Carbon-Aware Dynamic Parallelism for Distributed Training

WAFIK ABOUALIM, University of Pittsburgh, USA
STEPHEN LEE, University of Pittsburgh, USA

Training frontier language models combines data, tensor, and pipeline parallelism at scale to minimize total training time. While this maximizes throughput, it also drives substantial power consumption and carbon emissions, which vary with grid carbon intensity over the course of a multi-week pre-training run. We first characterize the power footprint of pre-training across model architectures and parallel configurations, and show that on a fixed topology, different architectures exhibit distinct power patterns as parallelism is varied. Building on this analysis, we show that a dynamic parallelism strategy, which scales down to leaner configurations when grid carbon intensity is high and scales up when it is low, can modestly reduce carbon emissions. We then introduce a carbon-aware profiler and a dynamic parallelism scheduler that jointly select configurations in response to real-time grid signals. We show that carbon-aware dynamic parallelism reduces emissions by up to 3.87% over the best static configuration, with savings varying across regions and workloads.

CCS Concepts: • **Hardware → Impact on the environment**; • **Computer systems organization → Parallel architectures**; • **Social and professional topics → Computing / technology policy**.

Additional Key Words and Phrases: carbon-aware computing, distributed training, dynamic parallelism, large language models

1 Introduction

Training large language models (LLM) is among the most carbon-intensive workloads in computing. Single training runs of published models now span hundreds to over ten thousand tonnes of CO_2eq. For example, Meta reports an estimated 11,390 tCO_2e of location-based emissions for the LLaMA-3.1 family [5], and OpenAI's GPT-3 (175B) is estimated to have emitted approximately 552 tCO_2e during training [4, 10, 12]. With AI workloads projected to dominate data center growth over the coming decade, reducing the per-run carbon footprint of pre-training has become a concern for both cloud providers and model developers.

Carbon-aware approaches to demand-side optimization broadly fall into three categories: temporal shifting, which delays workloads to periods of cleaner electricity supply [13, 18], spatial shifting, which migrates them to regions with greener grids [3], and elastic resource scaling, which varies the amount of allocated compute over time in response to carbon intensity [6]. These approaches are effective for short, elastic, loosely-coupled batch jobs, but each faces limitations for synchronous LLM pre-training. First, temporal shifting is difficult once a multi-week stateful pre-training run has started: high-carbon periods can persist for many hours, and pausing during these periods requires checkpointing terabytes of optimizer state and potentially releasing reserved GPU capacity, which can erode the carbon benefit. Second, spatial shifting is limited because synchronous gradient exchange does not tolerate wide-area communication latencies, so once a job has been launched on a particular cluster it is effectively pinned to that location for its full duration. Third, elastic resource scaling is conceptually closest to our work, but assumes that the job can vary its server or GPU allocation over time. In large reserved training clusters, however, releasing and later reacquiring scarce accelerators may be impractical. As a result, once admitted on a fixed allocation, the training job is exposed to a local grid whose carbon intensity varies by 3–5× over a 24-hour cycle [1], with no fixed-allocation mechanism to respond.

To overcome these drawbacks, we present a new approach that exploits the configurational elasticity of distributed training to reduce carbon emissions without moving the job or extending its completion time. LLM training partitions both the model and the data across many GPUs using several forms of parallelism. This paper focuses on three widely used axes: (i) data parallelism (DP), (ii) tensor parallelism (TP), and (iii) pipeline parallelism (PP). Frameworks such as Megatron-LM [11] support arbitrary combinations of the three. By convention, the (TP, PP, DP) configuration is selected once at job launch to maximize throughput, and held fixed for the entire run. Our approach, carbon-aware dynamic parallelism, instead treats the configuration as a tunable operating point that varies over time. The idea is analogous to dynamic voltage and frequency scaling (DVFS) in CPUs. Rather than fixing one operating point, the system selects among several equivalent-quality configurations along a power-throughput Pareto curve. In essence, dynamic parallelism switches to lower-power layouts during high-carbon hours and to higher-throughput layouts during clean hours, recovering progress later in the run so that the token budget and deadline are preserved.

