Improved Model Sharding for Distributed LLM Training

Jimmy Shong[*] *jimmys2@illinois.edu*

Yiming Su[*] *yiming34@illinois.edu*

Kartik Ramesh[*] *kartikr2@illinois.edu*

Abstract

Large language models (LLMs) have emerged as a cornerstone of machine learning research, driving innovations across various natural language processing (NLP) tasks. The rapid growth of model sizes and the increasing hardware diversity in GPU clusters have amplified the importance of efficient distributed training on heterogeneous GPUs. Existing solutions, such as Alpa and Metis, however, struggle to address the unique challenges of such environments. This paper introduces a novel enhancement to the intra-operator DFS search mechanism, directly identifying and prioritizing stages most likely to cause out-of-memory (OOM) issues. Instead of waiting for the DFS search to reach these critical stages, the proposed method dynamically adjusts Data Parallelism (DP) and Tensor Parallelism (TP) values at runtime, accelerating the search process while reducing memory overhead. By focusing computational resources on resolving bottlenecks, this approach consistently discovers parallelism plans that match or outperform Metis's. Experimental results demonstrate the efficacy of the proposed method, achieving an 8–30% reduction in search steps and latency speedups ranging from 1.09× to 2.01×, all while maintaining comparable or improved performance. These findings highlight the potential of targeted, memory-aware search strategies for efficient distributed training in heterogeneous GPU environments. We have open-sourced our code here: `https://github.com/kartikx/Metis.git`.

1 Introduction

Large language models (LLMs), with billions to trillions of parameters, have become foundational in machine learning, achieving state-of-the-art performance and showcasing emergent abilities across diverse NLP tasks such as translation, code generation, and question-answering [1, 2, 3]. However, the large size of these models presents significant computational challenges, particularly regarding the memory and computational power required for their pre-training and post-training. Machine Learning models used to be possible to train on higher-end personal computers, such as PCs with RTX 4090 graphics card. In LLM era, it is no longer possible. The growing scale of these models and the increasing diversity of affordable GPU hardware have made efficient distributed training on heterogeneous clusters a critical research challenge.

Large-scale models rely on distributed training frameworks such as Alpa [4] and Metis [5] to partition model layers and weights across multiple GPUs, enabling training to proceed efficiently despite the significant computational and memory demands. However, the heterogeneity of modern GPU clusters, which often include devices of varying compute capabilities and memory capacities, poses unique challenges. Simply put, the same model layer can require different time budget to train with different types of GPUs, and thus meticulously placing specific layers on certain GPUs is critical to efficient distributed training. This problem only becomes more challenging when we start to partition layers by their weights. The diversity in GPU composition of a cluster necessitates more sophisticated strategies for workload partitioning and resource allocation, mainly when designing parallelism plans for data and model partitioning.

Recent systems like Alpa and Metis have aimed to address these challenges by automating the search for parallelism strategies in distributed environments. Alpa simplifies the problem by treating heterogeneous clusters as homogeneous, averaging GPU performance metrics to reduce search complexity. While this abstraction enables easier planning, it often leads to suboptimal results as it fails to account for the nuanced differences in computing power and memory capacities across devices. Metis improves on this approach by explicitly addressing heterogeneity, introducing a depth-first search (DFS) algorithm to navigate the space of potential parallelism plans. However, Metis's intra-operator search remains naive in its design, as it sequentially explores stages without prioritizing the ones indicated to cause out-of-memory (OOM) issues from the profiling data. This results in unnecessary search steps, increased overhead, and suboptimal utilization of resources.

[*] Equal contribution.

To address these limitations, we propose a novel intra-operator planner. Our heterogeneous cluster training planner addresses two critical challenges in distributed training: the complexity of heterogeneous GPU environments and the high overhead of exploring large search spaces.

To address the first challenge, our planner systematically deconstructs the intrinsic characteristics of diverse GPUs into fundamental components, making the optimization understandable and straightforward. It maintains all possible execution scenarios on heterogeneous GPUs, allowing its profiler to collect performance metrics based on comprehensive GPU information precisely. Further, our planner generates all possible combinations of device groups within a GPU cluster, accommodating both homogeneous and heterogeneous compositions. This enables our planner to identify whether a device group consists of heterogeneous GPUs, a critical feature for its cost estimator and parallelism planner to assess model partitioning strategies accurately.

