCES

[1] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.
[2] T. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.
[3] A. Conneau *et al.*, "Unsupervised cross-lingual representation learning at scale," in *Annual Meeting of the Association for Computational Linguistics*, pp. 8440–8451, July 2020.
[4] I. Lakim *et al.*, "A holistic assessment of the carbon footprint of noor, a very large Arabic language model," in *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pp. 84–94, May 2022.
[5] A. Chowdhery *et al.*, "Palm: Scaling language modeling with pathways," *arXiv:2204.02311*, 2022.
[6] J. W. Rae *et al.*, "Scaling language models: Methods, analysis & insights from training gopher," *arXiv:2112.11446*, 2021.
[7] J. Hoffmann *et al.*, "Training compute-optimal large language models," *arXiv:2203.15556*, 2022.
[8] R. Thoppilan *et al.*, "Lamda: Language models for dialog applications," *arXiv:2201.08239*, 2022.
[9] O. Lieber *et al.*, "Jurassic-1: Technical details and evaluation," *White Paper. AI21 Labs*, vol. 1, 2021.
[10] S. Smith *et al.*, "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model," *arXiv:2201.11990*, 2022.
[11] T. L. Scao *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," *arXiv:2211.05100*, 2022.
[12] Yandex, "Yalm 100b." https://github.com/yandex/YaLM-100B, 2022.
[13] A. Zeng *et al.*, "GLM-130b: An open bilingual pre-trained model," in *International Conference on Learning Representations*, 2023.
[14] D. Lepikhin *et al.*, "Gshard: Scaling giant models with conditional computation and automatic sharding," in *International Conference on Learning Representations*, 2021.
[15] W. Fedus *et al.*, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 5232–5270, 2022.
[16] N. Du *et al.*, "Glam: Efficient scaling of language models with mixture-of-experts," in *International Conference on Machine Learning*, pp. 5547–5569, PMLR, 2022.
[17] M. Artetxe *et al.*, "Efficient large scale language modeling with mixtures of experts," *arXiv:2112.10684*, 2021.
[18] B. Zoph *et al.*, "St-moe: Designing stable and transferable sparse expert models," *arXiv:2202.08906*, 2022.
[19] S. Rajbhandari *et al.*, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International Conference on Machine Learning*, pp. 18332–18346, 2022.
[20] C.-J. Wu *et al.*, "Sustainable ai: Environmental implications, challenges and opportunities," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813, 2022.
[21] A. Lacoste *et al.*, "Quantifying the carbon emissions of machine learning," *arXiv:1910.09700*, 2019.
[22] A. Faiz *et al.*, "Llmcarbon: Modeling the end-to-end carbon footprint of large language models," in *International Conference on Learning Representations*, 2024.
[23] D. Narayanan *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *ACM International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.
[24] X. Chen *et al.*, "Pipeline moe: A flexible moe implementation with pipeline parallelism," *arXiv:2304.11414*, 2023.
[25] P. Henderson *et al.*, "Towards the systematic reporting of the energy and carbon footprints of machine learning," *Journal of Machine Learning Research*, 2020.
[26] Z. Yu, "Opencarboneval: A unified carbon emission estimation framework in large-scale ai models," *arXiv:2405.12843*, 2024.
[27] X. Zhang *et al.*, "Spatial-temporal embodied carbon models for the embodied carbon accounting of computer systems," in *International Conference on Future and Sustainable Energy Systems*, 2024.
[28] V. Sathish, "Llempower: Understanding disparities in the control and access of large language models," *arXiv:2404.09356*, 2024.