$LLMCO_2$: Advancing Accurate Carbon Footprint Prediction for LLM Inferences

ZHENXIAO FU, Indiana University, USA
FAN CHEN, Indiana University, USA
SHAN ZHOU, Purdue University, USA
HAITONG LI, Purdue University, USA
LEI JIANG, Indiana University, USA

Throughout its lifecycle, an LLM incurs significantly higher carbon emissions during inference than training. Inference requests vary in batch size, prompt length, and token generation, while cloud providers deploy heterogeneous GPU configurations to meet diverse service-level objectives. Unlike training, inference exhibits lower and highly variable hardware utilization, making equation-based carbon models unreliable. Existing network-based estimators lack accuracy, as they fail to account for the distinct prefill and decode phases, hardware-specific features, and realistic request distributions. We propose $LLMCO_2$, a graph neural network (GNN)-based model, to improve the accuracy of LLM inference carbon footprint estimation by ~ 67% over prior approaches. Source code is available at https://github.com/fuzhenxiao/LLMCO2.

1 Introduction

Large language models (LLMs) are increasingly embedded in everyday applications such as web browsing and code generation, yet their deployment incurs substantial carbon costs [7]. Inference emissions can surpass those from training, with OpenAI processing over 270 million requests daily [5], each averaging 1.2K tokens [17]. Training GPT-4 required approximately 13 trillion tokens [16], and each training epoch consumes 3× the FLOPs of a single inference [7]. Consequently, just 121 days of inference yields emissions equivalent to one round of training. As LLM adoption grows [16], the time to reach inference–training emission parity continues to shrink.

The total carbon footprint of LLM inference comprises *operational* and *embodied* components [22]. Operational emissions stem from energy consumed during runtime, while embodied emissions arise from hardware manufacturing. In modern AI data centers, operational emissions account for up to 70% of total LLM-inference-related carbon output [22]. Accurately quantifying operational emissions under different latency and accuracy requirements [17] is essential for transparent billing and promoting sustainable usage.

Despite this urgency, *modeling tools for LLM inference operational emissions remain limited.* Users issue requests with varying batch sizes, prompt lengths, and generated token counts, while cloud providers rely on heterogeneous GPU deployments to meet diverse latency and accuracy requirements. Prior efforts [14] measured emissions on specific hardware platforms, but exhaustive profiling across all configurations is infeasible. Equation-based models for LLM training [7] fail to generalize to inference due to its lower and more variable hardware utilization. While ML-based tools exist for latency [11, 25] and energy [20] prediction on mobile devices, their application to LLM inference results in poor accuracy due to the following limitations:

- *Autoregressive phase unawareness*: Existing tools, developed for CNNs, do not distinguish between the compute-bound prefill and memory-bound decode phases in LLMs [11, 20, 25], leading to inaccurate predictions.
- *Omission of hardware-specific factors*: These models ignore critical GPU characteristics such as peak compute throughput, memory bandwidth, and network capacity, necessitating labor-intensive profiling across platforms.
- *Misrepresentation of real-world usage*: Uniform sampling overlooks realistic configurations, where typical cloud-based LLM requests involve small batch sizes, medium-length prompts, and limited token generation [12], thereby reducing relevance in practical scenarios.

We present $LLMCO_2$, an estimator for accurately modeling the operational carbon of LLM inferences. $LLMCO_2$ introduces a novel graph representation that models each LLM layer's kernels as a graph, where nodes represent kernels and edges encode data dependencies. Prefill and decode phases are represented separately, with node features incorporating Roofline-based hardware-specific metrics. To enhance generalization, we propose a targeted data sampling strategy that prioritizes common request, LLM, and GPU configurations. $LLMCO_2$ achieves ~ 67% higher prediction accuracy compared to existing ML-based energy estimators across diverse requests and hardware setups.

2 Background

LLM Carbon Footprint. The carbon footprint of LLM inference comprises operational and embodied components [7]. Operational emissions result from energy consumption during hardware execution, while embodied emissions reflect the carbon costs of hardware fabrication. The latter can be estimated from total chip area using analytical models [7]. This work focuses on modeling the operational carbon footprint of LLM inference.