For the second challenge, our planner reduces overheads in navigating its expanded hetero-aware search space. It achieves this by profiling only essential information, such as metrics for a subset of layers, while skipping repetitive ones and reusing metrics for structurally similar layers. Additionally, it employs a memory estimation model to calculate memory usage for layer compositions, minimizing the need for exhaustive profiling. Our planner optimizes its search by utilizing an algorithm that prunes the search space, filtering out redundant device group combinations. Once the device group combinations are filtered, the planner will iterate through each possible intra-operator for each given inter-operator plan. If profiling data suggests this plan will not result in OOM memory issues, the profiler selects this intra-op plan and moves on to the next inter-operator plan. If a potential OOM is detected, it will use capacity-aware load balancing across heterogeneous GPUs to improve performance across stages or adjust the model parallelism plan of the specific stage, causing the bottleneck. Our essential contribution is to have a planner capable of detecting these pipeline stages most likely to cause OOM issues, directly modifying their Data Parallelism (DP) and Tensor Parallelism (TP) configurations without waiting for the DFS search to encounter them. Focusing computational resources on resolving potential bottlenecks early significantly accelerates the search process, reduces memory overhead, and improves overall training efficiency.

We implemented this planner on top of Metis and evaluated it on two deep learning models across multiple heterogeneous GPU setups (A100, V100). Our experiments demonstrated that our planner improves search speed by 1.09-2.01x and achieves 8-30% reduction in search steps while maintaining the quality of the final training plan.

Specifically, our contributions are as follows:

1. We design a faster intra-operator search method that dynamically identifies and addresses stages prone to OOM issues, overcoming the inefficiencies of Metis's naive DFS approach.
2. We integrate dynamic adjustments to DP and TP configurations, ensuring optimal performance by prioritizing critical stages in the search space.
3. We demonstrate through extensive experiments that our method achieves an 8–30% reduction in search steps and wall-clock speedups of 1.09× to 2.01× compared to Metis, consistently matching or surpassing its performance.

2 Related Work

2.1 Automated Model Parallelism

Automated model parallelism frameworks have been developed to address the complexities of efficiently partitioning large-scale deep learning models across multiple devices. Metis [6] is a notable framework that combines pipeline parallelism (PP), tensor parallelism (TP) and data parallelism (DP) to optimize memory usage and training speed. By employing a co-design methodology, Metis determines an efficient mapping of model layers to device groups, balancing workload and communication overhead [6]. However, Metis performs a computationally intensive intra-operator search by naively iterating through all possible stage mappings to determine optimal configurations, which can lead to substantial overhead in scenarios with large and heterogeneous models. Our work builds upon Metis by introducing a memory-aware prioritization mechanism in the intra-operator search. Instead of exhaustively iterating through all stage mappings, our method identifies stages likely to encounter out-of-memory issues and adjusts their DP and TP values first. This significantly reduces the search space and enhances computational efficiency while maintaining robustness.

Other frameworks like AMP [7] and Alpa [4] have explored different strategies for automating parallelism. AMP [7] leverages heuristics to simplify the parallelism configuration process, targeting models with well-structured computational graphs. Alpa [4], on the other hand, introduces graph-based optimization for parallelization, providing flexible support for interleaved PP, DP, and tensor parallelism (TP). These approaches offer valuable insights into reducing human intervention in model parallelism planning but do not incorporate guided prioritization for memory-constrained stages, leaving room for inefficiencies in larger-scale training (Figure 1).

Galvatron [8] further extends the landscape by enabling fine-grained control over hybrid parallelism strategies and interleaved layers, but it remains less efficient in scenarios where memory bottlenecks dominate the optimization process. Our method stands apart by tightly coupling memory awareness with automated decision-making during intra-operator search, ensuring rapid convergence on practical solutions.

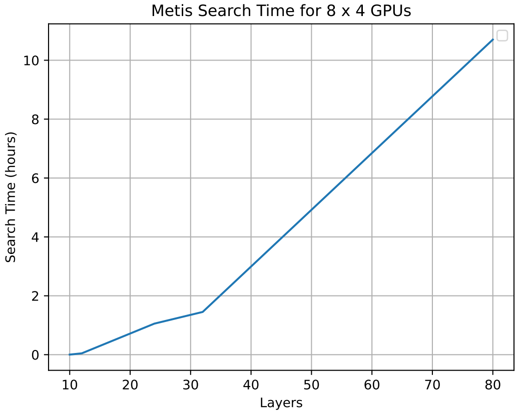

Figure 1: Metis search overhead increases significantly as the number of layers increase.

