# Compilation-Guided Energy Optimization in Large Model Training

Ruofan Wu
ruofanw@umich.edu

Zheng Li
jimmyli@umich.edu

Jeff J. Ma
jeffjma@umich.edu

Dhanvi Bharadwaj
dhanvib@umich.edu

## 1 Introduction

Large-scale deep learning models, particularly Large Language Models (LLMs), have revolutionized artificial intelligence, driving breakthroughs across natural language processing, computer vision, and generative tasks. However, the substantial computational requirements for training these models translate into enormous energy consumption. For example, training state-of-the-art models like Llama 3.2 and DeepSeek V3 often demands thousands of GPU-hours, resulting in significant environmental and economic impacts [1, 15, 18, 21]. Hence, optimizing the energy efficiency of large model training has become a critical research priority.

Recent advances have demonstrated substantial energy savings through software-driven scheduling of hardware resources compared to relying solely on hardware improvements [4, 10, 11, 25]. The effectiveness arises from the highly predictable nature of LLM training workloads, allowing software to dynamically adjust hardware settings based on workload characteristics. For instance, Perseus [11] reduces GPU frequency for training pipeline stages whose latency does not affect overall system performance, thus achieving meaningful reductions in energy consumption without compromising throughput.

However, the scope of current methodologies remains limited in several aspects. First, while Perseus identifies energy inefficiencies resulting from latency imbalances, it treats each pipeline stage and GPU hardware as *black boxes*. It neglects the opportunities at finer compilation granularity when analyzing the workload characteristics and GPU hardware specifications, such as intra-GPU resource usage imbalances. Second, existing techniques assume static compilation methods (e.g., kernel fusion and code generation) across all GPU frequencies, causing resource imbalance when hardware specifications change.

In this project, we seek to open this black box to further push the envelope of optimization. First, we show that energy consumption can be optimized through compilation techniques by eliminating imbalances in resource usage: specifically across computation, memory, and network. At the graph level, transformer computations alternating with GPU communications cause periodic resource idleness, which leads to unnecessary power consumption. By strategically overlapping computation and communication through a precise scheduling of GPU resources—such as the number of Streaming Multiprocessors (SMs) and threads—we achieve balanced resource utilization and energy savings. At the operator level, selecting optimal tile sizes for CUDA kernels balances the compute-to-memory ratio, further enhancing energy efficiency.

Second, we advocate dynamically co-optimizing compilation with GPU frequency adjustments for optimal energy efficiency. Although reducing GPU frequency can lower energy consumption, it alters the GPU's computation throughput (FLOPS) without changing its memory and network throughput. This breaks the balance of resource utilization because the balance was established assuming the GPU's default frequency. Therefore, dynamic adjustment of both compilation schedules and GPU frequency is essential to maintaining optimal resource balance and energy savings.

Based on these insights, we propose a co-optimization framework that integrates graph-level and operator-level compilation strategies with dynamic frequency tuning, as illustrated in Figure 1. This co-optimization approach significantly advances the energy-time Pareto frontier. In this report, we separately perform microbenchmarks for graph-level and operator-level compilation optimizer to validate the effectiveness of our approach. Section 3 illustrates our reduction in energy consumption of the training pipeline stage by 23%.

In Section 2, we briefly introduce relevant background information, including large model training, deep learning compilation techniques, and existing frameworks for energy-efficient training. In Section 3, we explore graph-level compilation opportunities for energy optimization. We demonstrate that latency and energy consumption can be optimized by dynamically allocating resources to overlap communication and computation kernels. Furthermore, we highlight the necessity of adopting distinct configurations for different GPU frequencies to recalibrate the balance between computation and network resources. In Section 4, we investigate energy optimization opportunities at the operator level. First, we analyze the impact of different code generation (codegen) schedules on time and energy, noting that some compilation-generated operations provide negligible performance benefit but increase GPU power consumption. Subsequently, we conclude, similar to the graph-level findings, that varying GPU frequencies require different codegen strategies to optimally rebalance computation and memory resources.

