

Watts Behind the Weights: Characterizing Energy Consumption of the AI Training Lifecycle in HPC

Alexandre Fernandes*
INESC TEC & U. Minho
alexandre.r.fernandes@inesctec.pt

Alicia Oliveira*
INESC TEC & U. Minho
alicia.oliveira@inesctec.pt

Beatriz Cepa*
INESC TEC & U. Minho
beatriz.cepa@inesctec.pt

Cláudia Brito
INESC TEC & U. Minho
claudia.v.brito@inesctec.pt

Ricardo Macedo
INESC TEC & U. Minho
ricardo.g.macedo@inesctec.pt

Abstract—Modern AI training workloads have become a leading contributor to energy consumption in HPC infrastructures. Yet this consumption is treated as a monolithic block, attributed to execution alone. The training lifecycle includes preparation and post-execution stages, and execution itself comprises multiple phases whose energy consumption varies with the task, model, dataset, and infrastructure-related factors. This work closes this gap by characterizing energy consumption across the entire AI training lifecycle in HPC infrastructures, following a fine-grained approach that attributes energy to individual phases and events, rather than to training as a whole. We apply this methodology on a production supercomputer, assessing preparation and post-execution using distinct datasets and models, and characterizing execution over workloads with distinct resource requirements, namely the I/O-intensive ResNet-50 and the GPU-bound Qwen3-8B. We draw several lessons to help HPC users accurately attribute energy consumption across their workloads and make their training pipelines more energy-efficient.

Index Terms—energy-efficiency, AI training, sustainable HPC.

I. INTRODUCTION

The rapid growth in the size and complexity of modern AI workloads has made training large-scale models a routine, resource-intensive activity on HPC supercomputers. As these continue to grow, so does the associated energy footprint, raising concerns not only about operational costs but also the environmental sustainability of AI development [3, 38].

Challenges. The lifecycle of AI training, however, is longer and more complex than the execution of a single training script. Before execution, a *preparation stage* takes place, in which users acquire their model and dataset, typically from external repositories (e.g., Kaggle, HuggingFace, GitHub), incurring network, CPU, and storage costs. *Execution* itself is not a single, homogeneous stage either, but a sequence of phases repeated at each training iteration: data samples are loaded from persistent storage, either node-local or a parallel file system (e.g., Lustre), moved from host memory to the GPU memory, and processed through forward and backward passes, gradient synchronization, and weight update, with periodic validation and checkpointing [23, 24]. Finally, upon completion, the resulting model is transferred outside the HPC

facility for deployment and inference. Each of these stages and phases stresses different hardware resources and exhibits distinct energy behaviors [20]. Prior work [11, 14, 19, 21, 29, 30, 33, 37], however, has largely reported energy consumption at the granularity of an entire training run, remaining agnostic to how much each stage or phase individually contributes to execution time and energy consumption. Without this visibility, it is challenging to identify where energy is being spent, and, consequently, to design targeted strategies for reducing training energy consumption in HPC environments.

This work. We present a characterization study of energy consumption across the complete AI training lifecycle in HPC infrastructures, spanning *preparation*, *execution*, and *post-execution* stages. Unlike prior work, we instrument the training pipeline at the level of individual events using a custom-built monitoring tool that collects fine-grained, software-based CPU and GPU energy measurements and correlates them with event-level traces that mark the boundaries of each phase. This correlation enables us to attribute energy consumption to specific phases and events, which in turn allows us to evaluate how factors such as data placement (parallel file system vs. node-local storage) and checkpointing affect this breakdown. We apply this methodology across the full training lifecycle: the *preparation* and *post-execution stages* are evaluated over 7 datasets and 5 models spanning several orders of magnitude in size (datasets range from 486 MB to 584 GB, and models range from 410 MB and 51.6 GB), while the *execution stage* is evaluated on two workloads with distinct resource profiles, ResNet-50, an I/O-bound convolutional neural network, and Qwen3-8B, a GPU-bound large language model. To the best of our knowledge, this is the first study to quantify the energy cost of individual phases across the entire AI training lifecycle in an HPC setting, rather than isolating execution alone.

Experimental results and takeaways. Our characterization study uncovers several findings. First, no single phase dominates energy consumption across workloads; for instance, ResNet-50 distributes energy across both CPU- and GPU-bound phases, while Qwen3-8B concentrates it almost entirely in GPU computation. Second, execution time is not a reliable

*Equal contribution.

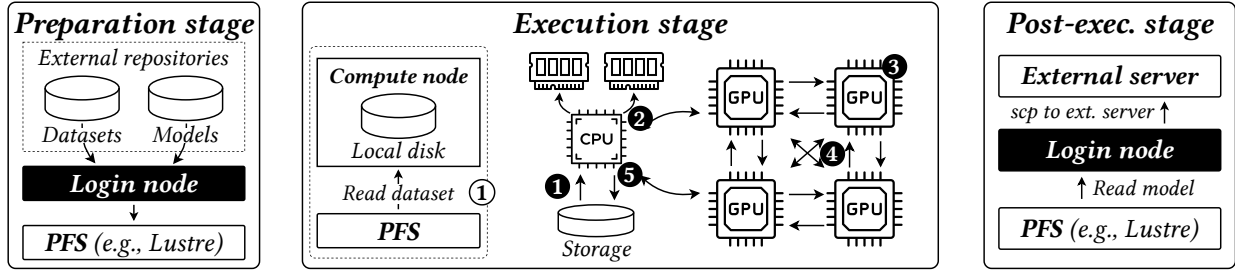


Fig. 1: Typical lifecycle of AI training in modern HPC infrastructures.