Designing a carbon-aware dynamic parallelism system requires addressing two design questions: *which* configurations are available for a given (model, hardware) pair, and *when* each should be used over the course of a run. Our profiling study across three model families on single-node and multi-node deployments shows that no fixed ordering holds. Tensor parallelism minimizes average power on a single node, but loses that property as cross-node communication grows with scale. Pipeline-heavy layouts become the lowest-power option at higher node counts, and the ordering changes again across model families on the same hardware. The second is a constrained scheduling problem. We minimize carbon emissions subject to a token budget, deadline, and switching overhead, under two information assumptions: an oracle with full trace access, and a greedy online policy using only the current hour's intensity. Because end-to-end multi-week pre-training runs across every (model, schedule, trace) combination are infeasible, we evaluate by empirically profiling the configurations and simulating the configuration switches against different carbon traces. Our contributions are as follows:

- We characterize parallelism configuration as a carbon control knob and profile (TP, PP, DP) configuration across different model families on single- and multi-node deployments. We show that the power-minimizing configuration differs across architectures and changes with system scale.
- We design a carbon-aware dynamic parallelism system on top of Megatron-LM, consisting of an offline profiler that

measures each configuration's operating point and switching cost, and a carbon-aware scheduler with three policies: two oracles (fixed-start and job-shifting) that bound achievable savings, and a greedy policy that uses only the current hour's carbon intensity.
- We implement and evaluate the system on real GPU profiles. Across GPT-2 and Llama 1B workloads with 1–8B tokens under matched deadline slack, fixed-start dynamic parallelism reduces emissions by up to 3.87% relative to the episode-best feasible static configuration within the same fixed GPU allocation.

2 Background and Related Work

2.1 Parallelism Configurations

LLM training jobs partition both the model and the data across hundreds to thousands of GPUs. There are three main forms of parallelism that are typically combined [11]. (i) *Data parallelism* (DP) replicates the model across GPUs and splits the batch, synchronizing gradients after each step. (ii) *Tensor parallelism* (TP) splits each layer's weights across multiple GPUs, which must communicate with each other every time a layer is computed. (iii) *Pipeline parallelism* (PP) splits the model into sequential stages, assigns each stage to a different GPU, and feeds small chunks of the batch (called *micro-batches*) through the stages so that all GPUs can work concurrently. Different combinations of these choices can reach the same final model quality. However, as our results show, they exhibit different throughput and energy characteristics, which can be leveraged to reduce the carbon footprint of training.

2.2 Reconfiguring Parallelism Configurations

Changing a training job's parallelism configuration requires repartitioning model parameters and optimizer state, re-establishing collective communication groups, and resuming the optimizer from a consistent state. The standard approach, supported by widely-used frameworks such as Megatron-LM [11], is *checkpoint and restart*. The job writes a checkpoint to disk, terminates, restarts with the new configuration, and reloads its state into the new GPU layout. Recent work has focused on reducing this overhead through in-place reshard primitives that avoid the disk round-trip [7, 8]. In our work, we use Megatron-LM, as it provides support for all parallelism strategies and works well with our existing training stack. Faster reconfiguration strategies (e.g., ElasWave [8]) are complementary to our approach and would only improve the overall throughput and carbon savings.

2.3 Related Work

Distributed Training. Recent work has explored both training efficiency optimization and dynamic parallelism reconfiguration. Zeus [19] reduces energy consumption by adjusting batch size while preserving convergence. Other work formulates parallelism selection as an optimization problem to minimize training time under a fixed compute budget [9]. Systems such as Varuna [2] and Tenplex [17] enable elastic parallelism across DP, TP, and PP through checkpointing and tensor reshaping. More recent work further reduces reconfiguration overhead through efficient rank-to-rank resharding between GPUs, significantly decreasing parallelism switching time [8]. However, these approaches are driven by performance or resource considerations. In contrast, our work treats carbon as the primary optimization objective and adapts parallelism in response to a time-varying grid signal.

Carbon-Aware Computing. Carbon intensity, measured in gCO_2e/kWh, reflects the emissions associated with electricity generation. It varies over time as the energy mix changes, often fluctuating significantly within a single day. In our evaluation, we use a real grid trace that spans both high- and low-carbon periods.

Prior work reduces carbon through several forms of demand-side adaptation. Temporal and spatial shifting delay flexible jobs to cleaner periods or move computation across regions [13, 15, 16, 18]. Elastic resource-scaling systems such as CarbonScaler dynamically vary the amount of allocated compute in response to carbon intensity [6]. Other work adjusts power usage at the machine level, for example by tuning batch size or power limits to reduce energy consumption [19]. These approaches are complementary, but target different control knobs. Temporal and spatial shifting are difficult for long, stateful LLM pre-training jobs once they have started, and machine-level tuning leaves the parallelism layout fixed. CarbonScaler is conceptually closest to our work because it adapts a power-throughput knob over time, but it does so by changing the resource allocation. In contrast, we study a fixed-allocation setting: the job keeps the same GPUs and treats the parallelism configuration itself as the carbon control knob, adapting it during the training run.

