Investigating the Energy and Latency Tradeoff in Rack Scale Computing

Ruth Lu
ruthluvu@mit.edu

April Hu
aprilhu@mit.edu

Fucheng Warren Zhu
wzhu@college.harvard

I. Introduction

Large-scale AI models, particularly transformers, have demonstrated remarkable capabilities and are driving significant economic value across industries [1]. However, the training of these state-of-the-art models presents substantial computational challenges. Models like Meta's Llama 4 Behemoth, with trillions of parameters trained on vast datasets (e.g., 30 trillion tokens), far exceed the capacity of individual hardware accelerators [2]. For instance, a transformer with $100\,\mathrm{B}$ parameters requires approximately $800\,\mathrm{GB}$ of device memory for training in `FP16`[1], significantly more than the $80\,\mathrm{GB}$ available on an NVIDIA A100 GPU [4] or even the projected capacity of next-generation hardware like Blackwell, with up to $384\,\mathrm{GB}$ [5].

Consequently, training these models necessitates massive parallelism. Data parallelism is required to process enormous datasets efficiently, while model parallelism becomes essential to distribute the model parameters themselves across multiple devices [2]. Achieving feasible training times and maintaining competitive advantage requires deploying thousands of GPUs concurrently. Production models like GPT-4 were trained on clusters of approximately 20,000 A100 GPUs [6], and Llama 4 utilized 32,000 H100s [2]. At small scales, the MFU (model flop utilization) often ranges from 35-45% [7]. At larger scales, it could be even lower, as in the case of Llama4 (40-5% depending on the precise precision mix) [2]. This escalating scale underscores the critical importance of understanding and optimizing rack-scale computing environments.

While performance scaling through parallelism is crucial, the associated energy consumption has emerged as a primary bottleneck for continued AI progress [8]. Training large models is incredibly energy-intensive; GPT-3 training, for example, consumed an estimated 1,287 MWh [9], [10]. Datacenter power capacity directly limits the number of GPUs that can be hosted, with large AI initiatives requiring dedicated power infrastructure, such as XAI's reported 150MW power procurement [11]. Furthermore, power efficiency often exhibits diminishing returns as systems scale, exacerbating the challenge [12].

Previous research has investigated various parallelism strategies [13] and techniques for latency hiding or reducing communication overhead [14], [15]. However, many studies focus primarily on workload performance (latency and throughput) without explicitly modeling the energy implications, or they rely on handcrafted optimizations requiring significant domain expertise. Directly searching the vast design space of parallelism strategies, hardware configurations, and workload mappings for optimal energy-latency points is computationally prohibitive.

Therefore, there is a pressing need for analytical tools that can systematically explore the complex trade-offs between energy consumption and latency in rack-scale systems. By developing models that capture both computation and communication costs, including energy, we can enable more efficient co-design of hardware architectures and software mapping strategies. This work aims to bridge this gap by extending analytical frameworks like Timeloop [16] and LoopTree [17] to model rack-scale parallelisms, providing a foundation for automated exploration and optimization of energy and latency in large-scale AI training.

II. Related Works

A. Parallelisms in Machine Learning

Data Parallelism: Data parallelism (DP) replicates the model, shards the input batch, and reconciles weight updates with an `all-reduce`. It offers near-linear throughput scaling for modest layer sizes and keeps the communication surface limited to weight gradients—making it attractive as a first target for rack-scale studies. Ring `all-reduce` implementations in Megatron-LM [18] and ZeRO/FSDP [19] scale to hundreds of GPUs with $> 85\%$ efficiency.

Model Parallelism: When a single layer exceeds GPU memory, the work must be split *within* the layer:

- **Tensor parallelism** tiles individual matrix operands across GPUs; Megatron's intra-layer strategy set the standard practice [18].
- **Pipeline parallelism** partitions the layer stack into stages; GPipe [20] and PipeDream-2BW [21] showed how weight-versioning and 1F1B scheduling hide bubble overhead.
- **Expert parallelism** activates only a sparse subset of parameter "experts" per token; the Switch-Transformer trains 1.6 T-parameter models with the same FLOP budget as dense baselines [22].

In this paper we focus on DP, Tensor Parallelisms, and two-stage pipeline parallelism as the lowest-communication points in the design space. Notably, we do not model expert parallelism, and we leave it to future work.

[1] 2 bytes/parameter for weights, optimizer states, and gradients [3].

B. Interconnect Technologies

Training clusters exhibit a heterogeneous network hierarchy: on-package coherence fabrics such as NVLink and Infinity Fabric provide hundreds of GB/s at $\sim 1\ \mathrm{pJ/bit}$ [23]–[25]; board- and node-level crossbars (e.g. NVSwitch) sustain multi-TB/s bisection bandwidth [24]; rack- and datacenter-scale fabrics like InfiniBand NDR and Ethernet/RoCE deliver 400–800 Gb/s links with higher latency and energy $>10\ \mathrm{pJ/bit}$ [26]; and SmartNIC-based in-network aggregation (SHARP) or programmable-dataplane approaches (SwitchML, NetReduce) offload reductions into switches or NICs for 2–5$\times$ lower collective latency [27]–[29]. Accurately capturing this heterogeneity is crucial for modeling cluster-scale energy–delay trade-offs.