proxy for energy consumption, as phases of comparable duration can incur substantially different energy costs. Further, data placement and checkpointing similarly affect energy in workload-dependent ways: local storage benefits I/O-sensitive workloads far more than compute-bound ones, and checkpointing cost depends less on the volume of persisted state than on whether it is exposed by, or overshadowed by, the workload’s dominant phase. Finally, we show that preparation and post-execution costs, though individually modest, are incurred repeatedly and redundantly at the scale of shared HPC systems, representing a source of avoidable energy consumption.

Contributions. This paper makes the following contributions:

- A **principled approach** to *characterize energy consumption of the AI training lifecycle in modern HPC infrastructures*, spanning preparation, execution, and post-execution stages.
- An **in-depth analysis** of the *energy consumption and execution time of the distinct phases of the training pipeline over I/O-bound and GPU-bound models*.
- A set of **empirical findings and takeaways** on the *workload- and infrastructure-dependent nature of AI training energy consumption*, intended to guide HPC users toward more energy-efficient training practices.

II. TRAINING LIFECYCLE IN HPC

This section characterizes the AI training lifecycle in HPC infrastructures, comprising three sequential stages: *preparation*, *execution*, and *post-execution*, as depicted in Fig. 1. In this study, we adopt a simplified view of the pipeline, focusing on the most common phases and those that dominate GPU and CPU utilization and, consequently, power draw [11, 33, 37].

A. Preparation stage

Before training can proceed, users must first prepare the experimental setup, conducted in two main steps.

Dataset acquisition. Depending on the target task (e.g., image classification, natural language processing), datasets range from a few MBs to several GBs, or even TBs, in size [22, 31]. Datasets are typically transferred via tools such as `curl`, `wget`, and `scp`, executed on the CPU of a login node, with the retrieved data stored directly on the shared parallel file system (e.g., Lustre). The performance of this step is generally limited by the external network link or by rate limiting imposed at the data source, rather than by local compute or storage resources.

Model acquisition. Analogously to dataset acquisition, users must fetch a model prior to training, typically from a public model repository. Model size scales with the number of parameters and ranges from a few MBs to several GBs. This step is also network-bound and is performed on the login node.

During this stage, additional steps may occur, such as compilation, decompression, and code preparation. However, since these are highly user- and task-dependent, we defer their characterization to future work.

B. Execution stage

Once the preparation stage is complete, execution proceeds on the allocated compute nodes. The training pipeline spans GPU, CPU, memory, storage, and network resources, and comprises the following phases within each iteration.

Data transfer. Prior to training, the dataset may be transferred from the shared parallel file system (PFS) to the compute node’s local storage to accelerate data access and reduce I/O variability during training [6] (①).

Data loading. Training samples are read from persistent storage (local or PFS) into host memory [24] (①). This step is typically implemented asynchronously: while the GPU processes the current batch, the next batch is prefetched from storage, overlapping data fetching with GPU computation. However, when storage access is slower than computation, the GPU stalls until the batch becomes available, making data loading synchronous during this period.

Memory transfer. Loaded samples are organized into batches and transferred from host memory to GPU memory [24] (②).

GPU computation. The core training computation executes on the GPU and comprises three phases: a *forward pass*, which propagates inputs through the model to produce predictions and compute the loss; a *backward pass*, which computes gradients with respect to model parameters via backpropagation; and an *optimizer step* (e.g., SGD, Adam), which updates model parameters using the computed gradients [20, 23] (③).

Synchronization. In multi-GPU and distributed training, workers must synchronize before training can proceed, introducing inter-device dependencies (④). The specific communication pattern and the data exchanged depend on the parallelization strategy employed [11, 37].

Checkpointing. To enable fault tolerance and recovery, the model is periodically checkpointed, persisting its training

state (e.g., model parameters, optimizer state) to durable storage [23], either remote or local (📍).

Validation. Periodically, the model is evaluated on a held-out dataset, consisting of data loading and forward passes [20].

C. Post-execution stage

Upon successful completion of training, the resulting model, along with associated artifacts such as logs and training statistics, is collected. The model is typically transferred outside the HPC facility (e.g., to an external organization) for deployment and inference.¹ As with model acquisition, this transfer operates at the CPU and is network-bound.

III. METHODOLOGY

This section describes the methodology used to characterize the energy consumption of the training lifecycle in HPC.

Hardware and OS configurations. Experiments were conducted on the Deucalion supercomputer [25], on a server equipped with 2× 64-core AMD EPYC processors, 4× NVIDIA A100 80GB GPUs, 512 GiB of memory, and a 500 GiB SATA SSD, connected to a shared Lustre file system via a 200 Gb/s interconnect. The system runs Rocky Linux 8.5 with PyTorch 2.6.0, CUDA 11.8, and NCCL 2.21.5.