LLM Inference Serving. Cloud-based LLM inference is typically managed by serving engines such as Sarathi [1] and vLLM [10], which handle request reception, batching, scheduling, and response generation. Consequently, the operational carbon footprint of LLM inference comprises two components: the serving engine's overhead and the inference computation itself.

Autoregressive LLM Inference. As shown in Figure 1, LLM inferences generate tokens autoregressively [17]. The initial iteration processes all input tokens in parallel to produce the first output—this

Authors' Contact Information: Zhenxiao Fu, Indiana University, Bloomington, USA, zhfu@iu.edu; Fan Chen, Indiana University, Bloomington, USA, fc7@iu.edu; Shan Zhou, Purdue University, West Lafayette, USA, zhou1487@purdue.edu; Haitong Li, Purdue University, West Lafayette, USA, haitongli@purdue.edu; Lei Jiang, Indiana University, Bloomington, USA, jiang60@iu.edu.

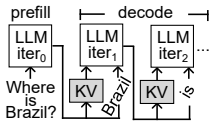
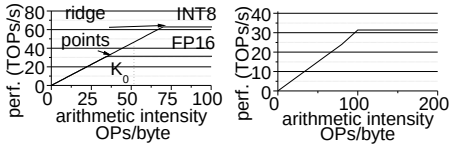
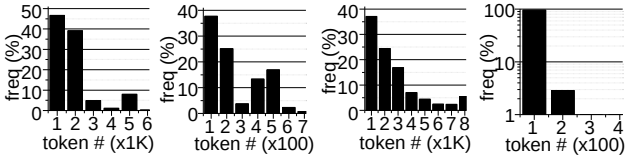


Fig. 1. An LLM autoregressive inference.



(a) HBM (OP: operation). (b) NVLink.

Fig. 2. The Roofline model.



(a) conv prompt. (b) conv gen. (c) code prompt. (d) code gen.

Fig. 3. The request distribution.

is the prefill phase [17]. Attention contexts are buffered in the key-value (KV) cache for subsequent iterations, where new tokens are generated using only the latest token and the KV cache—this defines the decode phase [17]. The prefill phase is compute-bound, dominated by hardware computing throughput, while the decode phase is memory-bound, constrained by KV cache access. This leads to distinct profiles in latency, energy, and carbon emissions. Ignoring this phase distinction or lacking sufficient training data significantly degrades energy prediction accuracy.

Transformer Layer. Each transformer layer [21] includes a masked multi-head attention (MHA) block and a feed-forward (FF) module, surrounded by normalization layers. The MHA block computes attention using query, key, and value kernels. Optimizations such as Grouped-Query Attention [2] and Flash Attention [6] reduce memory overhead by limiting key-value heads and fusing attention computations. The FF module is a two-layer MLP.

Inference Parallelism. Tensor parallelism (TP) [3, 13] accelerates large-model inference by splitting transformer weights and KV caches across multiple GPUs. It enhances throughput at high batch sizes and reduces latency by reduce-scatter and all-gather operations [9]. However, TP incurs frequent blocking communication, necessitating high-bandwidth interconnects like NVLink. In contrast, pipeline parallelism (PP) [13] partitions the model into sequential stages, assigning each GPU a subset of layers and transferring activations through send and recv operations. PP offers a better compute-to-communication ratio than TP but may suffer from pipeline bubbles.

Roofline Model. The Roofline model [4] estimates GPU kernel performance by incorporating peak compute throughput, memory and network bandwidth, and the kernel's arithmetic intensity—defined as total operations divided by data transferred. A ridge point marks the transition between compute-bound and bandwidth-bound regimes. A GPU with support for FP16 and INT8 under HBM yield separate ridge points (Figure 2(a)), while NVLink contributes another ridge point (Figure 2(b)). Kernels below the ridge point are bandwidth-bound; those above are compute-bound. For example, kernel K_0 is compute-bound in FP16 but memory-bound in INT8.