## 2 Background

### 2.1 Large Model Training

Large model training typically involves three parallelism paradigms: data parallelism (DP), tensor parallelism (TP), and pipeline parallelism (PP). Pipeline parallelism partitions

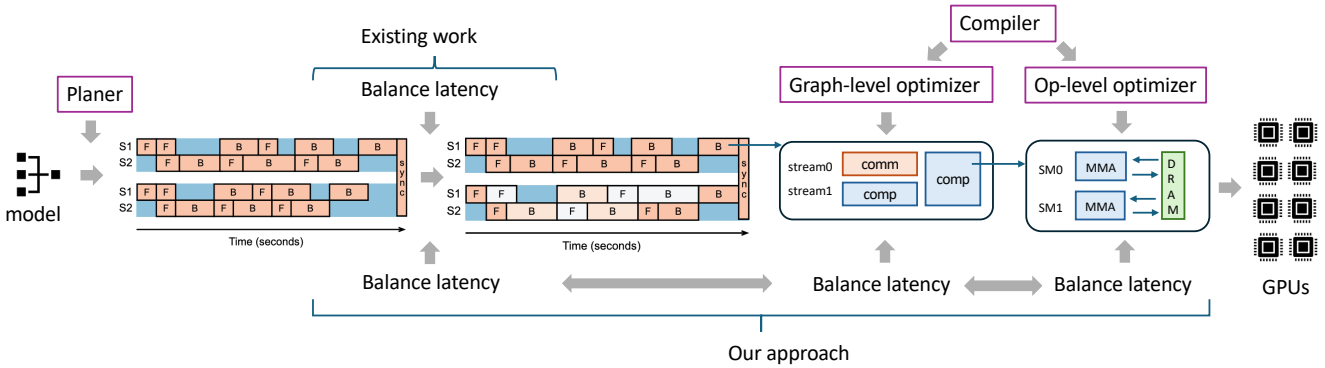



Figure 1: The motivation and overview of our project.

model layers into groups executed across different GPU groups. Tensor parallelism partitions individual layers across GPUs, requiring communication (via AllReduce) to aggregate intermediate results after computation of attention and MLP layers in Transformer models. Data parallelism partitions input data and replicates the entire pipeline.

## 2.2 Deep Learning Compilation

Given a model definition provided by a training framework, compilation translates the model into executable code. Whether utilizing a deep learning compiler like PyTorch [5, 19] or manually optimized kernel libraries such as Transformer Engine [2], compilation decision-making involves two levels of optimization. Graph-level optimization determines the execution order and combination of operators, such as fusing attention operations into a single kernel like FlashAttention [12, 13] and overlapping communication with computation operations [16, 24]. Operator-level optimization specifies the code generation schedule for fused graphs or individual operators, such as selecting data processing tile sizes for matrix multiplication kernels and FlashAttention kernels. Although deep learning compilation has been extensively studied for performance optimization, research on improving its energy efficiency [17], particularly in conjunction with GPU frequency tuning, remains limited.

## 2.3 Energy-Efficient Training

Researchers have increasingly focused on developing energy-efficient training techniques. Perseus [11], a state-of-the-art energy optimization framework for large model training, identifies intrinsic and extrinsic energy inefficiencies arising from latency imbalances among pipeline stages and data copies of pipelines. Perseus dynamically reduces GPU frequency to balance the latency among pipeline stages, achieving energy savings without compromising overall training performance, as shown on the left in Figure 1. It simply adjusts GPU frequencies to explore the energy-time trade-off frontier, ignoring the optimization opportunities to uncover detailed hardware utilization patterns.

# 3 Graph-level Optimizer

## 3.1 Balance of Computation and Network

GPU wastes idle power when computations are not occurring, presenting a significant opportunity for optimizing GPU energy consumption. Besides pipeline stage latency imbalances [11], a closer examination of each pipeline stage reveals additional idle power waste due to GPU communication with other GPUs. Tensor parallelism introduces substantial communication overhead, and with the dramatically increased computational throughput of the latest GPU generations, communication can occupy up to 40% of GPU time [23]. While GPUs wait for network bandwidth, SMs become idle without computational tasks. Therefore, balancing computation and network latencies is crucial for optimizing both latency and energy.