C. Energy Efficient Machine Learning

Energy efficiency in machine learning is becoming crucial due to significant environmental impacts [30]—as well as practical limits like datacenter power caps and edge device battery constraints [31], [32]. Consequently, a rich line of work focuses on designing energy-efficient ML accelerators [33]. Parallel efforts on the algorithmic side further reduce energy demands through methods like structured pruning and quantization [34], sparse expert routing [22], and mixed-precision training [35]. Despite the importance of energy usage, previous work in parallelism search [13], [14] has not taken it into consideration as they do not model the architecture.

D. Accelerator Simulation and Design Space Exploration

Need for joint mapping & architecture search.: Energy and latency vary by $>20\times$ across legal loop-nests for the *same* layer on a fixed array [16]. Accurate rack-scale evaluation therefore requires (i) a language to express hierarchical topologies and (ii) a mapper that searches software tilings jointly with hardware parameters.

Tool chain overview:

Timeloop	analytical performance model + mapper for dense DNN accelerators [16].
Accelergy	plug-in energy estimator that achieves $>95\%$ accuracy versus post-layout silicon and shares component libraries with Timeloop [**?**].
Sparseloop	extends Timeloop's design-space taxonomy to sparse tensor accelerators and shows $> 2000\times$ speed-ups over RTL simulation while keeping $< 8\%$ error [36].
CiMLoop	introduces voltage-aware models for compute-in-memory arrays and co-optimises circuit/architecture decisions [37].
LoopTree	models fused-layer dataflows and demonstrates up to $10\times$ buffer-area savings via inter-layer tiling [17].

Research gaps.: None of the above frameworks natively capture the collective-communication operators (e.g., `all-reduce`, pipeline stage-to-stage links) that dominate rack-scale LLM training. Our project bridges this gap by (i) casting DP and pipeline schedules as timeloop workloads, and (ii) extending Timeloop's cost model with calibrated NVLink/InfiniBand terms.

III. Experiment Setup

We provide a hierarchical YAML description (rack $\rightarrow$ GPU/PE $\rightarrow$ block $\rightarrow$ thread) including register files, SRAM/L2, GPU-local DRAM, global DRAM, and an abstracted NVLink/InfiniBand interconnect. This model is compatible with Accelergy for energy estimation [38]. We assume that RDMA is used to send data directly from one GPU to another GPU, bypassing the host memory buffer [39].

To evaluate how different parallelism strategies and hardware configurations affect compute and communication latency: (i) a two-layer feed-forward network and (ii) the attention block of Llama-3-8B (hidden size $d_{model} = 4096$) with softmax removed [40]. For each we supply (a) a LoopTree problem specification, (b) LoopTree architecture specification, and (c) parallel mappings that implement ring all-reduce (data parallel) and 2-stage pipeline parallelism. We use a sequence length of $S = 2048$. The SwiGLU activation function's communication cost within the MLP is ignored, so was softMax in self-attention as they do not highlight the crucial properties of the workload but increases the modeling complexity.

We extend TimeLoop's cost model with latency/energy terms for inter-GPU links and global synchronization, using the bandwidth and energy data for NVLINK-C2C and NDR 400.

- **Impact of Number of GPUs on Compute and Communication Latency:**
  In the first experiment, we explored how increasing the number of GPUs influences compute and communication latency under three different parallelism strategies: *data parallelism*, *tensor parallelism*, and *pipeline plus tensor parallelism (PP+TP)*.
  We included PP+TP because pipeline parallelism alone is fundamentally limited by the number of einsums in the workload. For example, the FFN block contains only two einsums, so pipeline parallelism alone cannot scale beyond two GPUs. By combining pipeline and tensor parallelism, we enable multi-GPU execution within each pipeline stage, allowing us to scale the workload to more GPUs and compare results with the other two types of parallelism.
  For data and tensor parallelism, we varied the GPU count across $\{4, 6, 8, 10, 16, 32\}$. For PP+TP on the attention block, we varied the number of GPUs used per einsum across $\{2, 4, 8\}$, resulting in total GPU counts of 12, 24, and 48 respectively, since the attention block consists of six einsums.
- **Impact of Block Size on Compute and Communication Latency:**

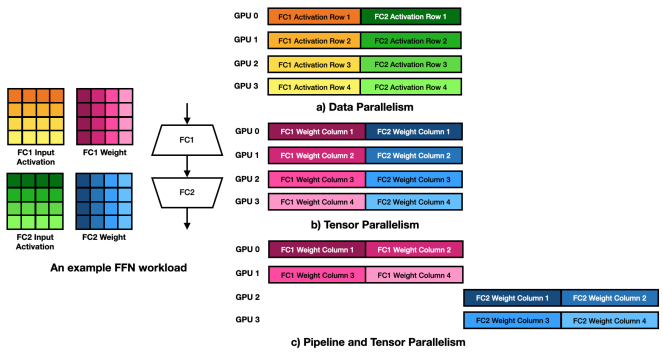

Fig. 1. Visualization of different parallelism modes on an example workload

In the second experiment, we examined how varying the *block size*, which changes the total number of threads or compute units per GPU, affects latency. The goal was to explore the trade-off between *intra-GPU* and *inter-GPU* communication, and to understand whether it is more efficient to run a workload on a single large GPU or distribute it across multiple smaller GPUs.