Request Characterization. In public LLM serving platforms such as Microsoft Azure, inference configurations (e.g., prompt length, generated token count) exhibit non-uniform distributions. Analysis of public Azure traces [12] shows that conversation requests typically have long prompts, averaging 1.1K tokens (Figure 3(a)), with generated token counts averaging 145 tokens (Figure 3(b)). Coding requests exhibit even longer prompts, averaging 2K tokens (Figure 3(c)), but shorter generated token counts, averaging only 28 tokens (Figure 3(d)). Major cloud providers employ mixed continuous batching [1], resulting in variable batch sizes.

Table 1. The comparison of $LLMCO_2$ against prior work.

advantages	[7]	[25]	[20]	[11]	**$LLMCO_2$**
prefill & decode phases	✗	✗	✗	✗	✓
hardware features	✓	✗	✗	✗	✓
real-world requests	✗	✗	✗	✗	✓
serving engine energy	✗	✗	✗	✗	✓

3 Related Work

We summarize the comparison between $LLMCO_2$ and prior methods in Table 1. LLMCarbon [7] estimates inference emissions using an equation-based model, but lacks accuracy due to its inability to capture variable hardware utilization. ML-based methods such as nn-Meter [25], DeepEn [20], and NNLQP [11] predict latency or energy for CNNs using neural networks, but treat inference as a monolithic task—failing to differentiate between the compute-bound prefill and memory-bound decode phases of autoregressive LLMs. These approaches rely on architecture-specific features and exhaustive sampling, yet omit essential hardware-level characteristics such as GPU throughput, memory bandwidth, and network capacity, limiting their generalizability across hardware platforms. Additionally, they uniformly sample inference configurations, failing to capture real-world patterns such as small batch sizes and medium-length prompts. None of these models account for the energy overhead of LLM serving engines. In contrast, $LLMCO_2$ explicitly models prefill & decode phases, incorporates hardware-specific features, emphasizes realistic request distribution, and includes serving engine overhead—yielding significantly more accurate carbon footprint estimates.

4 LLMCO2

We present $LLMCO_2$, a regression model for accurately estimating the operational carbon footprint of LLM inference, comprising both inference and serving engine carbon emissions. For inference operational energy, we introduce a graph-based representation of transformer layers executed across one or more GPUs, incorporating LLM architectural features and hardware-specific Roofline performance metrics for both prefill and decode phases. A 2-GNN architecture is employed: the first GNN predicts prefill energy, and the second estimates total inference energy. To construct training, validation, and test datasets, we propose a targeted energy data sampling strategy that emphasizes real-world request patterns, model architectures, and GPU configurations in cloud deployments. For serving engine operational energy, we develop an equation-based model to estimate overhead per batch of LLM inferences. The total operational energy E_{total}—the sum of inference and serving engine energy—is converted to carbon emissions using $E_{\text{total}} \cdot PUE \cdot CI$ [7], where PUE denotes power usage effectiveness and CI represents the data center's carbon intensity.

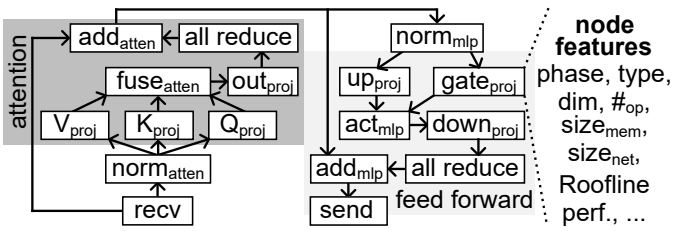


Fig. 4. The graph representation of an LLM layer.

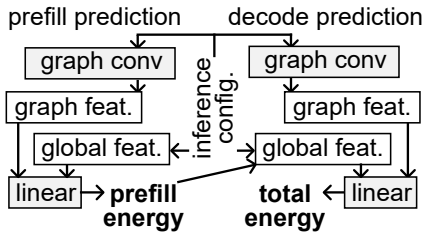


Fig. 5. Two GNNs for energy modeling.

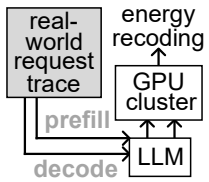


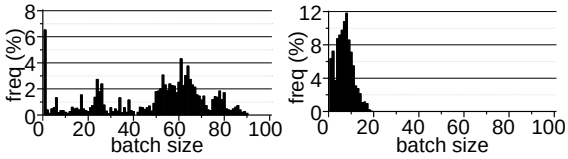
Fig. 6. Targeted energy sampling.