Training models and datasets. To assess energy consumption across different phases of the training lifecycle, we conducted experiments using distinct models and datasets tailored to each part of the evaluation. Specifically, we used five models spanning different parameter count ranges: ResNet-50 (M1, 25.6M) [12], ELECTRA (M2, 110M) [4], TRELIS (M3, 1.1B) [34], Qwen3 (M4, 8B) [36], and Gemma (M5, 27B) [10]. For datasets, we used hd_tmp (D1, 486 MB) [1], WikiText (D2, 644 MB) [32], the arXiv Dataset (D3, 2.06 GB) [5], Documentation Images (D4, 6.51 GB) [15], ImageNet (D5, 167 GB) [8], KakologArchives (D6, 225 GB) [17], and Banned Historical Archives (D7, 549 GB) [2].

Training configurations. To assess energy consumption of the execution stage, we trained ResNet-50 on the ImageNet dataset and Qwen3-8B on the WikiText dataset, given their I/O-bound and GPU-bound nature, respectively. ResNet-50 was trained using Distributed Data Parallel, optimized with SGD, with a global batch size of 256 split across the GPUs, in FP32. In contrast, Qwen3-8B was trained using Fully Sharded Data Parallel, sharding model parameters, gradients, and optimizer state across GPUs; it used the AdamW optimizer with a per-rank batch size of 8, in BF16. All experiments were conducted for 5 epochs and repeated over 3 runs; when applicable, each epoch was followed by a checkpoint.

Metric collection. We developed a custom energy monitoring tool that collects and attributes CPU and GPU energy consumption to the training lifecycle. CPU energy is derived from Intel’s RAPL [7] Model-Specific Register (MSR) readings, combined with the busy clock cycles of the processes involved in training, obtained via `perf_event_open`. GPU

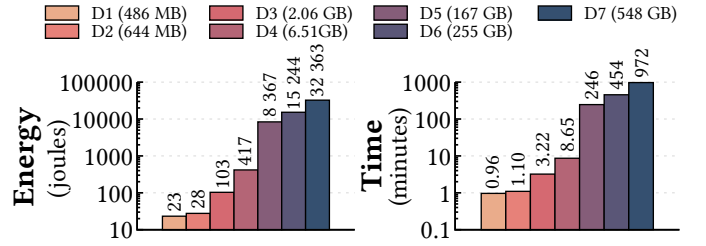


Fig. 2: Energy consumption (joules) and execution time (in minutes) of the dataset acquisition step.

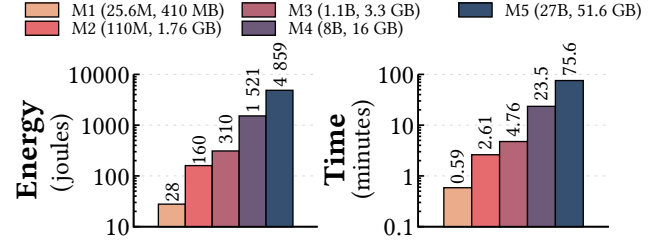


Fig. 3: Energy consumption (joules) and execution time (in minutes) of the model acquisition step.

energy is collected via NVML [28], which exposes per-device power and utilization measurements. To minimize monitoring overhead, metrics are sampled at 50 ms intervals for CPU and GPU. Collected metrics are correlated offline with training events via a logging and tagging mechanism, enabling energy consumption to be attributed to specific training phases.

We focus on software-based measurement tools to obtain a granular view of energy consumption throughout training. As these tools monitor specific components rather than the full system, they underreport total energy consumption; however, they preserve the same curves and tendencies at a lower global energy scale, and remain more broadly accessible across HPC systems [9, 16, 18]. Our methodology remains generalizable and stands to directly benefit from future improvements in the accuracy of software-based energy measurement.

Instrumentation. Energy metrics reported in isolation from the training behavior provide limited insight into the energy consumption patterns of modern workloads. As such, we instrument all training scripts with OpenTelemetry² to mark the main pipeline stages, namely data loading, memory copy, forward and backward passes, and weight update, for both training and validation. Moreover, to expose internal PyTorch operations, including asynchronous data prefetching and inter-GPU synchronization, we instrument PyTorch directly.

IV. EXPERIMENTAL EVALUATION

Our evaluation aims at answering the following questions:

- How does each individual stage of the training lifecycle contribute to overall energy consumption (§IV-A-§IV-C)?
- How do execution time and energy consumption differ across workloads with distinct resource profiles (§IV-B)?

¹While model serving is becoming an emerging workload in HPC facilities, we defer its energy characterization to future work.

²<https://opentelemetry.io/>

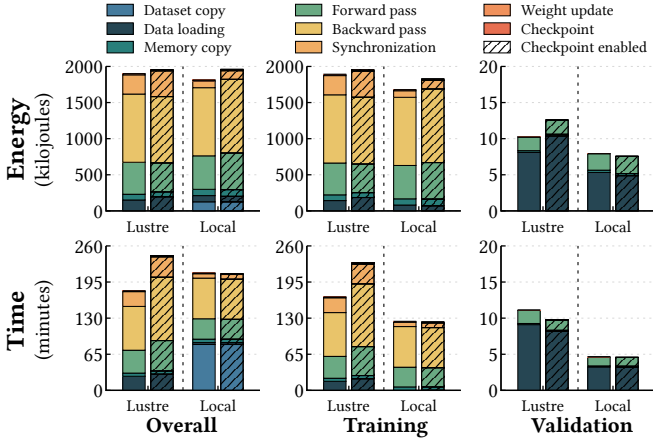


Fig. 4: Overall, training-only, and validation-only per-phase energy consumption and execution time under ResNet-50.