3 Profiling Parallelism Configurations

This section characterizes how average power and throughput vary across the valid parallelism (TP, PP, DP) configurations of a model on a fixed topology and across node counts in multi-node deployments, and assesses whether the resulting power-throughput tradeoffs differ across model architectures and hardware setups.

3.1 Setup

We conduct experiments in both single-node and multi-node settings using NVIDIA L40 and A100 GPUs. Distributed training is implemented using Megatron Core [14], and our dynamic parallelism prototype targets the single-node setting.

The single-node study uses one node with 4 NVIDIA L40 GPUs and the multi-node study uses NVIDIA A100 nodes with 4 GPUs each, connected over the cluster fabric. We evaluate three model families (GPT-2, Llama-style, and Pythia) at multiple sizes. For each (model, configuration) pair, we record steady-state power, throughput, energy per step, and energy per token. Power and energy are measured at the GPU level across both initialization and steady-state training phases.

We conduct experiments across all valid parallel configurations. A parallel configuration is a triple (TP, PP, DP) specifying the degree of tensor, pipeline, and data parallelism. Given a world size W (the total number of GPUs in the job), every GPU must occupy exactly one position in the decomposition. So, a valid configuration satisfies

$TP \times PP \times DP = W$. In the 4-GPU setting this yields six configurations: (4, 1, 1), (1, 4, 1), and (1, 1, 4) are pure tensor, pipeline, and data parallel, respectively. While (2, 2, 1), (2, 1, 2), and (1, 2, 2) combine two axes.

3.2 Results

Single Node. Figures 1a and 1b show the average power and throughput across the six valid configurations within a single 4-GPU node and for each model family. We observe that the tensor parallel configuration (TP=4) achieves the lowest average power across every model in our profile. However, this low power behavior comes at the cost of throughput. TP=4 is also the slowest configuration in tokens per second, and thus, may take longer to complete a fixed training budget. Among the remaining five configurations, average power differs substantially both within a model family and across families at a fixed configuration. For instance, in the Llama family the PP=4 configuration draws higher steady-state power than (TP=2,PP=2), whereas in Pythia the ordering is reversed.

Insight: While tensor parallelism achieves the lowest average power, it also has lower throughput. A carbon scheduler can exploit tensor parallelism during high carbon hours and a higher throughput configuration during cleaner ones. Furthermore, the power and throughput profiles vary across model families, so the scheduler must profile the configurations and cannot assume a fixed order.

Multi-Node. Figure 2 shows the steady-state power and throughput across valid configurations as we scale the deployment from 2 to 8 A100 nodes for GPT-2 Medium. As shown, tensor parallelism does not always have the lowest power setup. As the node count grows, the cross-node communication cost of tensor parallelism rises faster than that of pipeline or data parallelism, and the minimum-power configuration may change. At n=6, for example, a configuration that maximizes the pipeline parallelism degree achieves lower average power than maximizing the tensor-parallel layout. Similarly, the maximum throughput configuration varies with the node count.

Insight: The single-node finding that TP=4 minimizes power does not generalize to multi-node deployments. Both the minimum-power and the maximum-throughput configurations depend on the deployment scale. Importantly, GPU power draw is not determined by the hardware platform alone. These results motivate treating power as a measured property of a training workload, rather than assuming a fixed per-GPU value from the device specification.

4 Carbon-aware Dynamic Parallelism Scheduler

Our goal is to reduce the carbon emissions of a single training run by switching parallel configurations in response to time-varying grid carbon intensity, without missing the training deadline. The system has two components: (i) *profiler* that profiles the energy-throughput tradeoff of training configurations, and (ii) a *carbon-aware scheduler* that consumes the profile and a grid trace and produces a sequence of configurations over time.

4.1 Profiler

For each model and hardware setup, the profiler runs a short Megatron-LM job for each feasible configuration $\pi = (TP, PP, DP)$. It records steady-state power P_π and throughput r_π, derives per-token energy, and measures an average switching cost $\bar{K}$ for switching between configurations. Throughput is taken directly from Megatron-LM's reported training metrics, while power is measured through NVML and aggregated across the GPUs assigned to the run. We use an average $\bar{K}$ rather than a pairwise configuration because pairwise measurements are dominated by filesystem and page-cache noise. That said, the average is a good approximation for the switching cost. The dominant cost of a switch is the checkpoint write and state reload, which is largely invariant to the source-target configuration pair.