2.2 Systems for Model-Parallel Training

A variety of systems have been proposed to support model-parallel training by addressing challenges in computation, communication, and memory management. OneFlow [9] introduces a distributed computation graph optimization mechanism that minimizes communication bottlenecks and efficiently utilizes hardware resources. OneFlow's [9] unique contribution lies in its unified programming model, which simplifies the implementation of complex parallelism strategies. However, its reliance on homogeneity in hardware clusters limits its applicability in heterogeneous settings, where devices may have varied computational and memory capabilities. Our approach builds on OneFlow's strengths by explicitly addressing heterogeneity in resource planning and adapting DP and TP configurations dynamically for memory-constrained stages.

Gpipe [10] and PipeDream [11] focus on pipeline parallelism to optimize training throughput by overlapping computation and communication. Gpipe [10] introduces micro-batching as a way to mitigate pipeline stalls, enabling better utilization of all GPUs in a system. PipeDream [11] advances this by introducing weight stashing, which minimizes recomputation overhead during backward propagation. Despite these innovations, both systems rely on pre-defined scheduling strategies that lack the adaptability required for heterogeneous environments or dynamically changing workloads. Our work integrates the benefits of pipeline parallelism with memory-aware prioritization, dynamically adjusting configurations to maximize resource utilization while avoiding theoretical out-of-memory issues.

Megatron-LM [12] takes pipeline parallelism a step forward by utilizing tensor parallelism. Tensor Parallelism partitions the layer weights and computations of a stage into non-overlapping smaller chunks called tensors. These tensors are then distributed across multiple GPUs that constitute the TP group. By employing a combination of tensor, pipeline, and sequence parallelism, Megatron enable efficient training of large models across distributed GPU architectures. Megatron-LM minimizes memory bottlenecks and achieves near-linear scaling efficiency when training on hundreds or thousands of GPUs, and integrates seamlessly with PyTorch. Its support for mixed-precision training further enhances computational efficiency while reducing memory usage. However, while Megatron-LM excels in scalability and performance for homogeneous hardware setups, its reliance on uniform GPU clusters presents challenges in heterogeneous environments.

Additionally, newer systems such as ZeRO [13] and DeepSpeed [14] have focused on optimizing memory usage through techniques like partitioned activation storage and offloading to CPU or NVMe. While these innovations reduce the memory footprint during training, they do not address the search space complexity of determining optimal parallelism configurations. Our method complements these systems by streamlining the intra-operator search process, providing a mechanism to integrate their memory optimizations into a broader planning framework.

2.3 Heterogeneous DNN Training

The proliferation of heterogeneous hardware in modern computing clusters, including GPUs, TPUs, and specialized accelerators, has introduced new challenges for distributed DNN training. Frameworks such as SDPipe [15], HPH [16], and HetPipe [17] have been developed to address these challenges by optimizing workload partitioning and scheduling in heterogeneous environments. SDPipe [15] adopts a static-dynamic hybrid partitioning approach, where static layers are allocated based on device capabilities, and dynamic layers are adjusted during training to balance workload and communication. However, this approach assumes a relatively stable hardware configuration and does not account for dynamically emerging memory bottlenecks during training.

HPH [16] introduces a hardware-aware partitioning heuristic that incorporates device-specific characteristics such as memory capacity and computational throughput. By tailoring partitioning strategies to the underlying hardware, HPH improves training efficiency but lacks fine-grained control over interleaved parallelism strategies, which are increasingly important for modern transformer-based models. HetPipe [17], another prominent framework, optimizes workload distribution across heterogeneous devices by considering both device and model characteristics during partitioning. While HetPipe addresses critical aspects of heterogeneity, it does not incorporate prioritization mechanisms for resolving memory constraints at specific stages, which can lead to suboptimal configurations in large-scale training.

Our work extends these efforts by introducing heterogeneity awareness into the intra-operator search process. By clustering model layers and hardware accelerators into homogeneous groups during the planning phase, we effectively reduce the complexity of sharding decisions. Within each group, we apply memory-aware prioritization to ensure that configurations are optimized for the most constrained stages first. This approach combines the strengths of prior heterogeneous training frameworks with the memory-centric focus of our intra-operator search, providing a scalable solution for training large models on diverse hardware.