- How do data placement and checkpointing affect training time and energy consumption (§IV-B)?

A. Preparation phase

We begin by characterizing the energy consumption and execution time of *dataset* and *model acquisition*.

Dataset acquisition. We measure the time and energy consumption of transferring the datasets described in §III from their respective repositories (*i.e.*, HuggingFace, Kaggle) to the shared PFS, depicted in Fig. 2. Results show that both energy consumption and execution time proportionally increase with dataset size, from 23 J and 0.96 min for D1 to 32 kJ and 972 min for D7. As dataset acquisition is network-bound rather than compute-bound, this increase is driven primarily by longer transfer times rather than higher computational activity.

Model acquisition. Similar to dataset acquisition, downloading a model from its repository to the shared PFS is a network-bound operation executed on the login node’s CPU. Fig. 3 depicts the energy consumption and execution time of model download across model sizes. Costs range from 28 J and 0.59 min for M1 (25.6M parameters) to 4.9 kJ and ≈ 75 min for M5 (27B parameters), showing that both metrics increase in a near-linear relationship with model size, rather than model architecture or complexity.

Takeaway. While the preparation stage consumes comparatively little energy relative to other stages of the training lifecycle, it accounts for a disproportionately high share of execution time. This cost is negligible in isolation, but in shared, large HPC systems, the same datasets and models are frequently downloaded by hundreds of users. At this scale, energy that appears negligible per download accumulates into substantial, avoidable costs, highlighting opportunities for optimization.

B. Execution stage

We now evaluate the *execution stage* under four configurations, combining two storage placements (PFS and local storage)

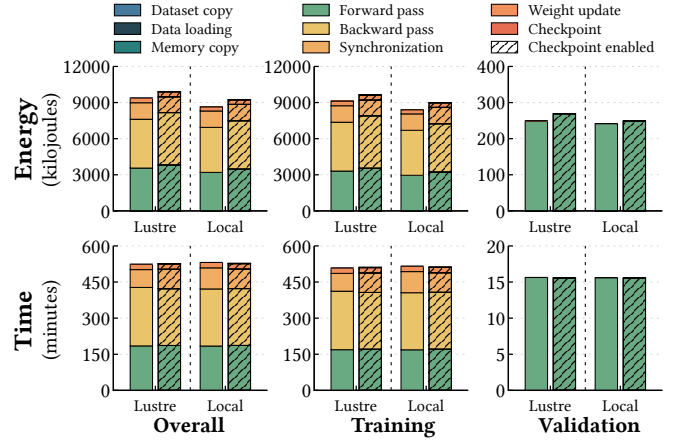


Fig. 5: Overall, training-only, and validation-only per-phase energy consumption and execution time under Qwen3-8B.

with and without checkpointing, to isolate the effects of data placement and fault tolerance on runtime and energy. Figs. 4 and 5 report the overall, training, and validation energy consumption and execution time, broken down per phase, for ResNet-50 and Qwen3-8B, respectively, while Table I reports the corresponding CPU-GPU energy consumption proportion.

Dataset transfer from PFS to local disk. Copying the dataset from Lustre to the local disk is a common strategy to accelerate training and reduce performance variability [6]. However, the performance and energy gains achieved during local training, compared to Lustre, may be offset by the one-time cost of the dataset copy itself, as observed in Fig. 4, *overall vs. training*. This trade-off depends heavily on dataset size (D5 vs. D2), on the workload’s resource profile (I/O-bound vs. compute-bound), and on execution length (*e.g.*, number of epochs).

Takeaway. The cost of dataset copying should be assessed not only by dataset size but also by execution volume (*i.e.*, number of epochs) and overall execution duration.

Overall analysis. As shown in Figs. 4 and 5, although the two workloads exhibit different resource bottlenecks and phase compositions, both are dominated by *training* (center), with *validation* (right) contributing only a small fraction of total energy and execution time. ResNet-50 exhibits a heterogeneous execution profile, in which both CPU (16% on average) and GPU (84% on average) contribute to overall energy consumption, with the backward pass being the largest contributor. Qwen3-8B, in contrast, being compute-heavy, concentrates most of its execution time and energy consumption in the GPU computation phases (forward pass, backward pass, weight update) and inter-device synchronization.

Results further show that execution time alone is not a reliable proxy for energy consumption. For ResNet-50, dataset copying and data loading account for a non-negligible share of execution time ($\approx 39\%$ and $\approx 7\%$, respectively) but a disproportionately smaller share of energy consumption. This stems from both phases being CPU-exclusive, with the GPUs idle

TABLE I: Energy split between CPU and GPU, with (ckp) and without checkpointing over Qwen3-8B and ResNet-50.

	Qwen3-8B		ResNet-50	
	CPU	GPU	CPU	GPU
<i>Lustre</i>	7.8%	92.2%	16%	84%
<i>Lustre (ckpt)</i>	7.1%	92.9%	20.5%	79.5%
<i>Local</i>	7.3%	92.7%	12.2%	87.8%
<i>Local (ckpt)</i>	8.4%	91.6%	15.4%	84.6%

throughout their execution, combined with the CPU’s lower power draw relative to the GPU. For Qwen3-8B, time and energy distributions are more closely aligned, as most of its execution time and power draw occur on the GPU.