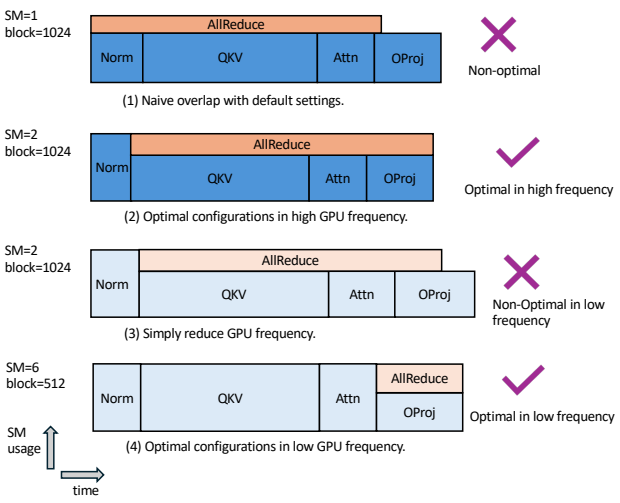



Figure 2: The insights of graph-level optimizer.

**Limitations of existing work.** Several studies have optimized large-model training latency by overlapping computation with communication [6, 16, 20, 22, 24]. However, these approaches primarily focus on hiding communication latency through simplistic scheduling, neglecting the balance between computation and network latencies. First, they overlook the influence of different overlap scopes on resource balance. Ex-

isting methods immediately launch communication kernels once dependencies are ready (Figure 2 (1)), causing idle SMs if communication completes prematurely. Additionally, some works fuse only one communication kernel with a single computation kernel [16, 20, 24], missing opportunities for broader overlaps and causing imbalance when communication kernels are lengthy.

Second, prior methods disregard how varying resource allocation affects latency balance, typically using fixed configurations for SMs and threads in communication kernels. Such inflexibility leads to suboptimal resource usage. As illustrated in Figure 2 (2), strategically adjusting communication kernel scheduling and resource allocation can precisely balance computation and communication and reduce GPU idle power.

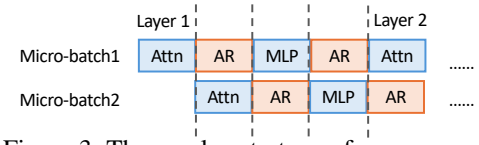



Figure 3: The overlap strategy of our approach.

**Our approach.** Inspired by Domino [22], our work divides inputs into two micro-batches without data dependency for each pipeline stage employing tensor parallelism. Then, we overlap the communication kernel (Allreduce) of one micro-batch with the attention and MLP computation kernels of another micro-batch, as shown in Figure 3. Each pipeline stage comprises numerous consecutive Transformer layers, enabling all communication kernels to overlap effectively with computation. This strategy also provides flexible control over overlap scopes within attention and MLP layers.

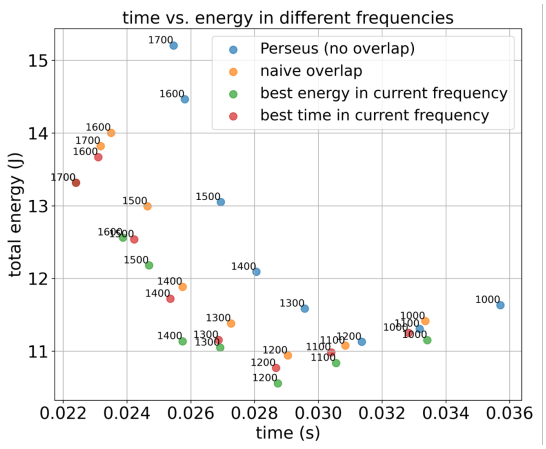
We adopt NanoFlow's [28] communication kernel design and develop customized GPU communication kernels using MSCCL++ [9], allowing flexible control over the number of SMs and threads. By strategically adjusting the overlap scope and SM resource allocation, our approach achieves balanced computation and network latency, significantly reducing energy consumption. Experimental results at the graph-level are presented in Section 3.2.