3 Method

3.1 Efficient Profiling

Profiling is a crucial component behind our planner's design. Profiling is necessary to estimate execution times and memory usage for different configurations. However, profiling all possible combinations of layers, device groups, and parallelism strategies is computationally prohibitive. To address this, the profiling process is streamlined by leveraging the repetitive structure inherent to many deep learning models. Instead of profiling each layer individually across all configurations, a representative subset of layers is selected. Metrics from these layers are extrapolated to estimate costs for similar layers. This drastically reduces the total number of profiling runs required.

Additionally, the profiling system focuses on unique device group configurations. Device groups are distinguished by GPU type, count, and node placement. For example, profiling results from groups comprising A100 GPUs are not reused for groups with V100 GPUs, as the compute and memory characteristics differ significantly. However, profiled results are shared across device groups within the same GPU type to avoid redundant computations. We can leverage heuristics learned from the machine learning community to make informed estimations about the latency and memory footprint of training using the same GPU type but with batch sizes and tensor parallel degrees not covered during the initial profiling. This selective reuse balances accuracy and efficiency, ensuring that critical differences in execution costs are captured without the overhead of exhaustive profiling.

While exploiting layer similarity has been introduced in past works before [18], they have not been utilized in conjunction with each other in the context of auto-parallelization or LLM pre-training.

3.2 Pruning the Search Space

One of the primary components of our approach is to prune the search space dynamically, focusing computational resources on the most promising configurations. Here, we leverage insights provided in previous works and implement a similar method [5]. Consider device groups and layer partitions with respect to distributed ML training on heterogeneous GPU clusters. Device groups are logical groupings of GPUs. Layer partitions are intra-layer parallelism mappings from model layers to GPUs. Here, we use a model of 10 layers, 4×A100 GPUs and 4×V100 GPUs to demonstrate our method.

Unpromising device groups to layer partition mappings are filtered out early based on profiling metrics and heuristic rules. We first profile imbalance device groups. For example, let us consider a PP stage planning:

stage 1:

$$L = 0,1,2,3,4$$

,

stage 2:

$$L = 5,6,7,8,9$$

.

Here, L represents the indices of layers that the stage is assigned. Let us assume that the layers are identical in computational requirements. With this setup, a PP stage planning of $(1 \times V100)$, $(3 \times V100, 4 \times A100)$ is unrealistic to achieve optimal performance. Similarly, stages that require excessive inter-cluster communications are de-prioritized in favor of mappings that localize data exchanges within high-bandwidth clusters. For an extreme example, a PP stage planning with 10 groups, where each odd index group is on one cluster and each even index group is on another one, is also unrealistic to achieve optimal performance due to time spent on inter-cluster communications, thus de-prioritized.

The pruning process also leverages the observation that similar device group configurations often yield identical performance. If we number the GPUs above as A_0, A_1, A_2, A_3 and V_0, V_1, V_2, V_3. Then a device group mapping of (A_0, A_1, V_0, V_1) should not differ much, if at all, from (A_2, A_3, V_2, V_3). Here, we consider similar as in the numbers of unique GPUs in the device groups are similar. Therefore, for any PP stage with layers, it should not matter which device group above it maps to. Thus, we should only leave one of the device groups in the search space and eliminate the other. By grouping such configurations and evaluating only representative samples, the