This experiment was conducted across all three parallelism strategies: data parallelism, tensor parallelism, and pipeline plus tensor parallelism. For both FFN and attention workloads, we fixed the number of GPUs to 4 and varied the block size across $\{64, 128, 256, 512, 1024\}$. This allowed us to observe how local compute granularity impacts overall latency and communication behavior under different parallelism modes.

A. Workload Mapping and Mapspace Exploration

Figure 1 illustrates how we implemented different types of parallelism for a feed-forward network (FFN) workload. In data parallelism (Figure 1a), each GPU holds a full copy of the model weights and processes a separate row of the input activation, allowing all GPUs to compute independently. In tensor parallelism (Figure 1b), the weight matrices are column-wise sharded across GPUs, requiring each GPU to read the full input activation to compute partial outputs. Finally, in pipeline plus tensor parallelism (Figure 1c), we divide the two FC layers into separate pipeline stages, and apply tensor parallelism within each stage.

B. Communication Modeling

The parallelisms that we modeled involves the following communications:

- **Data Parallelism (DP):** Modeled via the backward pass `All-Reduce` of full-precision gradients.
- **Tensor Parallelism (TP):** Modeled via the forward pass `All-gather` of input activations and `Reduce-scatter` of output activations across transformer blocks.
- **Pipeline Parallelism (PP):** Modeled via the forward pass point-to-point transfer of activation checkpoints between stages.

We model the Reduce-scatter and the All-gather as using a Ring Algorithm, so with the All-Reduce. This maximally utilizes the available bandwidth, but is bottlenecked by the slowest link. Extensions to tree algorithms are possible.

Our communication model is simple and does not estimate the Communication/computation overlap or pipeline bubbles.

The communication model distinguishes between intra-node (`NVLink-C2C`) and inter-node (`InfiniBand NDR 400`) hops. The latency for a message of size M bytes traversing n_{NV} `NVLink` hops and n_{IB} `InfiniBand` hops is calculated using the following model:

$$\text{Latency} = n_{\text{NV}}\left(\beta_{NV} M\right) \; + \; n_{\text{IB}}\left(\beta_{IB} M\right)$$

The associated energy cost is:

$$\text{Energy} = 8M\left(\text{cost}_{\text{NV}} \cdot n_{\text{NV}} + \text{cost}_{\text{IB}} \cdot n_{\text{IB}}\right)$$

Hardware link parameters used are:

- **`NVLink-C2C`**: Peak BW = 900 GB/s, $\beta_{NV} = 1.11$ ps/B, Energy $\text{cost}_{\text{NV}} = 1.3$ pJ/bit [41].
- **`NDR 400`**: Peak BW = 50 GB/s, $\beta_{IB} = 20$ ps/B, Energy $\text{cost}_{\text{IB}} \approx 20$ pJ/bit [42].

(Note: BW values are peak; startup latencies are assumed negligible, and communication kernel launch overheads are not modeled).

IV. Results and Discussion

A. Impact of Changing GPU Number

As the number of GPUs increase, the amount of compute that can be done in parallel increases, but so does the communication overhead, especially if there is communication between two different racks because inter-rack communication uses NDR 400, which is slower and more costly per bit.

As shown in Figures 2, 3, 4, and 5, we made some observations on how compute latency changes with respect to number of GPUs when we use different parallelism modes:

- **Data vs. Tensor Parallelism:** We observed that both data and tensor parallelism exhibit similar latency trends as the number of GPUs increases as shown in Figure 2. The latencies are also roughly equivalent, which can be attributed to the fact that in both cases, the workload is compute-bound. Since there are no inter-GPU dependencies that stall execution, all compute units are fully utilized.

  However, tensor parallelism incurs noticeably higher energy consumption for GPU memory reads as shown in Figure 6. This is likely due to the fact that while weights are sharded across GPUs in tensor parallelism, each GPU still needs access to the entire input activation. Consequently, each row of the input activation experiences reduced reuse and must be loaded multiple times across different GPUs.

  In contrast, data parallelism replicates the entire model on each GPU, leading to repeated loading of large weight matrices. Therefore, tensor parallelism can be more memory-efficient for workloads with large weights,

as it avoids loading the same weights redundantly. The trade-off, however, is higher GPU memory read energy and reduced data locality due to repeated activation loads. Another observation we made is that for the attention workload, data parallelism fails to reduce compute latency beyond 16 GPUs (Figure 4), as the batch size is limited to 16 and thus provides no additional parallelism beyond that point. In contrast, tensor parallelism continues to yield latency improvements beyond 16 GPUs, since it parallelizes along the width of the weight matrix—a dimension that is significantly larger than 16. This highlights the importance of considering the size of the parallelizable dimension when selecting a parallelism strategy. Depending on the structure of the workload, certain dimensions may offer greater scaling potential than others.