**GPU frequency reshapes the balance.** Reducing GPU SM frequency is an effective method to decrease energy consumption. However, simply lowering GPU frequency disrupts the carefully established balance. This imbalance arises because reducing GPU frequency affects computational throughput but does not alter network and memory throughput. As illustrated in Figure 2 (3), computation kernel execution times increase significantly, whereas allreduce kernel durations remain nearly constant, causing SMs dedicated to communication to become idle prematurely. Intuitively, using a smaller overlap scope and adjusting SM resources can rebalance computation and communication, as demonstrated in Figure 2 (4). Thus, we propose *dynamically adjusting communication overlap scopes and SM resources under varying GPU frequencies*, further enhancing energy efficiency.

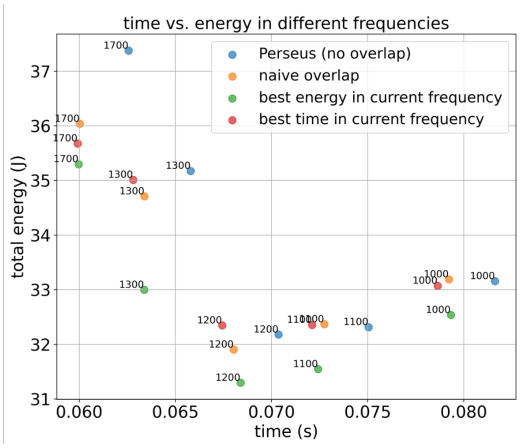
## 3.2 Experimental Results

In this subsection, we demonstrate the advantages of our approach through two microbenchmarks, each overlapping the Allreduce kernel with attention and MLP layers, respectively.

**Experimental setup.** We conduct experiments on the forward pass of Llama3.1-8b model [14]. We fix the tensor parallel degree to 2, batch size to 4 and sequence length to 4096 and evaluate all combinations of overlap scope, SMs number and threads number of Allreduce kernel. The experimental platform includes two NVIDIA A40 GPU and an AMD EPYC 7513 32-Core processors, using PyTorch 2.5.0 and CUDA 12.6. We use ZeusMonitor [25] to measure GPU energy consumption and execution time.




(a) Attention




(b) MLP

Figure 4: Energy consumption and time under different GPU frequencies.

Figure 4 shows the energy consumption and execution time resulting from overlapping communication and computation (attention and MLP layers) under various overlap configurations and GPU frequencies. The numbers near each data point indicate GPU frequencies. Blue points represent our baseline, Perseus, which does not use overlap techniques. Orange points correspond to naive overlap configurations using default kernel settings (Figure 2(1)). Green and red points

denote optimal energy and time configurations, respectively, identified within the explored parameter space for each frequency. Since results for the MLP layer are similar from 1700 MHz to 1400 MHz, only results at 1700 MHz are shown.

We draw the following conclusions. First, dynamically adjusting compilation parameters significantly reduces energy consumption within the same time constraint compared to the baseline. For instance, in the attention layer, given a time limit of 0.026 s, Perseus sets the GPU frequency to 1600 MHz, resulting in an energy consumption of 14.46 J. In contrast, our optimized configuration at 1400 MHz achieves 11.14 J, representing a 23% energy saving. Second, optimal configurations for energy differ from those for execution time, indicating an inherent tradeoff between energy and performance. This enriches our resulting energy-time Pareto frontier.

# 4 Operator-level Optimizer

## 4.1 Balance of Computation and Memory

The operator-level compiler, or tensor compiler, generates the codegen schedules for operators [7, 8, 26, 27]. Tile size influences the computational load and memory access within the SM, as well as computational throughput. For example, in matrix multiplication (Matmul) kernels, increasing the tile size improves data reuse, thus reducing memory access. However, larger tile sizes require more shared memory to store tiles, decreasing the number of thread-blocks that an SM can simultaneously launch and consequently lowering GPU utilization and computational throughput. Since the computing units within GPU SMs dominate power consumption, GPU power consumption is always positively correlated with computational throughput. Therefore, tile size comprehensively affects both latency and energy consumption.

Current tensor compilers typically aim at optimizing latency, such as minimizing compute latency for compute-bound operators or memory latency for memory-bound operators [27]. However, these approaches often overlook considerations of energy and power consumption. We observe that imbalanced compute and memory latencies also increase energy consumption, meaning that the compiler's latency-optimal solution may not always be energy-optimal. We use an automatic tensor compiler to generate hundreds of codegen schedules for different DNN operators. By examining the impact of different codegen schedules on energy and execution time, we illustrate our insights.