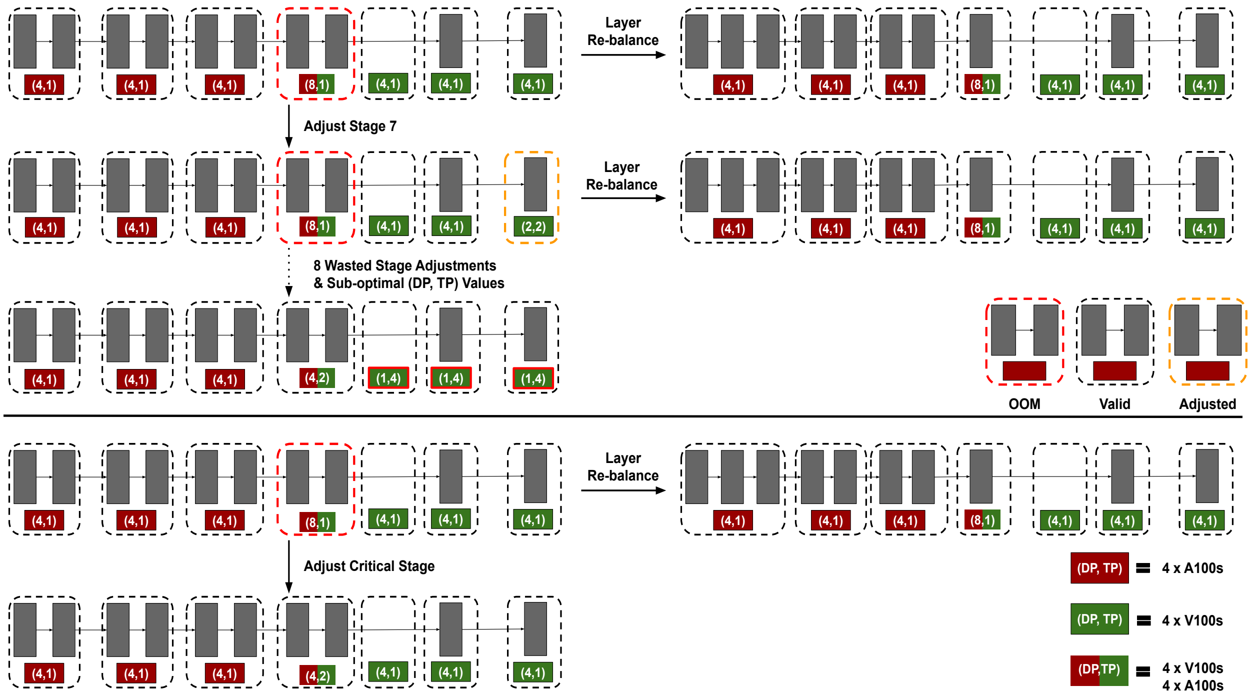

Figure 2: We can see how Metis (top) performs many unnecessary and redundant searches to find a potential non-rebalanced candidate. Our planner (bottom) can immediately identify that the 4th stage (marked in red) is the cause of the memory bottleneck and correctly adjusts the DP, TP value to alleviate the memory pressure. This allows us to find the candidate plan far more quickly than Metis.

search algorithm reduces redundant computations without sacrificing plan quality.

3.3 Load Balancing Across Heterogeneous GPUs

Traditional DL frameworks like Megatron-LM [12] and Deep-Speed [14] support uniform partitioning of data and layers, which is tailored for homogeneous GPUs. However, non-uniform partitioning becomes more valuable when it comes to heterogeneous GPUs as they differ in computing and memory capacities.

To maximize performance, the algorithm incorporates load balancing mechanisms that distribute computational and memory loads proportional to the capabilities of the GPUs in each device group. These mechanisms fall into two categories: layer load balancing, which manages the layer distribution across PP stages, and data load balancing, which adjusts local batch sizes across GPUs in a DP group. Faster GPUs with higher memory capacities are assigned a more significant share of the workload, while slower GPUs handle fewer layers or smaller micro-batches. This balancing process ensures no GPU becomes a bottleneck due to under-utilization or excessive memory pressure.

The load-balancing strategy is tightly integrated with the profiling system. Execution times and memory usage for individual layers are estimated based on profiled data, allowing the algorithm to allocate workloads in a way that minimizes variance in execution time across device groups. By aligning the workload distribution with each GPU's compute and memory characteristics, the system achieves high efficiency, even in highly heterogeneous clusters.

3.4 Dynamic Search with Heterogeneity Awareness

Our novel search algorithm leverages the metrics from the profiling data to explore device-group-to-layer mappings systematically. Unlike traditional methods that treat all stages equivalently, this implementation identifies stages likely to encounter out-of-memory (OOM) issues and prioritizes them for adjustment. By narrowing down the search space to focus on high-risk stages, the algorithm significantly reduces the search overhead, particularly in scenarios involving large GPU clusters or highly heterogeneous environments.

The search process begins by assigning layers to stages based on their computational and memory demands, which is determined by the capacities of the available GPU devices. To each device group, we assign the maximum degree of DP that can be supported by the device group, and a TP degree of 1. This insight stems from our evaluation that prioritizing DP leads to faster training time (Figure 3).