- **Pipeline + Tensor Parallelism on FFN:** When combining pipeline and tensor parallelism for FFN layers, we observed a significant increase in latency compared to data and tensor parallelism alone (Figure 3). Furthermore, latency plateaued after 8 GPUs, indicating vanishing returns from adding more compute resources. This is likely because the FFN workload becomes memory-bound under this configuration.
  This hypothesis is supported by the marked increase in both Main Memory Read Energy and GPU Memory Write Energy (Figure 6). The higher write energy may result from GPUs needing to repeatedly write intermediate outputs (e.g., from the first FFN layer) to memory so that another set of GPUs can read them as inputs to the second layer.
  The elevated main memory read energy may be due to how pipeline parallelism assigns layers to available GPUs dynamically. As a result, a GPU may not always process the same layer and same weights, leading to repeated weight loads from main memory due to reduced reuse.
- **Pipeline + Tensor Parallelism on Attention Block:** In contrast, the attention block showed only a modest latency increase when comparing tensor parallelism to the combined pipeline + tensor approach as shown in Figure 5. Latency continued to decrease with more GPUs, indicating the workload remained compute-bound. This is likely due to the smaller size of both weights and activations in attention layers.
  Still, we observed a latency increase when pipeline parallelism was introduced. This can be attributed to two key factors:
  1) *Pipeline bubbles* arise due to initial idle time—only certain layers (e.g., `KeyProjection`, `QueryProjection`, `ValueProjection`) can begin execution immediately, while dependent layers (e.g., `QueryKeyMultiplication`, `AttentionValue`, `Output`) must wait for the outputs of earlier layers.
     Additionally, under tensor parallelism, each GPU in the first layer produces only a partial output—specifically, a subset of the columns required by the next layer. However, for the second set of GPUs to begin computation, they require full input rows, not just partial columns. This means the second stage must wait until all GPUs in the first layer complete their respective computations, leading to significant idle time and pipeline bubbles that increase total compute latency.
     One potential solution to mitigate this issue is to tile the weights of the second layer along the row dimension instead of the column dimension. This would allow the second-stage GPUs to begin computation as soon as a partial row of input becomes available, improving pipeline efficiency and reducing idle cycles.
     However, this approach comes with its own trade-offs. Tiling along the row dimension reduces spatial locality in writing to the same output positions, potentially increasing memory traffic and write energy. In particular, GPUs may perform more scattered memory accesses, which can degrade overall memory efficiency and impact performance depending on the hardware memory hierarchy.
  2) *Asymmetric dependencies* in the compute graph—the Output layer depends on two preceding layers with different depths in the compute graph. This imbalance introduces additional pipeline bubbles and increases overall latency.

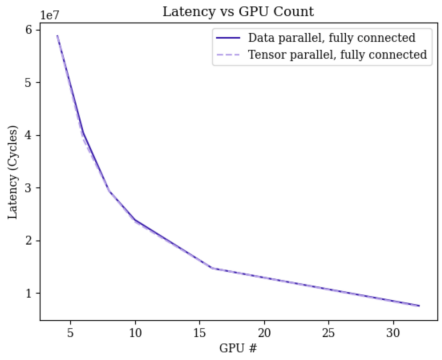

Fig. 2. GPU sweep for tensor parallelism and data parallelism on fully connected network.

B. Impact of Changing Blocksize

Another way to increase parallelism is to use more powerful GPUs. To study this, we fixed a low GPU count (4 GPUs) and swept block sizes of 64, 128, 256, 512, and 1024.

As seen in Figure 7, When sweeping GPU blocksize on the transformer workload, data parallelism is better than the other 2 types of parallelism for small block sizes, but plateaus out much sooner on the transformer workload. This is likely because the bottleneck is no longer computation but communications. In data parallelism, there is a lot of data movement within a single GPU, which means that increasing compute parallelism doesn't help if the amount of computation

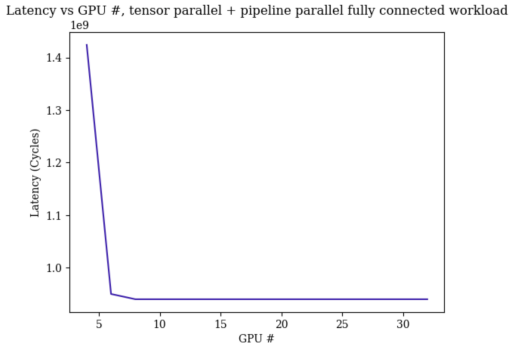

Fig. 3. GPU sweep for tensor plus pipeline parallelism on fully connected network.

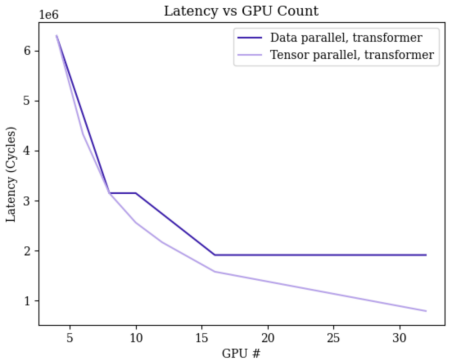

Fig. 4. GPU number sweep for tensor and data parallelism on transformer workload.