4.1 Inference Operational Energy

Graph Representation. We model each transformer layer as a directed acyclic graph of compute kernels, denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. As shown in Figure 4, each node $v \in \mathcal{V}$ corresponds to a compute kernel (e.g., $fuse_{atten}$ or Q_{proj}), and each edge $e \in \mathcal{E}$ captures the flow of data between kernels. Each node is annotated with two key attributes: (1) a one-hot encoded *kernel type* that specifies the operation performed, and (2) a *dimension* vector describing the input, weight, and output tensor shapes. To account for the distinct characteristics of the prefill and decode phases in autoregressive inference, each node is assigned two separate feature sets—one for each phase. Each feature set concatenates the following components:

- *Operation Number* indicates the total number of operations performed by the kernel. This is computed separately for the prefill and decode phases using the LLM analysis tool LLM-Viewer [24].
- *Memory Size* means the memory accessed by the kernel. Prefill and decode memory footprints are also extracted via LLM-Viewer.
- *Network Transfer* represents the volume of data transferred over the GPU network interface. For all-reduce kernels operating on an $n \times n$ matrix distributed across l GPUs, the transfer size S_{allr} is computed as $\frac{n^2}{l} \cdot (l-1) \cdot DS$, where DS denotes the byte size of each matrix element (e.g., $DS = 2$ for FP16). For pipeline parallelism involving l GPUs and batch size B, the transferred data size S_{pipe} is $(l-1) \cdot B \cdot n \cdot DS$.
- *Roofline Performance* quantifies the kernel's performance based on Roofline model analysis. For non-all-reduce kernels, the memory-related arithmetic intensity (MAI) is defined as the ratio of total operations to memory accessed. For all-reduce kernels, the network-related arithmetic intensity (NAI) is the ratio of total operations to network transfer. The memory and network ridge points (MRP and NRP) are computed as $Th_{\max}/BW_{\max}$ and $Th_{\max}/NET_{\max}$, respectively, where $Th_{\max}$ is the GPU's peak throughput, $BW_{\max}$ is its maximal memory bandwidth, and $NET_{\max}$ is its maximal network bandwidth. The Roofline performance of a kernel is defined as:
  - For non-all-reduce kernels: $\min(Th_{\max}, BW_{\max} \cdot MAI)$,
  - For all-reduce kernels: $\min(Th_{\max}, NET_{\max} \cdot NAI)$.

2-GNN Predictor. As shown in Figure 5, LLMCO$_2$ utilizes a dual-GNN architecture to capture the distinct energy characteristics of the prefill and decode phases in LLM inference. The first GNN predicts prefill energy, and the second estimates the total inference energy. Each GNN consists of multiple graph convolutional layers (e.g., GraphSAGE [8]) to extract graph-level embeddings, which are concatenated with global features comprising both inference and model configurations. Inference features include batch size, prompt length, token count, total FLOPs, memory access volume, and network transfer size. LLM configuration features encompass quantization bitwidth, hidden size, intermediate size, head count, and layer count. These combined features are passed through linear layers to produce energy predictions. The predicted prefill energy is appended to the global feature vector and fed into the second GNN to facilitate accurate estimation of total operational energy.



(a) conversation. (b) coding.

Fig. 7. The batch size distribution (LAM-8b on A100).

Targeted Energy Data Sampling. To build training, validation, and test datasets, we adopt a targeted sampling strategy that prioritizes impactful configurations across the joint space of LLM architectures, inference requests, and hardware platforms, rather than relying on random sampling (Figure 6). We first sample 50K points from $\mathcal{A} \times \mathcal{I} \times \mathcal{H}$, where $\mathcal{A}$ includes LLM variants (head count, layer count, intermediate size, quantization, hidden size), $\mathcal{I}$ captures real-world inference request distributions, and $\mathcal{H}$ covers diverse GPU setups. The serving engine batches requests of varying sizes; for example, coding tasks typically have smaller batches than conversation tasks (Figures 7(a) and 7(b)). We assess sampling quality by training GNN predictors and refining samples through fine-grained selection around high-error regions, iterating until achieving the target accuracy.