Takeaway. No single phase dominates energy consumption across workloads, and execution time alone is not a reliable proxy for energy consumption, as phases of comparable duration can incur different energy costs. This emphasizes the need for phase-level, rather than workload-level energy attribution to correctly identify optimization targets.

Lustre vs. local disk. Data placement affects the two workloads differently. For ResNet-50, local storage reduces data loading time (and energy) in both training (-17 min, -88 kJ) and validation (-5 min, -4 kJ), due to its lower latency and variability compared to the remote PFS, resulting in fewer I/O stalls [6, 24]. Table I shows a corresponding decrease in CPU energy share and an increase in GPU when moving from Lustre to local storage, suggesting that the GPU remains more consistently active throughout training. For Qwen3-8B, execution time and energy are similar across placements, since the dataset (WikiText, 644 MB) fits entirely in GPU memory once read from persistent storage, and computation remains the dominant bottleneck regardless of storage backend.

Takeaway. Local storage placement improves execution time and energy consumption for I/O-sensitive workloads, but yields limited benefit for compute-bound workloads.

Checkpoint enabled vs. disabled. Checkpointing also affects the two workloads differently, primarily driven by data placement. For ResNet-50, enabling checkpointing over Lustre noticeably increases both execution time and energy consumption, but has minimal impact over local storage; this difference is caused by the additional latency and variability of PFS accesses. Counter-intuitively, despite Qwen3-8B’s checkpoint volume being substantially larger than ResNet-50’s, given its considerably larger parameter and optimizer state, its time and energy impact remains minimal, as execution continues to be dominated by GPU computation.

Takeaway. Checkpointing cost is not simply a function of the volume of state being persisted, but of how that cost

compares to a workload’s dominant phases: for GPU-bound workloads, even a large checkpoint is overshadowed by computation, whereas for I/O-intensive workloads, its cost is more exposed when writing directly to the PFS.

C. Post-execution stage

Model upload. Model upload follows a process analogous to model acquisition (§IV-A), transferring the trained model to a remote host outside the HPC facility rather than downloading it. From a computational perspective, both operations are equivalent: the transfer remains network-bound and executes on the login node’s CPU, regardless of direction. We draw conclusions similar to those in *model acquisition* and omit a dedicated discussion here.

Takeaway. As the energy cost of model upload is incurred after training, it is typically excluded from reported energy figures for the training lifecycle.

V. RELATED WORK

We now discuss prior work and position this study in context.

Measuring and reporting energy. MLPerf Power [33] aims to standardize how ML energy efficiency is tested and reported across heterogeneous scales. AIMeter [14] measures and visualizes energy consumption, carbon emissions, and hardware metrics in AI workloads. FECoM [30] measures energy consumed by TensorFlow APIs, accounting for execution duration, computational load, and thermal effects. These tools target general workload monitoring or API profiling, remaining agnostic to the distinct training stages and phases.

Characterizing training energy. McDonald et al. [21] study LLM training and inference across HPC platforms, proposing measurement techniques and tunable hardware and datacenter settings to reduce energy consumption. Go et al. [11] characterize how parallelism strategies interact with hardware power, utilization, and thermal behavior across platforms. Zhang et al. [37] relate parallelism configuration and GPU frequency scaling to performance, power, and energy. Latif et al. [19] measure the instantaneous node-level power draw of an 8× GPU H100 compute node during ResNet and Llama-2-13B training, reporting peak power and the effect of batch size on total training energy. Patel et al. [29] characterize power consumption patterns of LLM training and inference workloads in cloud datacenters to inform power oversubscription strategies, focusing on cluster-level power management. These works treat training as a single unit of analysis, rather than decomposing it into individual pipeline events.

Positioning. Unlike prior work, which profiles training as a monolithic phase, we characterize energy across the full AI training lifecycle, namely *preparation* (§IV-A), *execution* (§IV-B), and *post-execution* (§IV-C). Within execution, to the best of our knowledge, this is the first work to characterize the execution time and energy consumption of individual phases (§IV-B), rather than treating execution as a single unit of analysis, allowing us to identify energy consumption patterns and assess whether they generalize across workloads.

VI. CONCLUSION

We presented a characterization of the AI training lifecycle in HPC infrastructures, covering the preparation, execution, and post-execution stages, and, within execution, providing a granular breakdown of energy consumption and execution time for each individual phase. This granularity enables HPC users and practitioners to attribute energy consumption to its actual source within their training pipelines, and to quantify the impact of subsequent optimizations. Our results show that energy consumption is highly workload-dependent, that no single phase or optimization dominates across workloads, and that even seemingly minor stages, such as dataset and model transfers, can add up to a high, avoidable cost at the scale of shared HPC infrastructures. These findings not only demonstrate the need for workload- and phase-aware energy optimizations in HPC but also for sustainable, energy-conscious practices in AI development and use.