Note that profiling is a one-time cost per (model, hardware) pair. Profiling can be performed offline and incurs negligible overhead. In our experiments, each feasible configuration takes a few minutes to reach steady state and collect enough samples for stable estimates of power and throughput. Each profiling run begins with a 10-to-60-second warmup that terminates early once the mean step time of the last 10 iterations is within 2% of the preceding 10.

4.2 Scheduler

The scheduler decides when each parallel configuration is used over the lifetime of the job. Its inputs are a token budget N, a deadline D, an hourly carbon-intensity trace or forecast CI_t, and the profiled $(P_\pi, r_\pi, \bar{K})$. We discretize time into windows of length Δt and introduce two decisions: a delayed start $s \in \{0, \Delta t, 2\Delta t, \ldots, S\}$ and an assignment $x_{t,\pi} \in \{0, 1\}$ that is 1 if configuration π is used in window t, with at most one configuration active per window. Let $a_t = \sum_{\pi \in \Pi} x_{t,\pi} \in \{0, 1\}$ be the active indicator for window t, and let m count the windows in which the active configuration changes from the previous window. The scheduler solves

$$\min_{s,x} \quad \sum_t \sum_{\pi \in \Pi} x_{t,\pi} P_\pi \Delta t \cdot \mathrm{CI}_{s+t} \tag{1}$$

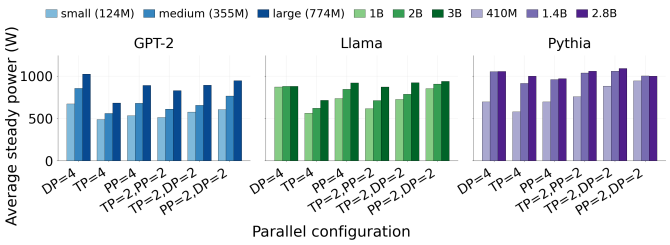
$$\text{s.t.} \quad \sum_t \sum_{\pi \in \Pi} x_{t,\pi} r_\pi \Delta t \;\geq\; N, \tag{2}$$

$$s + \sum_t \Delta t \cdot a_t + m\bar{K} \;\leq\; D, \tag{3}$$

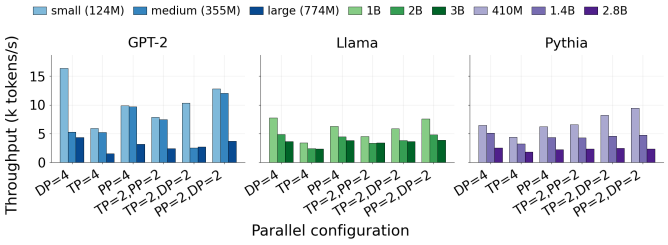
$$\sum_{\pi \in \Pi} x_{t,\pi} \;\leq\; 1 \quad \forall t. \tag{4}$$

The objective minimizes total carbon emissions. Constraint (2) forces the schedule to reach the token budget; constraint (3) bounds completion time and charges the wait s, the active windows, and the switching overhead; constraint (4) enforces a single active configuration per window. We assume a waiting job does not consume GPU energy, so s does not appear in the objective. The scheduler trades throughput for power during high-carbon hours, but only when the lost progress is recoverable before the deadline.

Variants. We instantiate the scheduler in two variants that differ only in how s is treated. (i) *Fixed-Start Dynamic Parallelism* fixes $s = 0$ and chooses only the configuration sequence x. This captures the case where a training job has already been admitted to the cluster but can still adapt its parallel layout over time. (ii) *Job Shifting + Dynamic Parallelism* treats s as a free decision in the shift window, combining two carbon-aware controls: shifting the entire job toward cleaner grid periods and adapting the parallel configuration during the run.

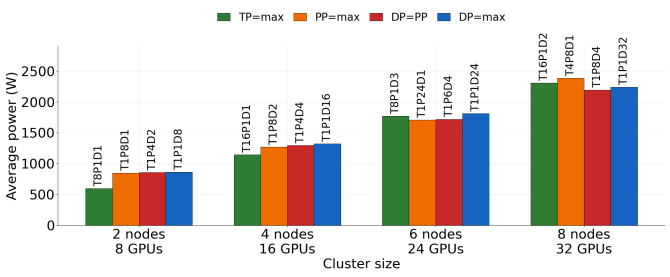


(a) Average power draw across different LLM families and configurations.