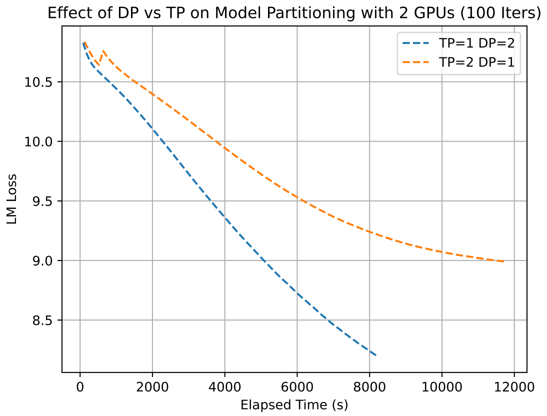

Figure 3: Effect of DP vs TP training time for GPT-2.

The algorithm dynamically evaluates the data parallelism and tensor parallelism configurations for each stage, using estimated memory usage to identify potential OOM risks. When a stage is flagged, its DP and TP values are adjusted dynamically within the search process rather than relying on post hoc corrections. We start off with Specifically, the DP value gets halved by 2 and the TP value gets multiplied by 2. Decreasing DP reduces memory pressure on individual GPUs, and TP is increased to maintain full utilization of the device group. This approach ensures that problematic stages are addressed as soon as potential OOM issues are detected, enabling efficient exploration without the inefficiencies of a naive, stage-by-stage DFS search. This proactive adjustment ensures efficient convergence while maintaining high throughput and avoiding memory errors.

The proposed profiling and search method significantly accelerates the discovery of efficient parallelism strategies for distributed training. By focusing on stages likely to encounter OOM issues, the system avoids unnecessary iterations through the search space, leading to faster search speeds. This approach balances thorough exploration and practical feasibility, making it especially valuable in distributed training, where combining large models and heterogeneous hardware can make exhaustive exploration prohibitively expensive.

4 Evaluation

To validate the efficacy and scalability of our proposed planner, we conducted a rigorous set of experiments aimed at:

1. demonstrating the computational efficiency of our planner compared to the state-of-the-art Metis framework, and
2. ensuring that the quality of the generated plans remains uncompromised despite improvements in speed and scalability.

4.1 Evaluation Plan

To validate the efficacy and scalability of our proposed planner, we conducted a rigorous set of experiments aimed at addressing two core objectives: (1) demonstrating the computational efficiency of our planner compared to the state-of-the-art Metis framework, and (2) ensuring that the quality of the generated plans remains uncompromised despite improvements in speed and scalability.

For the baseline, we utilized the profiling data provided by the Metis authors for a 10-layer, 2.5B parameter GPT-2 model. The profiling data consists of essential metrics such as the number of parameters in each layer and the latency for both forward and backward passes, which are critical for accurately modeling distributed training workloads. To stress-test the scalability of our method, we extended this evaluation to a theoretical 20-layer, 4.5B GPT-2 model by synthesizing profiling data for larger models. This synthetic data was generated using community-validated heuristics that account for larger batch sizes and increased tensor parallelism (TP) degrees, simulating conditions representative of real-world distributed training environments. All clusters we tested on were heterogeneous clusters made up of nodes with $4 \times A100s$ or $4 \times V100s$. The clusters were split in a 50/50 manner between A100s and V100s

The primary metrics used to evaluate our planner are the number of search steps, wall-clock search time, and simulated cost of the top plan generated by each planner. Our hypothesis is that our planner will demonstrate significant improvements in search efficiency and latency while maintaining parity in plan quality. Achieving this balance would establish our planner as a scalable and efficient alternative for heterogeneous large language model (LLM) training workloads, especially in multi-GPU environments where planning overhead can become a bottleneck.

4.2 Evaluation Results

Table 1 highlights the reduction in search steps achieved by our planner compared to Metis. For a 10-layer model, our method achieved a modest 8.17% reduction in search steps. However, as the model scaled to 20 layers, this reduction grew to 29.05%, illustrating the scalability of our approach. These improvements are attributed to our prioritization of high-risk stages, enabling faster convergence to a valid plan.