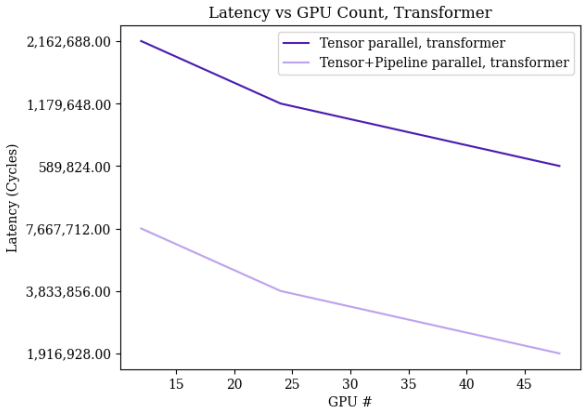

Fig. 5. GPU number sweep for tensor plus pipeline parallelism on transformer workload. Because of the pipeline parallelism, only multiples of 6 could be used.

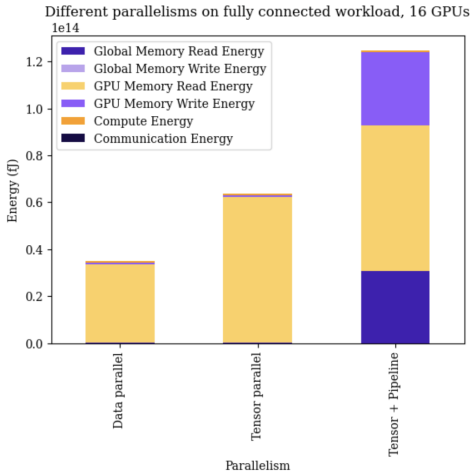

Fig. 6. A comparison of energy breakdown between different styles of parallelisms on 16 GPUs, fully connected.

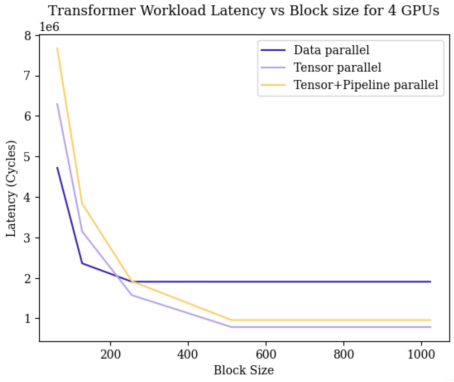

Fig. 7. Sweep of blocksizes on transformer workload for 4 GPUs.

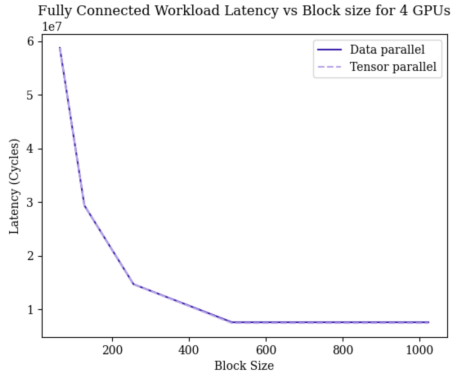

Fig. 8. Sweep of blocksizes on fully connected workload for 4 GPUs. Tensor/pipeline parallelism omitted due to being orders of magnitude larger.

any single GPU has to do is already small enough. On the other hand, both tensor and tensor plus pipeline parallelism have a lot of inter-GPU movement, which means that using fewer GPUs with more blocks makes it possible to have less communication between GPUs, leading to a greater benefit.

In the fully connected workload (Figure 8), data parallelism and tensor parallelism have nearly identical latencies (while tensor-pipeline parallelism had much worse latencies, two orders of magnitude greater.) In the fully-connected workload, both parallelisms are compute-bound. On the other hand, because the fully connected workload has much fewer layers, it can only utilize only two GPUs for pipeline parallelism, whilst still incurring the pipeline bubbles and tensor parallel overhead, making it have a much higher latency.

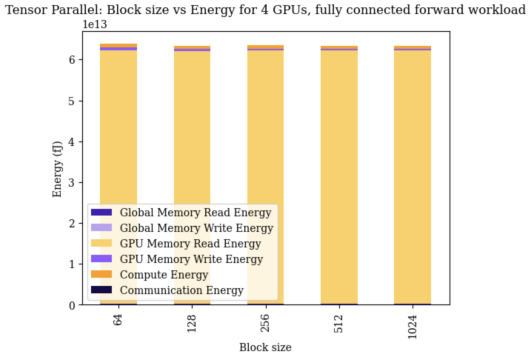

Fig. 9. Sweep on blocksize using tensor parallel on fully connected workload

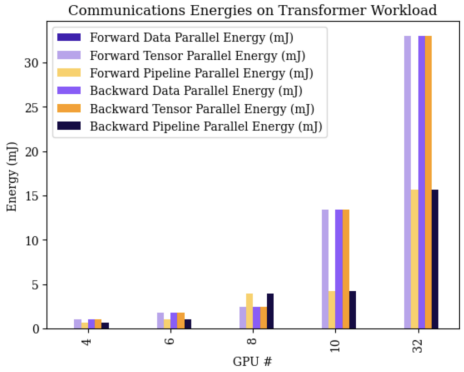

Fig. 10. Inter-GPU communications energies for different GPU counts. Note that forward data parallelism has no communication cost.

This result shows that increasing parallelism within the GPU can dramatically improve the latency performance of tensor and tensor/pipeline parallelism, while for data parallelism, the benefits only come from a much larger workload.

When sweeping across block sizes for the fully connected workload for tensor parallelism, there is not a significant energy difference (Figure 9). This suggests that the same reads and writes are being made from the memory, but the computation is being spread out between blocks.