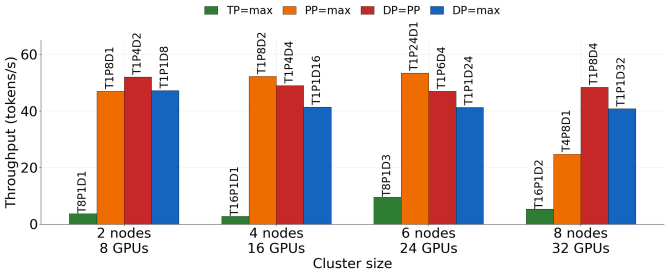


(b) Throughput across different LLM families and configurations.

Fig. 1. Power and throughput landscapes across model families and parallel configurations.



(a) Average power draw.



(b) Average throughput.

Fig. 2. Scaling behavior of different parallel strategies for GPT-2 Medium across node counts, with 4 GPUs per node. Panel (a) shows average power draw, and panel (b) shows average throughput.

4.3 Greedy Online Scheduler

The constrained scheduler assumes access to the full carbon-intensity trace. To evaluate dynamic parallelism under more realistic information assumptions, we implement a *greedy online scheduler* that uses only the current hour's carbon intensity. Greedy uses the same profile table $(P_\pi, r_\pi, \bar{K})$ and starts from the highest-throughput strategy. At each hourly decision point, it considers switching to a lower-power strategy when the current hour is dirty. The switch is taken only if the carbon saved in the current interval exceeds the estimated cost of catching up later, including the switching overhead $\bar{K}$.

Implementation. The profiler is implemented on top of Megatron-LM and measures the per-configuration operating points $(P_\pi, r_\pi, \bar{K})$ once per (model, hardware) pair. Because executing full multi-week pre-training runs for every (model, schedule, trace) combination is infeasible under our compute budget, we built a simulator that estimates end-to-end carbon savings by replaying schedules against historical grid traces using the empirically measured operating points. For each (policy, episode) pair, the simulator records total carbon emissions, total energy, training wall time, wait time, completion delay from arrival, number of reshards, token completion, and deadline feasibility. Carbon savings are reported relative to the paired static reference, defined as the lowest-carbon static configuration that meets the deadline on the same trace window.

5 Evaluation

5.1 Experimental Setup

Dataset. Our dataset consists of hourly carbon-intensity traces from May 2024 for three AWS data center locations: `us-west-1`, `us-west-2`, and `us-east-2`, obtained from Electricity Maps[1]. As shown in Figure 3, these regions span different grid mixes and carbon-intensity patterns during this month, which create opportunities for a dynamic parallelism scheduler to align low-power configurations with dirty hours and high-throughput configurations with clean ones.

Methodology. We evaluate two profiled model/hardware setups: GPT-2 on an 8× A100 setup, and Llama 1B on a 4× L40 setup. An *episode* is a simulated training job replayed against a sampled carbon-trace window, characterized by an arrival time, model, token budget, deadline, and profiled set of feasible (TP, PP, DP) configurations. We use May 2024 hourly carbon traces for AWS `us-west-1`, `us-west-2`, and `us-east-2`, sampling 64 arrival offsets for each model, region, and token budget.

We sweep token budgets from 1B to 8B tokens in 1B increments. GPT-2 uses sequence length 1024 on 8 GPUs, while Llama 1B uses sequence length 256 on 4 GPUs; both use global batch size 16. For each model and token budget, the deadline is set to 25% slack over the fastest measured static strategy rather than fixed at 200 hours. The scheduler uses simulated switching costs and a token quantum chosen by token budget. Carbon savings are computed relative to

[1] https://www.electricitymaps.com/

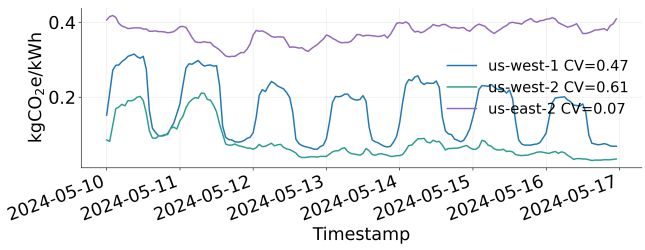


Fig. 3. Hourly carbon intensity across AWS regions over a representative 7-day window in May 2024. The legend reports each region's coefficient of variation (CV) over the displayed window. The traces show both regional differences and temporal variability, motivating carbon-aware schedulers that adapt training execution across time and region.