Future work. We plan to extend this characterization study to a broader range of models (*i.e.*, Llama, GPT, OPT) and parallelization strategies (*i.e.*, pipeline parallelism, mixture-of-experts), to assess how our findings generalize beyond the workloads studied in this paper. Another interesting exploration avenue is to consider additional HPC platforms with different storage, network, and accelerator configurations, given the infrastructure-dependent nature of some steps in the training lifecycle. Extending the lifecycle to include *model serving* as an additional stage would provide a more complete picture of AI's end-to-end energy footprint, given its relevance in modern computing infrastructures [13, 35]. Finally, we plan to characterize distributed training, where inter-node communication introduces additional, currently unmeasured energy costs beyond the single-node synchronization, as well as include other phases within training, such as data preprocessing [26, 27].

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback. This work was co-funded by national funds through FCT - Fundação para a Ciência e a Tecnologia, I.P., with the PhD grant 2025.03394.BDANA with DOI identifier <https://doi.org/10.54499/2025.03394.BDANA> (Beatriz Cepa); project EPI-CURE, with reference 101139786, funding from the EuroHPC Joint Undertaking under grant agreement No 101139786 (Alicia Oliveira); by the European Regional Development Fund (ERDF) through the Innovation and Digital Transition Programme (COMPETE 2030) under Portugal 2030, within the scope of the project CDMS, reference 17409 (COMPETE2030-FEDER-01193000) (Alexandre Fernandes); and by EuroHPC JU for awarding the project EHPC-DEV-2026D02-192 access to Deucalion hosted by CNCA, Portugal. This work was also done under the scope of Extra Exploratory Research Projects (PATA) from the UT Austin Portugal Program. The acquisition and operation of the EuroHPC super-computer is funded jointly by the EuroHPC Joint Undertaking, through the European Union's Connecting Europe Facility and

the Horizon 2020 research and innovation programme, as well as Portugal (Ricardo Macedo). Claude (Anthropic) was used to assist with proofreading and improving the grammar and clarity of the paper.

REFERENCES

- [1] ayuo. `hd_tmp`. https://huggingface.co/datasets/ayuo/hd_tmp, 2024. Hugging Face dataset.
- [2] banned-historical-archives. Banned historical archives. <https://huggingface.co/datasets/banned-historical-archives/banned-historical-archives>, 2024. Hugging Face dataset.
- [3] Mohak Chadha, Eishi Arima, Amir Raoofy, Michael Gerndt, and Martin Schulz. Sustainability in HPC: Vision and Opportunities. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*. ACM, 2023.
- [4] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *International Conference on Learning Representations*, 2020.
- [5] Cornell University. `arXiv` dataset. <https://www.kaggle.com/datasets/Cornell-University/arxiv>, 2024. Kaggle dataset.
- [6] Marco Dantas, Diogo Leitão, Peter Cui, Ricardo Macedo, Xinlian Liu, Weijia Xu, and João Paulo. Accelerating Deep Learning Training Through Transparent Storage Tiering. In *22nd IEEE International Symposium on Cluster, Cloud and Internet Computing*. IEEE, 2022.
- [7] Howard David, Eugene Gorbato, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. RAPL: Memory Power Estimation and Capping. In *16th ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 189–194, 2010.
- [8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [9] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. A Validation of DRAM RAPL Power Measurements. In *Proceedings of the Second International Symposium on Memory Systems*, page 455–470. ACM, 2016.
- [10] Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.
- [11] Seokjin Go, Joongun Park, Spandan More, Hanjiang Wu, Irene Wang, Aaron Jezghani, Tushar Krishna, and Divya Mahajan. Characterizing the Efficiency of Distributed Training: A Power, Performance, and Thermal Perspective. In *58th IEEE/ACM International Symposium on Microarchitecture*. ACM, 2025.
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [13] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, et al. Characterization of large language model development in the datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation*, pages 709–729. USENIX Association, 2024.
- [14] Hongzhen Huang, Kunming Zhang, Hanlong Liao, Kui Wu, and Guoming Tang. AIMeter: Measuring, Analyzing, and Visualizing Energy and Carbon Footprint of AI Workloads, 2025.
- [15] Hugging Face. Documentation images. <https://huggingface.co/datasets/huggingface/documentation-images>, 2024. Hugging Face dataset.
- [16] Mathilde Jay, Vladimir Ostapenko, Laurent Lefèvre, Denis Trystram, Anne-Cécile Orgerie, and Benjamin Fichel. An experimental comparison of software-based power meters: focus on CPU and GPU. In *23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, pages 106–118. IEEE, 2023.
- [17] KakologArchives. KakologArchives dataset. <https://huggingface.co/datasets/KakologArchives/KakologArchives>, 2024. Hugging Face dataset.
- [18] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 3(2), March 2018.
- [19] Imran Latif, Alex C. Newkirk, Matthew R. Carbone, Arslan Munir, et al. Single-Node Power Demand During AI Training: Measurements on an 8-GPU NVIDIA H100 System. *IEEE Access*, 13:61740–61747, 2025.