## 4.2 Different Codegen Schedules

**Experimental Setup.** The experimental setup in this section includes a Tesla V100-SXM2-16GB GPU and an Intel Xeon E5-2667 v4 CPU, with CUDA version 12.2. The tested DNN operators include the compute-intensive Matmul and the memory-intensive ReduceMin and elementwise exponential (Exp) operators. The Matmul operation has a shape of [1024, 4096, 4096], while ReduceMin and Exp have a shape of [524288, 1024], which are commonly found in the Llama model. We use Microsoft Antares [3] v0.3.x to automatically generate operator code, as it is faster and more user-friendly compared to Ansor [26]. For measuring kernel energy consumption, we leverage interfaces from the NVML library. To better understand the kernel behaviors, we use NVIDIA Nsight Compute CLI (NCU) to profile the kernels.

**Matmul.** Figure 5 (a) illustrates the impact of different codegen schedules on the energy consumption and execution time of the Matmul operator. As Matmul is a compute-intensive operator, most kernels are bottlenecked by computation. This results in high GPU SM activity and consequently high power consumption, often nearing the power limit. Under similar power conditions, the energy consumption of Matmul is approximately linearly related to execution time.

Additionally, we observe that the energy-optimal candidate is not the same as the time-optimal candidate. This difference arises because the energy-optimal configuration employs a larger tile size, resulting in lower GPU utilization and reduced power consumption. Consequently, appropriately increasing the tile size can lead to energy savings.

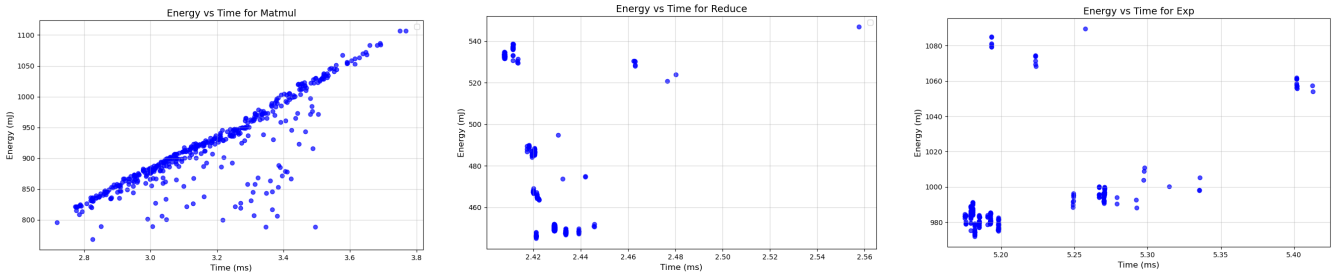
**Reduce and Exp.** Figure 5 (b) and (c) show the energy-time distribution for the Reduce and Exp operators. Unlike Matmul, memory-intensive operators exhibit relatively lower average power levels, and their energy consumption is not linearly related to execution time. Notably, for the Reduce operator, there are codegen schedules with similar execution times but significantly different energy consumption levels.

The NCU profiling results reveal that kernels with higher energy consumption use more shared memory to cache inputs and intermediate results, whereas the kernels with the lowest energy consumption avoid shared memory usage. However, as the performance of this operator is bound by global memory reads, the use of shared memory does not significantly contribute to throughput. Consequently, kernels with similar execution times but higher shared memory accesses consume more energy.

**Take away.** Our experimental observations indicate that solely optimizing computation latency for compute-bound operators leads to very high GPU power, thereby increasing overall energy usage. Similarly, solely focusing on reducing memory latency for memory-bound operators often introduces additional operations, which also increases GPU dynamic power consumption. Balance between computation and memory latency at the operator-level optimizer is important for energy optimization.