4.2 Serving Engine Operational Energy

The primary functions of a serving engine—request reception, batching, scheduling, and dispatching batches to the LLM—are not computationally intensive [1, 10]. As a result, the operational energy of the serving engine is dominated by system idle power, including CPU, DRAM, and other peripheral components. We estimate the serving engine's operational energy E_{serv} as $P_{\text{idle}} \cdot t + E_{\text{form}}$, where P_{idle} is system idle power, t means batch formation latency, and E_{form} denotes dynamic energy consumed during batch formation.

5 Experimental methodology

LLMs. We evaluated LLMCO$_2$ on three LLMs: meta-llama/Llama-2-7b-hf (LAM-7b) [19], internlm/internlm-20b (INT-20b) [23], and meta-llama/Llama-2-70b-hf (LAM-70b) [19], covering different model

Table 2. The configuration of GPU servers.

server	throughput (TOPs/s)		memory	network	node	area	tech
	FP16	INT8	(GB/s)	(GB/s)	size	(mm^2)	(nm)
A100	624	1248	2039	600	8	826	7
DGX	7nm 2× Xeon Platinum 8480C, 2TB DRAM DDR5						
H100	1979	3958	3430	900	8	814	5
DGX	7nm 2× AMD Rome 7742, 2TB DRAM DDR4						

Table 3. The MAPE comparison (The lower, the better).

scheme	LAM-7b	INT-20b	LAM-70b	mean
LLMCarbon	97.2%	115.6%	321.9%	178.2%
DeepEn	36.2%	42.3%	48.6%	42.4%
NNLQP	32.4%	38.6%	49.3%	40.1%
$LLMCO_2$	**12.3%**	**14.6%**	**19.8%**	**15.6%**

scales. Each model is characterized by quantization bitwidth (b), hidden size (h), intermediate size (i), head count (c), and layer count (l). The LLM details are shown as:

- LAM-7b: $b = 16$, $h = 4096$, $i = 11008$, $c = 32$, $l = 32$
- INT-20b: $b = 16$, $h = 5120$, $i = 13824$, $c = 40$, $l = 60$
- LAM-70b: $b = 16$, $h = 8192$, $i = 28672$, $c = 64$, $l = 80$

GPUs. LLM inferences were performed on NVIDIA DGX systems (details in Table 2), each comprising 8 GPUs interconnected via NVLink, supporting tensor parallelism degrees of 1, 2, 4, and 8. Both A100 and H100 GPUs were used, representing typical hardware in commercial cloud-based LLM deployments. The A100 DGX system includes two Intel Xeon Platinum 8480C CPUs and 2TB of DDR5 DRAM, while the H100 DGX system is configured with two AMD EPYC 7742 CPUs and 2TB of DDR4 DRAM.

Request Traces. Public Azure LLM serving traces [12] were used to generate inference requests, encompassing two distinct traces: one for chat and the other for code completion.

Serving Engine. For evaluation, we adopt Sarathi [1], a state-of-the-art LLM inference serving engine that offers a superior throughput–latency trade-off compared to alternatives such as vLLM [10].

Measurement. We used the Nvidia Management Library [15] and the RAPL interface to measure the energy consumption of LLM inferences on the target GPUs and CPUs (including DRAMs), respectively. Each inference was executed 5 times, and the average value was recorded as the ground truth.

Dataset Construction. We generated LLM inference requests based on real-world request traces and submitted them to the serving engine deployed on the GPU systems listed in Table 2. The batch size distribution made by the serving engine is shown in Figure 7. The dimensions of tensor parallelism and pipeline parallelism for each batch can be 1, 2, and 4. Using our targeted energy data sampling, we constructed training and validation datasets using all three LLMs. In contrast, the training and validation datasets for baseline schemes were built using random sampling. The dataset sizes were 8K for training, 1K for validation, and 1K for testing.

Implementation. Each GNN in $LLMCO_2$ comprises two GraphSAGE layers [8] followed by two linear layers. $LLMCO_2$ is implemented using PyG [18] and trained on a Tesla L4 GPU using the Adam optimizer with a learning rate of 0.01 and a batch size of 64.