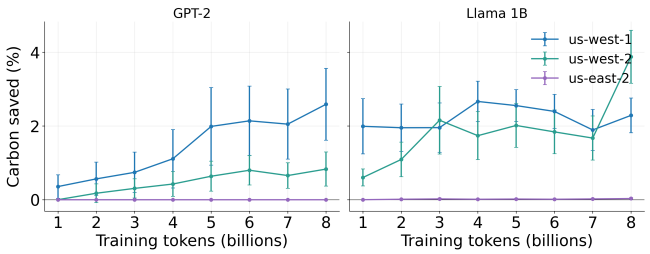


Fig. 4. Fixed-start dynamic parallelism savings relative to Best Static under matched deadline slack. Each workload uses a deadline rounded up to the next whole hour after setting a target of 125% of the best-static runtime. Bars show mean percent carbon reduction; error bars show 95% paired bootstrap confidence intervals across episode offsets.

the episode-best deadline-feasible static parallelism configuration for the same model, region, token budget, and arrival window.

Policies. We compare dynamic policies against the best static policy to quantify the carbon benefit of switching parallel configurations in response to a time-varying grid signal. (i) *Best Static* is the lowest-carbon static configuration that completes the same token budget within the deadline on the same trace window. (ii) *Greedy* is the online scheduler that uses only the current hour's carbon intensity. (iii) *Fixed-Start Dynamic Parallelism* is the constrained scheduler with arrival fixed. (iv) *Job Shifting + Dynamic Parallelism* is the scheduler with an additional decision over a delayed start s. The two constrained schedulers have access to the full carbon-intensity trace and approximate an upper bound on the carbon savings achievable under perfect information.

Metric. We report carbon savings as the percent reduction in total CO2eq emissions of a dynamic policy relative to Best Static on the same workload, trace window, and deadline. Positive values indicate that the dynamic policy emits less carbon than the lowest-carbon static configuration meeting the same deadline.

5.2 Static vs. Dynamic Configuration

We analyze whether dynamic parallelism saves carbon over the best static configuration that meets the same deadline. To do so, we first isolate the value of dynamic parallelism from the value of delaying a job. In this experiment, every job starts immediately at its sampled arrival offset; the scheduler can only change the TP/PP/DP configuration during the run. All workloads use the May 2024 hourly carbon trace and 64 paired episode offsets. For each workload, we set the deadline to $\lceil 1.25 \times T_{\text{static}} \rceil$, where T_{static} is the mean runtime of the best static configuration. This gives the dynamic policy approximately 25% relative slack over the best-static runtime while keeping the start time fixed.

Best Static is the completed static TP/PP/DP configuration with the lowest mean carbon for the same workload and deadline. Fixed-start dynamic parallelism uses the same arrival offsets and simulated switching costs, but may switch parallel configurations over time. We also evaluate a Greedy policy that switches using only the current hour's carbon intensity. Greedy does not see the future carbon trace, but it can still beat Best Static on some episodes.

Figure 4 shows that fixed-start dynamic parallelism provides modest carbon savings relative to the episode-best feasible static configuration, with gains concentrated in regions whose carbon intensity varies substantially over time. Across the 1B–8B token sweep, GPT-2 on 8× A100 achieves its largest savings in `us-west-1` at the 8B-token budget, reducing mean carbon from 6160.82 gCO_2e to 5995.32 gCO_2e, a 2.59% reduction. GPT-2 savings are smaller in `us-west-2`, reaching 0.83% at 8B tokens, and are effectively zero in `us-east-2`.

For Llama 1B on 4× L40, the largest gain occurs in `us-west-2` at the 8B-token budget, reducing mean carbon from 11598.43 gCO_2e to 11126.13 gCO_2e, a 3.87% reduction. Llama 1B also saves up to 2.66% in `us-west-1`, while `us-east-2` again shows almost no benefit (maximum 0.03%). These results indicate that dynamic parallelism is most useful when the trace contains enough temporal carbon variation to make switching worthwhile under the deadline constraint.

Figure 5 shows that Greedy can beat Best Static in individual episodes, even though it uses only a short-horizon carbon heuristic. The greedy policy switches to leaner TP=1, PP=4 and TP=2, PP=4 configurations during high-intensity periods and to a higher-throughput TP=1, PP=1 configuration during clean ones, exploiting the power-throughput tradeoff.

The Oracle (*Fixed-Start Dynamic Parallelism*) achieves higher savings by selecting a schedule that mostly alternates between `TP=1, PP=1` and `TP=1, PP=4`, with a short `TP=2, PP=4` interval during dirtier hours. This does not mean Greedy is better on average: because it does not optimize over the full trace, throughput loss, and switching overheads, it can also make poor decisions. Nonetheless, this episode demonstrates that dynamic reconfiguration can yield carbon benefit even without full-trace optimization.