## 4.3 GPU Frequency Reshapes the Balance

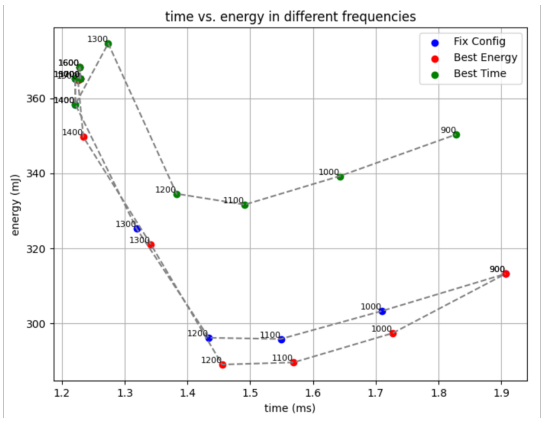
Similar to the graph-level optimizer, reducing GPU frequency decreases computational throughput but minimally impacts memory throughput. Consequently, the tile size that achieves balance between computation and memory latency at default frequencies becomes suboptimal at lower frequencies due to increased computational latency.



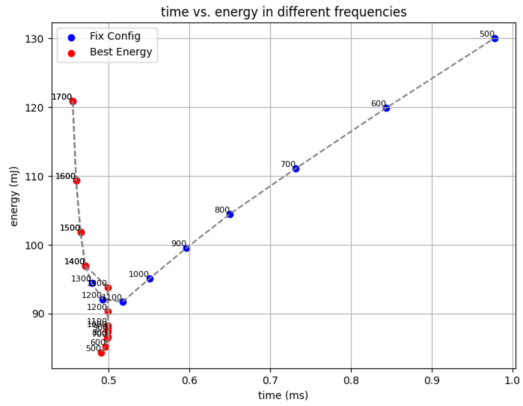

(a) Matmul (b) Reduce (c) Exp

Figure 5: Energy consumption and time of different codegen schedules for DNN operators.

**Experimental observation.** We experimentally explored the performance of different codegen schedules across varying GPU frequencies. In real-world training frameworks, memory-intensive operators often execute in combination with compute-intensive operators like Matmul or multiple memory-intensive operators, such as fused FlashAttention and RMSNorm kernels. In this section, we adopt workloads involving Matmul, RMSNorm, and FlashAttention from real training frameworks, using the same experimental setup as described in Section 4.




(a) FlashAttention




(b) RMSNorm

Figure 6: Energy consumption and time under different GPU frequencies.

**Matmul.** Our observations are as follows. First, for the Matmul operator, the optimal tile size configuration at the default frequency remains optimal at lower frequencies. Due to consistent results and space limitations, these findings are not shown graphically. This consistency occurs because tile sizes typically scale in powers of two, while frequency reductions are less impactful. Thus, tile size changes significantly overshadow frequency effects for individual operator kernels. However, for complex kernels combining various operators such as RMSNorm and FlashAttention, tile size impacts are more intricate, leading to frequency-dependent optimal configurations.

**RMSNorm and FlashAttention.** Figure 6 shows energy and time outcomes for RMSNorm and FlashAttention kernels with different tile sizes at various frequencies. First, FlashAttention kernels trade increased computation for reduced memory access, where tile sizes yield higher compute-to-memory ratios. However, due to lower computational throughput at reduced GPU frequencies, smaller tile sizes become necessary to maintain the compute-memory balance. Second, for the RMSNorm kernel, the default tile size causes a noticeable drop in compute throughput as GPU frequency decreases, making the kernel compute-bound and leading to slower execution. However, by adjusting the tile size, we can keep the kernel memory-bound, making its performance largely insensitive to frequency reductions.

**Take away.** Overall, adjusting tile sizes to balance computation and memory latency for kernels with complex operators like FlashAttention and RMSNorm achieves improved energy savings at different GPU frequencies.

## 5 Conclusion and Future Work

This project shows that jointly optimizing GPU frequency and compilation-level decisions significantly improves energy efficiency in large model training. For future work, we first plan to design a search algorithm to automatically explore the extensive configuration space to find an optimal energy-time frontier. Second, we intend to implement the energy optimizer atop Perseus and integrate it into existing training frameworks.

## References


[1] Openai keynote on building scalable ai infrastructure. https://www.servethehome.com/openai-keynote-on-building-scalable-ai-infrastructure/. Accessed: 2024-09-19.

[2] Accelerating a hugging face llama 2 and llama 3 models with transformer engine. https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/examples/te_llama/tutorial_accelerate_hf_llama_with_te.html, 2024.

[3] Microsoft antares. https://github.com/microsoft/antares, 2024.