Evaluation Metrics. We assess prediction accuracy using Mean Absolute Percentage Error (MAPE) and Error Bound Accuracy (EBA(δ)). MAPE measures the average absolute percentage deviation between predicted and actual energy values, where lower values indicate better accuracy. EBA(δ) denotes the proportion of predictions falling within a δ% error margin of the ground truth, with higher values indicating greater precision.

Table 4. The EBA comparison (The higher, the better).

scheme	EBA(30%)				EBA(10%)			
	LAM-7b	INT-20b	LAM-70b	mean	LAM-7b	INT-20b	LAM-70b	mean
LLMCarbon	8.2%	5.4%	12.3%	8.63%	2.1%	1.6%	2.2%	1.97%
DeepEn	48.3%	52.3%	56.6%	52.4%	38.2%	41.6%	43.2%	41%
NNLQP	52.4%	61.6%	48.3%	54.1%	39.6%	43.5%	40.1%	41.7%
$LLMCO_2$	**79.5%**	**81.3%**	**84.2%**	**81.7%**	**62.3%**	**75.7%**	**67.8%**	**68.6%**

Baseline Schemes. We compare $LLMCO_2$ against followings.

- *LLMCarbon* [7] uses a FLOP-counting equation-based model to estimate carbon footprint, without accounting for hardware variability or execution phases.
- *DeepEn* [20] collects energy samples across GPUs and trains a random forest model for kernel-level energy prediction.
- *NNLQP* [11] adopts a GNN-based approach to model energy consumption by representing each model layer as a graph.

However, they do not distinguish prefilling from decoding, incorporate hardware-related features, or prioritize sampling of prevalent configurations in cloud-based LLM serving.

6 Validation and Analysis

6.1 LLM Inference Operational Energy Predictor

MAPE. Table 3 compares the Mean Absolute Percentage Error (MAPE) of $LLMCO_2$ with baseline schemes across different LLMs. On average, $LLMCO_2$ achieves the lowest MAPE values, demonstrating superior prediction accuracy. LLMCarbon, relying solely on FLOP counts, disregards memory accesses within transformer layers and essential hardware-specific metrics such as GPU memory and network bandwidth, resulting in the highest MAPE across all LLMs. A MAPE greater than 100% indicates that the prediction errors are significantly larger than the ground truth values. The ML-based predictors, DeepEn and NNLQP, exhibit similar average MAPE values. However, DeepEn slightly outperforms NNLQP on LAM-70b due to its random forest model's robustness with smaller training datasets. Overall, $LLMCO_2$ reduces the average MAPE by 63.2% compared to DeepEn and 61% compared to NNLQP by (1) separately modeling prefill and decode phases, (2) incorporating kernel-level Roofline performance, and (3) leveraging energy data derived from real-world serving traces.

EBA. Table 4 reports the Error Bound Accuracy (EBA) at 30% and 10% error margins for various energy predictors. $LLMCO_2$ consistently achieves the highest EBA values across all error bounds and LLMs. In contrast, LLMCarbon records the lowest EBA values, as it neglects memory accesses and network data transfers, resulting in substantial errors, particularly for LLMs with long decode phases and those constrained by all-reduce kernels (e.g., LAM-70b). Although DeepEn and NNLQP display comparable MAPE values, NNLQP attains higher EBA values at smaller error bounds, benefiting from its GNN model's capacity to learn finer-grained energy patterns when sufficient training data is available. Overall, $LLMCO_2$ improves the average EBA(10%) by 67.3% and 64.5% over DeepEn and NNLQP, respectively.