C. Summary of Comparison between Parallelism Methods

- **Data Parallelism** Comparatively, data parallelism had the least energy usage. This is because each GPU processes an independent subset of the input data using a full copy of the model, which allows for excellent reuse of activations and minimal communication during the forward pass. The trade-off is that each GPU must store a full replica of the model, which can become inefficient for large models due to memory and redundancy overhead. Data parallelism also is limited when it comes to latency reductions as the data can only be split as much as the batch size. Additionally, the forward pass for data parallelism has no communication costs, although the backward pass has a cost for the reduce (Figures 10, 11).
- **Tensor Parallelism** Tensor parallelism introduces higher GPU memory read energy due to the need for each GPU to access the entire input activation, despite only owning a shard of the weights. This leads to reduced activation reuse and increased memory traffic. Nonetheless, tensor parallelism is not constrained by batch size, allowing it to continue achieving latency reductions beyond the point where data parallelism plateaus. It is particularly advantageous in models with large weight matrices, where sharding weights across GPUs reduces memory load redundancy. Although tensor parallelism incurs communication costs in both the forward and backward passes, these costs were not significant enough to dominate in our modeling. Overall, it provides a strong trade-off between memory efficiency and scalability when model dimensions (e.g., width) support deep parallelization (Figures 10, 11).

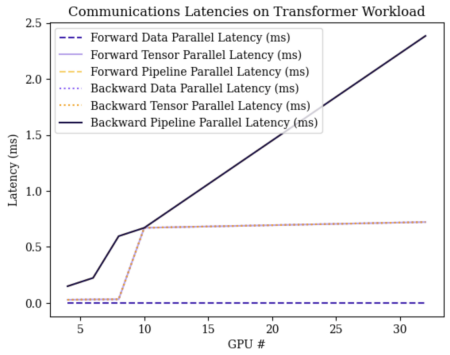

Fig. 11. Inter-GPU communications latencies for different GPU counts. Note that forward data parallelism has no communication cost.

- **Tensor and Pipeline Parallelism** When pipeline parallelism is combined with tensor parallelism, energy usage patterns shift considerably. We observed much higher global read and GPU write energy, driven by the need to repeatedly read activations and write intermediate outputs between stages. Our hypothesis is that this arises because each GPU processes different layers in a pipeline and must fetch weights and activations afresh due to reduced reuse. For FFN layers, this setup becomes memory-bound, leading to high latency and limited performance gains beyond 8 GPUs. In the attention block, while latency still decreases with GPU count, pipeline bubbles and asymmetric dependencies between layers introduce stalls that diminish gains. Notably, despite the higher latency at 32 GPUs, this configuration exhibited the lowest communication energy. (Figures 10, 11) This is because pipeline parallelism does not use all-reduce operations like data or tensor parallelism; instead, it sends less data point-to-point along a critical path. As such, while the compute latency and memory pressure can be high, pipeline parallelism can reduce communication volume when ring-based reductions are a bottleneck.

V. Limitations and Future Work

There are several limitations of our current analytical framework and modeling that offer avenues for future research:

- **Communication Model Simplification:** The current communication model uses peak bandwidth figures and neglects crucial overheads like collective operation startup latencies and kernel launch times. Further, although RDMA bypasses host memory but still incurs network stack and NIC processing delays not fully captured. This simplification likely contributes to the observation that communication overhead appeared negligible in some results. We also primarily modeled ring-based collectives, while tree-based or hardware-accelerated collectives (e.g., SHARP [27]) could yield different results.
- **Lack of Overlap Modeling** The model does not explicitly account for computation/communication overlap or the impact of pipeline bubbles beyond basic scheduling dependencies. Techniques like 1F1B scheduling in PipeDream [21], Gradient Bucketing in allreduce [43], or ZeRO's overlap strategies [19] significantly affect real-world performance but are not modeled here.
- **Simplified Workloads and Parallelisms:** We focused on core FFN and Attention blocks, omitting components like normalization layers, activations (SwiGLU, Softmax communication), and the complexities of full model graphs. Furthermore, we modeled a subset of parallelism strategies, excluding important techniques like expert parallelism [22], [44], sequence parallelism [45], or more complex hybrid strategies like Fully Sharded Data Parallelism (FSDP) [19].
- **Static Analysis** The analysis is static, evaluating a single forward/backward pass instance. It does not capture dynamic training effects like checkpointing overhead [46], [47] or fault tolerance mechanisms [48]
- **Lack of Empirical Validation:** The results are based on analytical modeling using Timeloop/Accelergy. Validation against real-world cluster performance and power measurements is essential to confirm the model's accuracy and the practical relevance of the findings.

Addressing these areas will enhance the fidelity and scope of our analytical framework, providing more accurate tools for optimizing the design and deployment of energy-efficient, high-performance rack-scale systems for large AI model training.

REFERENCES

[1] I. King and D. Bass, "Openai expects revenue will triple to $12.7 billion this year," https://www.bloomberg.com/news/articles/2025-03-26/openai-expects-revenue-will-triple-to-12-7-billion-this-year, March 2025, accessed: 2025-04-28.