6 Conclusion

We introduced carbon-aware dynamic parallelism, an approach that treats the (TP, PP, DP) configuration as a carbon control knob for large-scale pre-training. Profiling across three model families and two hardware setups showed that the power-throughput tradeoff differs across architectures and changes with system scale, so no fixed configuration dominates. Our scheduler exploits this heterogeneity by switching configurations in response to a time-varying grid signal, and under matched deadline slack reduces carbon emis-

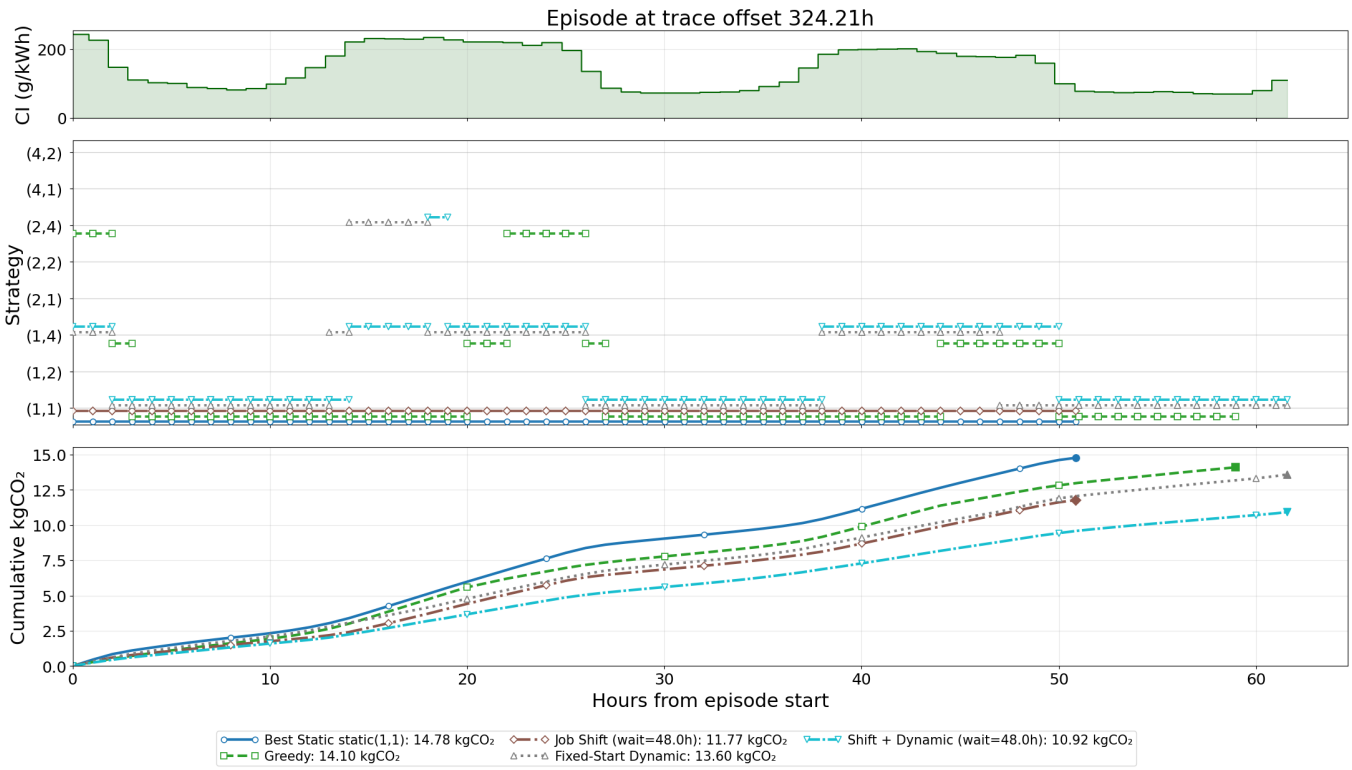


Fig. 5. Example optimization episode from the May 2024 us-west-1 trace for the 20B-token GPT-2 workload with a 120-hour deadline. Best Static selects TP=1 , PP=1 and emits 14.78 kgCO_2e. Greedy reduces emissions to 14.10 kgCO_2e, a 4.6% reduction, by switching configurations during the run. Fixed-start dynamic parallelism further reduces emissions to 13.60 kgCO_2e, while job shifting and job shifting plus dynamic parallelism reduce emissions to 11.77 kgCO_2e and 10.92 kgCO_2e, respectively.

sions over the episode-best feasible static configuration by up to 3.87% across 1–8B-token workloads. Our results also show that grid mix shapes the achievable carbon savings, with larger savings in regions with stronger carbon-intensity variation. We also expect to close the gap between the greedy online policy and the oracle through more sophisticated online algorithms, such as model predictive control over short-horizon carbon forecasts.