- [20] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, et al. Mlperf training benchmark. In *Proceedings of Machine Learning and Systems*, volume 2, pages 336–349, 2020.
- [21] Joseph McDonald, Baolin Li, Nathan Frey, Devesh Tiwari, et al. Great Power, Great Responsibility: Recommendations for Reducing Energy for Training Language Models. In *Findings of the Association for Computational Linguistics*, 2022.
- [22] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.
- [23] Jayashree Mohan, Amar Phanishayee, and Vijay Chidambaram. Check-Freq: Frequent, Fine-Grained DNN Checkpointing. In *19th USENIX Conference on File and Storage Technologies*, pages 203–216. USENIX Association, 2021.
- [24] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and Mitigating Data Stalls in DNN Training. *Proc. VLDB Endow.*, 14(5):771–784, 2021.
- [25] National Center for Advanced Computing. Deucalion Supercomputer. <https://deucalion.acnca.pt/>, 2026. Accessed: September 14th, 2026.
- [26] Rahma Nouaji, Stella Bitchebe, Ricardo Macedo, and Oana Balmau. MinatoLoader: Accelerating Machine Learning Training Through Efficient Data Preprocessing. In *Proceedings of the 21st European Conference on Computer Systems*, pages 2037–2053. ACM, 2026.
- [27] NVIDIA Corporation. DALI: Nvidia data loading library. <https://github.com/NVIDIA/DALI>, 2018. Accessed: September 14th, 2026.
- [28] NVIDIA Corporation. NVIDIA Management Library (NVML). <https://developer.nvidia.com/management-library-nvml>, 2024.
- [29] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrior, Nithish Mahalingam, and Ricardo Bianchini. Characterizing Power Management Opportunities for LLMs in the Cloud. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, page 207–222. ACM, 2024.
- [30] Saurabhsingh Rajput, Tim Widmayer, Ziyuan Shang, Maria Kechagia, et al. Enhancing Energy-Awareness in Deep Learning through Fine-Grained Energy Measurement. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–34, December 2024.
- [31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.
- [32] Salesforce. Salesforce/WikiText Dataset. <https://huggingface.co/datasets/Salesforce/wikitext>, 2016. Accessed: Dec 27th, 2025.
- [33] Arya Tschand, Arun Tejusve Raghunath Rajan, Sachin Idgunji, Anirban Ghosh, et al. MLPerf Power: Benchmarking the Energy Efficiency of Machine Learning Systems from μ Watts to MWatts for Sustainable AI. In *2025 IEEE International Symposium on High Performance Computer Architecture*, pages 1201–1216, 2025.
- [34] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, et al. Structured 3d latents for scalable and versatile 3d generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21469–21480. IEEE, 2025.
- [35] Yuxing Xiang, Xue Li, Kun Qian, Yan Zhang, et al. ServeGen: Workload Characterization and Generation of Large Language Model Serving in Production. In *23rd USENIX Symposium on Networked Systems Design and Implementation*, pages 1845–1859. USENIX Association, 2026.
- [36] An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- [37] Zhendong Zhang, Foteini Strati, and Ana Klimovic. Characterizing Energy and Performance for Distributed Training of Large Language Models. In *6th European Workshop on Machine Learning and Systems*, pages 208–217. ACM, 2026.
- [38] Dan Zhao, Siddharth Samsi, Joseph McDonald, Baolin Li, et al. Sustainable Supercomputing for AI: GPU Power Capping at HPC Scale. In *2023 ACM Symposium on Cloud Computing*. ACM, 2023.

Appendix: Artifact Description

Artifact Description (AD)

I. OVERVIEW OF CONTRIBUTIONS AND ARTIFACTS

A. Paper’s Main Contributions

- C_1 A **principled approach** to *characterize energy consumption of the AI training lifecycle in modern HPC infrastructures*, spanning preparation, execution, and post-execution stages.
- C_2 An **in-depth analysis** of the *energy consumption and execution time of the distinct phases of the training pipeline over I/O-bound and GPU-bound models*.
- C_3 A set of **empirical findings and takeaways** on the *workload- and infrastructure-dependent nature of AI training energy consumption*, intended to guide HPC users toward more energy-efficient training practices.

B. Computational Artifacts

- A_1 Public GitHub repository with the scripts required for reproducing the experiments: [dsrhaslab/aionhpc2026-watts-behind-weights](https://github.com/dsrhaslab/aionhpc2026-watts-behind-weights).

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1, C_2, C_3	Figures 2-5, Table 1

II. ARTIFACT IDENTIFICATION

Artifact A_1 corresponds to the complete experimental pipeline used to produce the paper’s results.

A. Computational Artifact A_1

Relation To Contributions

Artifact A_1 implements the full experimental pipeline described in the paper’s methodology (§III), comprising: *i*) the dataset and model acquisition scripts used to characterize the preparation and post-execution stages (C_1); *ii*) the instrumented training pipelines for ResNet-50 and Qwen3-8B, together with the custom energy monitoring and event-tracing toolkit, used to characterize the execution stage at phase-level granularity (C_2); and *iii*) the analysis scripts that produce the findings and takeaways reported throughout §IV (C_3).

Expected Results

Reproducing A_1 should yield the following, corresponding to Figs. 2–5 and Table I:

- Energy consumption and execution time of the dataset and model acquisition steps increase with dataset/model size (C_1, C_3).

- A per-phase breakdown of energy consumption and execution time for ResNet-50 and Qwen3-8B under varying data placement and checkpointing configurations (C_2).
- The corresponding CPU/GPU energy consumption split for both workloads (C_2, C_3).
- Energy consumption and execution time of the model upload step increase with model size (C_1, C_3).