Table 5. The ablation study (EBA(10%)) of $LLMCO_2$.

component	LAM-7b	INT-20b	LAM-70b	mean
+prefill/decode	51.6%	58.4%	55.1%	55.03%
+Roofline	56.2%	66.3%	61.2%	61.23%
+targeted sampling	62.3%	75.7%	67.8%	68.6%

Ablation Study. We performed ablation studies on EBA(10%) to assess the contribution of each component in $LLMCO_2$, as shown in Table 5. Separating node features for the prefill and decode phases increases EBA(10%) by 32% compared to NNLQP. This separation is particularly effective in real-world LLM serving clouds, where most requests involve medium-length prompts and limited token generation, causing significant prediction errors when the two phases are combined. Incorporating Roofline performance as a node feature further improves $LLMCO_2$'s EBA(10%) by 11.3%, as it enables effective knowledge transfer between different GPU servers. Finally, the targeted energy data sampling strategy raises $LLMCO_2$'s EBA(10%) to 164.5% of NNLQP, as training with data distributions that reflect typical LLM inference configurations enhances its accuracy on test datasets with similar prompt lengths and token distributions.

6.2 Serving Engine Energy Estimator

On average, the EBA(10%) of serving engine energy per batch is 89.2% for the conversation trace and 98.8% for the coding trace. In the conversation trace, request arrival intervals are short, resulting in an average batch formation latency that is only 11.7% of the LLM inference latency, with a median value of 6.4%. Consequently, serving engine energy is dominated by CPU batch formation energy, which $LLMCO_2$ approximates using a fixed value, leading to higher modeling errors. In contrast, the coding trace exhibits larger request arrival intervals, with the average and median batch formation latency reaching 13× and 11× that of LLM inferences, respectively. Here, serving engine energy is primarily driven by system idle energy, which is more predictable, resulting in much higher accuracy.

7 User Case Studies

Carbon Emission Speed Comparison. LLM training requires extensive GPU usage at high throughput over prolonged periods. For instance, LAM-70b training involves 800 A100 GPUs operating at ~50% peak throughput for three months [19]. In contrast, a typical LAM-70b inference employs four A100 GPUs at 10%–40% throughput for 2-4 seconds. Figure 8 presents the normalized carbon emission per GPU per second for LAM-70b training and inference, relative to the training baseline. The embodied carbon per GPU per second remains constant across configurations due to the uniform use of four A100 GPUs. For conversation requests, characterized by an average prompt length of 1.1K tokens, 145 generated tokens, and a batch size of 48, the prefill phase (conv-p) achieves slightly reduced GPU utilization, lowering operational carbon emissions per second by 7% compared to training. However, the decode phase (conv-d), with significantly lower GPU utilization, reduces operational carbon emissions per second by 74%. Coding requests, with an average prompt length of 2K tokens, 27 generated tokens, and a batch size of 7, exhibit substantially lower GPU utilization. Compared to conversation requests, the prefill phase (code-p) reduces

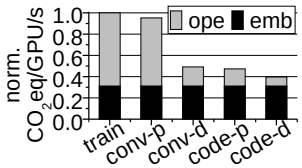


Fig. 8. The comparison of carbon emission per GPU per second (emb: embodied carbon; ope: operational carbon; LAM-70b on A100 with TP=4).

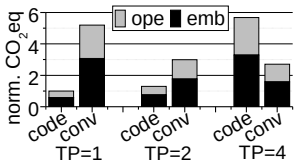


Fig. 9. The comparison of carbon emission between various TP settings (emb: embodied carbon; ope: operational carbon; LAM-7b on A100).

operational carbon emissions per second by 75%, while the decode phase (code-d) reduces them by 48%. Unlike training, the embodied carbon overhead dominates the prefill phase of coding inferences and both phases of both coding and conversation inferences.

Inference w Various TP Settings. We observe that increasing the number of GPUs does not always accelerate LLM inference, particularly for coding requests characterized by small batch sizes and short prompts. Consequently, employing more GPUs than necessary can be unsustainable. Figure 9 illustrates the carbon emission comparison for LAM-7b inferences across varying tensor parallelism (TP) settings, normalized to the carbon emission of coding inferences with TP=1. For conversation requests with larger batch sizes and longer prompts, increasing the TP setting reduces inference latency and operational energy per GPU, as the workload per GPU decreases. Conversely, for coding requests with smaller batch sizes and shorter prompts, adding more GPUs increases inference latency and operational energy per GPU due to the communication overhead of all-reduce kernels. As a result, the total carbon footprint decreases significantly for conversation inferences with higher TP settings but increases substantially for coding inferences.