[2] Meta AI, "The llama 4 herd: The beginning of a new era of natively multimodal ai innovation," Meta AI Blog. [Online]. Available: https://ai.meta.com/blog/llama-4-multimodal-intelligence/

[3] A. Dandgaval, "Mastering memory optimization in ml systems: Techniques for large models," https://medium.com/@aseemdandgaval1/mastering-memory-optimization-in-ml-systems-a-deep-dive-into-techniques-for-large-models-2f9438ec02d7, 2024, accessed 18 Apr 2025.

[4] NVIDIA Corporation, "Nvidia a100 tensor core gpu architecture," White Paper, Tech. Rep., 2020, rev. 2.1, WP-09183. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

[5] NVIDIA, "NVIDIA Blackwell Architecture," https://resources.nvidia.com/en-us-blackwell-architecture, March 2024, accessed: 2025-04-28.

[6] OpenAI, "Gpt-4 technical report," *CoRR*, vol. abs/2303.08774, 2023. [Online]. Available: https://arxiv.org/abs/2303.08774

[7] CoreWeave, "Coreweave leads the charge in ai infrastructure efficiency, with up to 20% higher gpu cluster performance than alternative solutions," 2025, accessed: 2025-05-05. [Online]. Available: https://www.coreweave.com/blog/coreweave-leads-the-charge-in-ai-infrastructure-efficiency-with-up-to-20-higher-gpu-cluster-performance-than-alternative-solutions

[8] SemiAnalysis, "100,000 h100 clusters: Power, network topology, ethernet vs infiniband, reliability, failures, checkpointing," 2024, accessed: 2025-05-05. [Online]. Available: https://semianalysis.com/2024/06/17/100000-h100-clusters-power-network/

[9] S. N. Gowda, X. Hao, G. Li, S. N. Gowda, X. Jin, and L. Sevilla-Lara, "Watt for what: Rethinking deep learning's energy-performance relationship," 2024. [Online]. Available: https://arxiv.org/abs/2310.06522

[10] S. A. Khowaja, P. Khuwaja, K. Dev, W. Wang, and L. Nkenyereye, "Chatgpt needs spade (sustainability, privacy, digital divide, and ethics) evaluation: A review," *Cognitive Computation*, vol. 16, no. 5, p. 2528–2550, May 2024. [Online]. Available: http://dx.doi.org/10.1007/s12559-024-10285-1

[11] "xAI & MLGW Quick Facts," Memphis Light, Gas and Water, 2024, accessed on 2025-04-28. [Online]. Available: https://www.mlgw.com/images/content/files/pdf/PDF2024/2024xAI%20and%20MLGW

[12] J. Fernandez, L. Wehrstedt, L. Shamis, M. Elhoushi, K. Saladi, Y. Bisk, E. Strubell, and J. Kahn, "Hardware scaling trends and diminishing returns in large-scale distributed training," 2025. [Online]. Available: https://arxiv.org/abs/2411.13055

[13] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1807.05358

[14] M. Tanaka, D. Li, U. Chand, A. Zafar, H. Shen, and O. Ruwase, "Deepcompile: A compiler-driven approach to optimizing distributed deep learning training," 2025. [Online]. Available: https://arxiv.org/abs/2504.09983

[15] L.-W. Chang, W. Bao, Q. Hou, C. Jiang, N. Zheng, Y. Zhong, X. Zhang, Z. Song, C. Yao, Z. Jiang, H. Lin, X. Jin, and X. Liu, "Flux: Fast software-based communication overlap on gpus through kernel fusion," 2024. [Online]. Available: https://arxiv.org/abs/2406.06858

[16] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, 2019, pp. 304–315.

[17] M. Gilbert, Y. N. Wu, J. S. Emer, and V. Sze, "Looptree: Exploring the fused-layer dataflow accelerator design space," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, pp. 1–15, 2024.

[18] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[19] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Yuxiong He, "ZeRO–Infinity: Breaking the gpu memory wall for extreme scale deep learning," *arXiv preprint arXiv:2104.07857*, 2021.

[20] Y. Huang, Y. Cheng, A. Bapna, Orhan Fırat, Mia Xu Chen *et al.*, "GPipe: Efficient training of giant neural networks using pipeline parallelism," *arXiv preprint arXiv:1811.06965*, 2019.

[21] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia, "Memory-efficient pipeline-parallel DNN training (pipedream-2bw)," *International Conference on Machine Learning (ICML)*, 2021.

[22] W. Fedus, Barret Zoph, and Noam Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *arXiv preprint arXiv:2101.03961*, 2021.

[23] NVIDIA Corporation, "NVLink and NVSwitch: Fastest hpc data center platform," https://www.nvidia.com/en-us/data-center/nvlink/, 2025, accessed 28 Apr 2025.