Due to hardware, driver, and interconnect variability, absolute energy (J) and execution time (min) may differ from those reported in the paper, but similar trends are expected.

Expected Reproduction Time (in Minutes)

Step	Estimated time
Artifact Setup	≈45 min
Artifact Execution	≈3200 min
Artifact Analysis	≈1 min

The Artifact Execution time only accounts for the time of a single run of each experiment in §IV.

Artifact Setup (incl. Inputs)

Hardware: We used a server with 2× 64-core AMD EPYC processors, 4× NVIDIA A100 80GB GPUs, 512 GiB of memory, 500 GiB SATA SSD, 200 Gb/s interconnect to the PFS. Experiments can be conducted on servers with other specifications but with an equivalent configuration.

Software: We used Rocky Linux 8.5 (kernel 4.18.0), PyTorch 2.6.0, CUDA 11.8, NCCL 2.21.5, NVIDIA Management Library, Intel RAPL, and OpenTelemetry.

Datasets and models: Our experiments used five models downloaded from HuggingFace:

- ResNet-50 (25.6M): [microsoft/resnet-50](https://huggingface.co/microsoft/resnet-50);
- ELECTRA (110M): [google/electra-base-discriminator](https://huggingface.co/google/electra-base-discriminator);
- TRELLIS (1.1B): [microsoft/TRELLIS-image-large](https://huggingface.co/microsoft/TRELLIS-image-large);
- Qwen3 (8B): [Qwen/Qwen3-8B](https://huggingface.co/Qwen/Qwen3-8B);
- Gemma (27B): [google/gemma-4-26B-A4B-it](https://huggingface.co/google/gemma-4-26B-A4B-it).

We also used seven datasets from HuggingFace and Kaggle:

- hd_tmp (486 MB): [ayuo/hd_tmp](https://huggingface.co/ayuo/hd_tmp);
- WikiText (644 MB): [Salesforce/wikitext](https://huggingface.co/Salesforce/wikitext);
- The arXiv Dataset (2.06 GB): [Cornell-University/arxiv](https://huggingface.co/Cornell-University/arxiv);
- Documentation Images (6.51 GB): [huggingface/documentation-images](https://huggingface.co/huggingface/documentation-images);
- ImageNet (167 GB): [ILSVRC/imagenet-1k](https://huggingface.co/ILSVRC/imagenet-1k);
- KakologArchives (225 GB): [KakologArchives/Kakolog-Archives](https://huggingface.co/KakologArchives/Kakolog-Archives);
- Banned Historical Archives (549 GB): [banned-historical-archives/banned-historical-archives](https://huggingface.co/banned-historical-archives/banned-historical-archives)

Installation and Deployment: The energy monitoring tool requires GCC 13.2.0, CMake, PostgreSQL, cuDNN, and CUDA 11.8.0 on both the build and execution environments. Monitoring also requires access to `perf_event_open` and Intel RAPL interfaces, which requires either `sudo` privileges or adequate read permissions. Experiments were submitted to the cluster via `sbatch`. The training process and the energy monitoring server are deployed on the same compute node, while a separate controller node collects, aggregates, and preprocesses the energy and OpenTelemetry data. Further, we used a custom-built version of PyTorch 2.6.0, and the monitoring tool attaches to the training process via `LD_PRELOAD`, loading the energy-tracking functionality into the target process at runtime.

Artifact Execution

Execution proceeds via three dependent tasks, in the following order: $T1 \rightarrow T2 \rightarrow T3$.

T1 (Data staging). Download WikiText and ImageNet datasets to the shared PFS. For the execution stage experiments (§IV-B), additionally copy these datasets from the PFS to node local storage. This task produces the staged inputs of T2.

T2 (Execution). Depending on the experiment being reproduced, T2 corresponds to one of three benchmark types:

- i) Preparation/post-execution benchmarks: execute the download/upload scripts under monitoring, producing the raw energy and time measurements underlying Figs. 2 and 3.
- ii) Execution-stage training: launch the energy monitoring

server and controller, then run the instrumented ResNet-50 or Qwen3-8B training script for the selected configuration (*e.g.*, storage placement, checkpointing). The monitoring tool, attached via `LD_PRELOAD`, samples CPU (RAPL) and GPU (NVML) power at 50 ms intervals, while OpenTelemetry instrumentation reports phase-boundary events (data loading, memory transfer, forward/backward pass, synchronization, checkpointing, validation) to the controller node. Each configuration is repeated over 3 independent runs, for 5 epochs each, consistent with §III. T2 produces raw, phase-tagged energy and time traces per run.

T3 (Aggregation and analysis). The controller node aggregates and preprocesses the phase-tagged traces produced by T2. The analysis scripts then consume this preprocessed output to compute per-phase energy/time breakdowns, averaged across the 3 repetitions, and the CPU/GPU energy split, producing the data for generating the plots and table reported in the paper.

Artifact Analysis (incl. Outputs)

CPU (RAPL) and GPU (NVML) power samples are correlated offline with OpenTelemetry phase-boundary timestamps to attribute energy and time to each lifecycle phase. We provide scripts to merge the separately collected energy and tracing logs into a single aggregated dataset per configuration, from which per-phase energy and time are averaged across the 3 repetitions; Figures and Table I were then manually composed from this aggregated output.