8 Conclusion

LLM inference produces a larger carbon footprint than training, necessitating accurate estimation tools for both users and cloud providers. Existing models fall short due to their inability to capture LLM autoregressive behaviors, hardware-specific features, and real-world request configuration distribution. We presented $LLMCO_2$, a GNN-based model to address these challenges, offering ~ 67% improved accuracy in predicting the carbon footprint of LLM inferences compared to prior methods.

Acknowledgments

This work was supported in part by NSF CCF-2105972, OAC-2417589, and CAREER AWARD CNS-2143120.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. *USENIX Symposium on Operating Systems Design and Implementation* (2024).

[2] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *the Conference on Empirical Methods in Natural Language Processing*. 4895–4901.

[3] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *IEEE International Conference on High Performance Computing, Networking, Storage and Analysis*.

[4] David Cardwell and Fengguang Song. 2019. An extended roofline model with communication-awareness for distributed-memory hpc systems. In *International Conference on High Performance Computing in Asia-Pacific Region*. 26–35.

[5] Andrew A Chien, Liuzixuan Lin, Hai Nguyen, Varsha Rao, Tristan Sharma, and Rajini Wijayawardana. 2023. Reducing the Carbon Impact of Generative AI Inference (today and in 2035). In *ACM HotCarbon Workshop on Sustainable Computer Systems*.

[6] Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations*.

[7] 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 *The Twelfth International Conference on Learning Representations*.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, Vol. 30.

[9] Mert Hidayetoglu, Simon Garcia De Gonzalo, Elliott Slaughter, Yu Li, Christopher Zimmer, Tekin Bicer, Bin Ren, William Gropp, Wen-Mei Hwu, and Alex Aiken. 2024. CommBench: Micro-Benchmarking Hierarchical Networks with Multi-GPU, Multi-NIC Nodes. In *ACM International Conference on Supercomputing*. 426—-436.

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *ACM SIGOPS 29th Symposium on Operating Systems Principles*.

[11] Liang Liu, Mingzhu Shen, Ruihao Gong, Fengwei Yu, and Hailong Yang. 2023. NNLQP: A Multi-Platform Neural Network Latency Query and Prediction System with An Evolving Database. In *ACM International Conference on Parallel Processing*. Article 78, 14 pages.

[12] Microsoft. 2024. Azure Trace. https://github.com/Azure/AzurePublicDataset/blob/master/AzureLLMInferenceDataset2023.md.

[13] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. In *ACM International Conference for High Performance Computing, Networking, Storage and Analysis*. Article 58, 15 pages.

[14] Sophia Nguyen, Beihao Zhou, Yi Ding, and Sihang Liu. 2024. Towards Sustainable Large Language Model Serving. In *ACM HotCarbon Workshop on Sustainable Computer Systems*.

[15] NVIDIA. 2024. NVIDIA Management Library (NVML). https://developer.nvidia.com/management-library-nvml

[16] OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[17] P. Patel, E. Choukse, C. Zhang, A. Shah, I. Goiri, S. Maleki, and R. Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *ACM/IEEE International Symposium on Computer Architecture*. 118–132.

[18] PyG Team. 2024. PyTorch Geometric. https://pytorch-geometric.readthedocs.io/.

[19] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR* abs/2307.09288 (2023). doi:10.48550/ARXIV.2307.09288

[20] Xiaolong Tu, Anik Mallik, Dawei Chen, Kyungtae Han, Onur Altintas, Haoxin Wang, and Jiang Xie. 2023. Unveiling energy efficiency in deep learning: Measurement, prediction, and scoring across edge devices. In *IEEE/ACM Symposium on Edge Computing*. 80–93.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30.

[22] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. 2022. Sustainable ai: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems* 4 (2022), 795–813.

[23] Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. InternLM2.5-StepProver: Advancing Automated Theorem Proving via Expert Iteration on Large-Scale LEAN Problems. arXiv:2410.15700 [cs.AI] https://arxiv.org/abs/2410.15700

[24] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. 2024. LLM Inference Unveiled: Survey and Roofline Model Insights. arXiv:2402.16363 [cs.CL]

[25] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. nn-Meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *ACM International Conference on Mobile Systems, Applications, and Services*. 81––93.