[24] ——, "NVSwitch Technical Overview," NVIDIA, Tech. Rep., 2017. [Online]. Available: https://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf

[25] AMD, Inc., "Amd looks to infinity for ai interconnects," https://www.wheelersnetwork.com/2024/01/amd-looks-to-infinity-for-ai.html, 2024, accessed 28 Apr 2025.
[26] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang, S. Zhang, M. J. Fernandez, S. Gandham, and H. Zeng, "Rdma over ethernet for distributed ai training at meta scale," in *Proc. ACM SIGCOMM 2024*, 2024.
[27] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer *et al.*, "SHArP: A scalable hierarchical aggregation and reduction protocol for infiniband networks," in *Proc. IEEE Hot Interconnects 25*, 2017.
[28] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," in *Proc. 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021.
[29] S. Liu, Q. Wang, J. Zhang, Q. Lin, Y. Liu, M. Xu, R. C. C. Chueng, and J. He, "Netreduce: Rdma-compatible in-network reduction for distributed dnn training acceleration," in *Proc. 49th International Conference on Parallel Processing*, 2020.
[30] P. Patrizio, M. Zhao, and C. Alatarvas, "Reconciling the contrasting narratives on the environmental impact of ai," *Scientific Reports*, vol. 14, no. 1234, pp. 1–12, 2024.
[31] "Edge TPU performance benchmarks," https://coral.ai/docs/edgetpu/benchmarks/, 2020, accessed May 4, 2025.
[32] Gartner, Inc., "Gartner Predicts Power Shortages Will Restrict 40
[33] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.
[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *International Conference on Learning Representations (ICLR)*, 2016.
[35] E. Bortac, A. Alsaffar, and S. Memon, "Impact of mixed precision techniques on training and inference efficiency of deep neural networks," *arXiv preprint arXiv:2405.01234*, 2024.
[36] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, "Sparseloop: An Analytical Approach To Sparse Tensor Accelerator Modeling ," in *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2022.
[37] T. Andrulis, J. S. Emer, and V. Sze, "CiMLoop: A flexible, accurate, and fast compute-in-memory modeling tool," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2024.
[38] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.
[39] NVIDIA Corporation, *GPUDirect RDMA*, https://docs.nvidia.com/cuda/pdf/GPUDirect$_R DMA.pdf$, $2025, accessed: 2025-05-05$.
[40] A. Grattafiori *et al.*, "The llama 3 herd of models," *CoRR*, vol. abs/2407.21783, 2024. [Online]. Available: https://arxiv.org/abs/2407.21783
[41] NVIDIA Corporation, "NVIDIA NVLink-C2C," https://www.nvidia.com/en-us/data-center/nvlink-c2c/, 2025, accessed: 2025-05-04.
[42] ——, "NVIDIA ConnectX-7 NDR 400G InfiniBand Adapter Card Datasheet," NVIDIA Corporation, Datasheet, Apr. 2021. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/networking/infiniband-adapters/infiniband-connectx7-data-sheet.pdf
[43] PyTorch Development Team, "Distributed data parallel," https://pytorch.org/docs/stable/notes/ddp.html (accessed May 5, 2025), 2024, pyTorch 2.7 documentation.
[44] DeepSeek-AI, "Deepep: An efficient expert-parallel communication library," https://github.com/deepseek-ai/DeepEP, 2025.
[45] S. Li, F. Xue, C. Baranwal, Y. Li, and Y. You, "Sequence parallelism: Long sequence training from system perspective," *arXiv preprint arXiv:2105.13120*, 2021.
[46] G. Wang, O. Ruwase, B. Xie, and Y. He, "FastPersist: Accelerating model checkpointing in deep learning," *arXiv preprint arXiv:2406.13768*, 2024.
[47] Z. Wang, Z. Jia, S. Zheng, Z. Zhang, X. Fu, T. Ng, and Y. Wang, "Gemini: Fast failure recovery in distributed training with in-memory checkpoints," in *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23)*. Koblenz, Germany: ACM, 2023, pp. 1–18.
[48] P. Contributors, "`torchft`: Easy per-step fault tolerance for pytorch," https://github.com/pytorch/torchft, 2025.

VI. Appendix

A. *Unused figures*

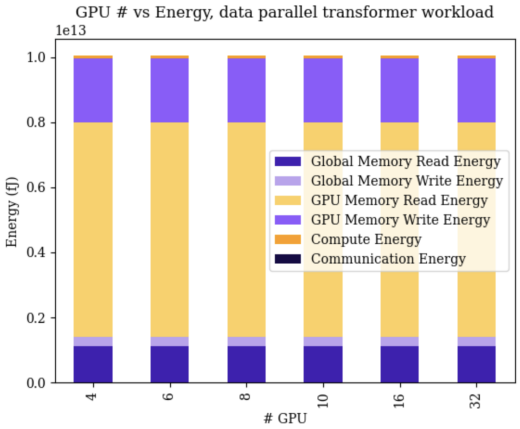

Fig. 12. Energy of different GPU counts using data parallelism on a transformer workload.

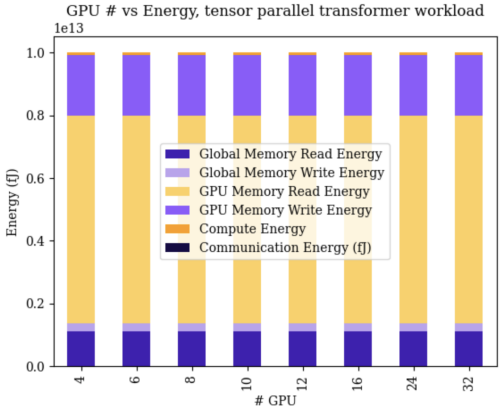

Fig. 13. Energy of different GPU counts using tensor parallelism on a transformer workload.