[4] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, and Dejan S. Milojicic. Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 715–731, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947, New York, NY, USA, 2024. Association for Computing Machinery.

[6] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 178–191, New York, NY, USA, 2024. Association for Computing Machinery.

[7] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated End-to-End optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, Carlsbad, CA, October 2018. USENIX Association.

[8] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[9] Peng Cheng, Changho Hwang, Abhinav Jangda, Suriya Kalivardhan, Binyang Li, Shuguang Liu, Saeed Maleki, Madan Musuvathi, Olli Saarikivi, Aashaka Shah, Wei Tsui, and Ziyue Yang. MSCCL++: A gpu-driven communication stack for scalable ai applications. https://github.com/microsoft/mscclpp, 2024. Accessed: 2025-04-29.

[10] Sangjin Choi, Inhoe Koo, Jeongseob Ahn, Myeongjae Jeon, and Youngjin Kwon. EnvPipe: Performance-preserving DNN training framework for saving energy. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 851–864, Boston, MA, July 2023. USENIX Association.

[11] Jae-Won Chung, Yile Gu, Insu Jang, Luoxi Meng, Nikhil Bansal, and Mosharaf Chowdhury. Perseus: Reducing energy bloat in large model training, 2024.

[12] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. 2023.

[13] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

[14] Aaron Grattafiori et al. The llama 3 herd of models, 2024.

[15] James Hamilton. Constraint-driven innovation. Keynote talk at the Conference on Innovative Data Systems Research (CIDR 2024), 2024. Accessed: 2025-04-28.

[16] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 402–416, New York, NY, USA, 2022. Association for Computing Machinery.

[17] Malith Jayaweera, Martin Kong, Yanzhi Wang, and David Kaeli. Energy-aware tile size selection for affine programs on gpus. CGO '24, page 13–27. IEEE Press, 2024.

[18] Eric Masanet, Nuoa Lei, and Jonathan Koomey. To better understand ai's growing energy use, analysts need a data revolution. *Joule*, 8(9):2427–2436, 2024.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[20] Suchita Pati, Shaizeen Aga, Mahzabeen Islam, Nuwan Jayasena, and Matthew D. Sinclair. T3: Transparent tracking & triggering for fine-grained overlap of compute & collectives. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 1146–1164, New York, NY, USA, 2024. Association for Computing Machinery.

[21] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training, 2021.

[22] Guanhua Wang, Chengming Zhang, Zheyu Shen, Ang Li, and Olatunji Ruwase. Domino: Eliminating Communication in LLM Training via Generic Tensor Slicing and Overlapping. *arXiv preprint arXiv:2409.15241*, 2024.

[23] Haiquan Wang, Chaoyi Ruan, Jia He, Jiaqi Ruan, Chengjie Tang, Xiaosong Ma, and Cheng Li. Hiding communication cost in distributed llm training via micro-batch co-execution, 2024.

[24] Shibo Wang, Jinliang Wei, Amit Sabne, Andy Davis, Berkin Ilbeyi, Blake Hechtman, Dehao Chen, Karthik Srinivasa Murthy, Marcello Maggioni, Qiao Zhang, Sameer Kumar, Tongfei Guo, Yuanzhong Xu, and Zongwei Zhou. Overlap communication with dependent computation via decomposition in large deep learning models. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS 2023, page 93–106, New York, NY, USA, 2022. Association for Computing Machinery.

[25] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *USENIX NSDI*, 2023.

[26] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. Ansor: generating high-performance tensor programs for deep learning. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, OSDI'20, USA, 2020. USENIX Association.

[27] Hongyu Zhu, Ruofan Wu, Yijia Diao, Shanbin Ke, Haoyu Li, Chen Zhang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Wei Cui, Fan Yang, Mao Yang, Lidong Zhou, Asaf Cidon, and Gennady Pekhimenko. ROLLER: Fast and efficient tensor compilation for deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 233–248, Carlsbad, CA, July 2022. USENIX Association.

[28] Kan Zhu, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Yufei Gao, Qinyu Xu, Tian Tang, Zihao Ye, Keisuke Kamahori, Chien-Yu Lin, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. Nanoflow: Towards optimal large language model serving throughput, 2024.