Acknowledgments

This work is supported in part by NSF grant #2324873 and Department of Energy #DECR0000041. The authors would also like to acknowledge the Mascaro Center for Sustainable Innovation at the University of Pittsburgh for its support. This research was also supported in part by the University of Pittsburgh Center for Research Computing and Data, RRID:SCR_022735, through the resources provided. Specifically, this work used the H2P cluster, which is supported by NSF award number OAC-2117681.

References

[1] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. 2023. Carbon explorer: A holistic framework for designing carbon aware datacenters. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 118–132.

[2] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. 2021. Varuna: Scalable, Low-cost Training of Massive Deep Learning Models. arXiv:2111.04007 [cs.DC] https://arxiv.org/abs/2111.04007

[3] Jesse Dodge, Taylor Prewitt, Remi Tachet des Combes, Erika Odmark, Roy Schwartz, Emma Strubell, Alexandra Sasha Luccioni, Noah A Smith, Nicole DeCario, and Will Buchanan. 2022. Measuring the carbon intensity of ai in cloud instances. In *Proceedings of the 2022 ACM conference on fairness, accountability, and transparency*. 1877–1894.

[4] Ahmad Faiz, Sotaro Kaneda, Ruhan Wang, Rita Osi, Prateek Sharma, Fan Chen, and Lei Jiang. 2024. Llmcarbon: Modeling the end-to-end carbon footprint of large language models. In *International Conference on Learning Representations*, Vol. 2024. 24727–24741.

[5] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[6] Walid A Hanafy, Qianlin Liang, Noman Bashir, David Irwin, and Prashant Shenoy. 2023. Carbonscaler: Leveraging cloud workload elasticity for optimizing carbon-efficiency. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 7, 3 (2023), 1–28.

[7] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. 2024. {MegaScale}: Scaling large language model training to more than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 745–760.

[8] Xueze Kang, Guangyu Xiang, Yuxin Wang, Hao Zhang, Yuchu Fang, Yuhang Zhou, Zhenheng Tang, Youhui Lv, Eliran Maman, Mark Wasserman, et al. 2025. ElasWave: An Elastic-Native System for Scalable Hybrid-Parallel Training. *arXiv preprint arXiv:2510.00606* (2025).

[9] Zongbiao Li, Xiezhao Li, Yinghao Cui, Yijun Chen, Zhixuan Gu, Yuxuan Liu, Wenbo Zhu, Fei Jia, Ke Liu, Qifeng Li, Junyao Zhan, Jiangtao Zhou, Chenxi Zhang, and Qike Liu. 2024. Automatically Planning Optimal Parallel Strategy for Large

Language Models. arXiv:2501.00254 [cs.AI] https://arxiv.org/abs/2501.00254

[10] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. 2023. Estimating the carbon footprint of bloom, a 176b parameter language model. *Journal of machine learning research* 24, 253 (2023), 1–15.

[11] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 1–15.

[12] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350* (2021).

[13] Ana Radovanović, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, et al. 2022. Carbon-aware computing for datacenters. *IEEE Transactions on Power Systems* 38, 2 (2022), 1270–1280.

[14] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL] https://arxiv.org/abs/1909.08053

[15] Abel Souza, Noman Bashir, Jorge Murillo, Walid Hanafy, Qianlin Liang, David Irwin, and Prashant Shenoy. 2023. Ecovisor: A virtual energy system for carbon-efficient applications. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 252–265.

[16] Thanathorn Sukprasert, Abel Souza, Noman Bashir, David Irwin, and Prashant Shenoy. 2024. On the Limitations of Carbon-Aware Temporal and Spatial Workload Shifting in the Cloud. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys '24)*. ACM, 924–941. doi:10.1145/3627703.3650079

[17] Marcel Wagenländer, Guo Li, Bo Zhao, Luo Mai, and Peter Pietzuch. 2024. Tenplex: Dynamic Parallelism for Deep Learning using Parallelizable Tensor Collections. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles (SOSP '24)*. ACM, 195–210. doi:10.1145/3694715.3695975

[18] Philipp Wiesner, Ilja Behnke, Dominik Scheinert, Kordian Gontarska, and Lauritz Thamsen. 2021. Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud. In *Proceedings of the 22nd International Middleware Conference*. 260–272.

[19] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. 2022. Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training. arXiv:2208.06102 [cs.LG] https://arxiv.org/abs/2208.06102