#Layer	#A100	#V100	Steps			Wall-clock Time (ms)			Costs		
			Metis	Ours	Reduction	Metis	Ours	Speedup	Metis	Ours	Saving
10	16	16	612	562	**8.17%**	95.34	87.34	**1.092x**	1898.37	1793.60	**104.77**
20	32	32	58078	41208	**29.05%**	17119.40	8498.32	**2.014x**	2078.56	2005.78	**72.78**
20	64	64	3734	3482	**6.749%**	857.18	511.60	**1.675x**	1101.65	1098.34	**3.31**
40	128	128	1714619	1553492	**9.39%**	756970.88	407860.06	**1.856x**	1154.72	1120.22	**34.5**

Table 1: Comparison of our method with Metis across steps, wall-clock time, and costs. Our planner has a lower search step count, a faster wall-clock search time, and returns better or equivalent plans to Metis

Table 1 compares the search latency of our planner against Metis. While the improvements for smaller models are minor (1.092× speedup), the latency reduction becomes substantial for larger models, achieving a 2.014× speedup for the 20-layer configuration. This result highlights that our reduction in search steps translates directly to tangible runtime improvements, which is critical for accelerating the planning phase in real-world deployments.

The validity of our planner hinges on maintaining or improving the quality of the generated plans. As shown in Table 1, the costs of the top plans generated by our method are identical to those produced by Metis across all tested configurations. This ensures that the speed and efficiency gains of our method do not compromise the quality of the plans, making it a viable alternative for production systems.

5 Discussion

5.1 Takeaways

The results from our evaluation demonstrate the clear advantages of our planner in both efficiency and scalability. The ability to reduce search steps and latency while maintaining plan quality highlights the robustness of our method and its suitability for large-scale heterogeneous LLM training workloads.

The reduction in search steps, particularly for larger models, directly translates to a more streamlined planning phase. This improvement is critical in distributed training scenarios where frequent re-planning might be necessary due to dynamic changes in resource availability. Similarly, the substantial reduction in search latency allows practitioners to iterate more rapidly, ultimately reducing the overall time-to-solution for training massive language models.

This improvement in search steps and wall-clock time can be directly attributed to the fast OOM adjustment. As the number of layers and the number of GPUs increase, the search space of optimal solutions also grows. It becomes more likely for the search algorithm to first generate a training configuration which is OOM. By immediately identifying the OOM stage and adjust accordingly while not drastically modifying the pipeline parallelism structure, we reduce the number of search steps compared with Metis.

Our findings also underscore the scalability of our approach. As model complexity grows, the benefits of our planner become increasingly pronounced, making it a compelling choice for next-generation LLM workloads. Furthermore, the equivalence in plan quality between our method and Metis ensures that users can adopt our planner without any concerns about a drop in performance, thus making it an effective drop-in replacement.

5.2 Limitations

While our results are promising, there are still limitations we need to work on in the future.

Model One major drawback is we have only tested our system in Metis trace on GPT-2 profiling data, assuming all in-between layers are identical. One immediate next step is to extend our evaluation to larger models such as GPT-3's scale and to more complex LLM architectures like Mix-of-expert models. We speculate that our improvements over previous methods will persist as the improvements do not rely on model.

Hardware Current evaluation only consider two types of GPUs. We plan to incorporate more diverse hardware configurations to test the generalization of our method. We aim to explore optimizations for highly heterogeneous setups involving accelerators with varying performance characteristics. Aside from compute, we also wants to incorporate various network transport setup like Infiniband, RoCE as well as Ethernet.

Fidelity Our current evaluation are based on profiling result, and the performance of different parallelism strategies are predicted via simulation. Simulation is fast and accessible, but its fidelity depends on how accurate the internal model used in simulator. To ensure our system is faithful and could be applied into real deployment, we need cross valiate the result with other source.

Parallelism Despite data parallelism, tensor parallelism and model parallelism are the most essential one used in distributed training. There are emerging strategies like FSDP(fully-sharded DP), sequence parallelism and context parallelism that are also been used in LLM training. Our goal is to cover all those strategies into the search space.

5.3 Future Work Directions

There are some directions we can work on to improve our system.

Incorporate Heuristic Techniques in Searching Integrating reinforcement learning or heuristic-based adaptive search techniques could further enhance the efficiency of our planner in more complex scenarios. As mentioned previously with discussions of figure 3, in certain cases, prioritizing DP or TP depends on the number of GPUs a cluster has. This knowledge serves as a good starting point for heuristic-based searching. A potential future direction is to consider other heuristics in the search algorithm. For example, it is well understood that in modern day LLM training, hardware failures, such as NVLink failures, are extremely common in large datacenters, causing latency or even wasted progresses in training [19]. Including heuristics such as hardware failure frequency in the profiling data can enable the search algorithm to make more realistic model placement decisions. This improvement can further aid previous system algorithm co-design approaches such as MegaScale to further boost the system's training efficiency [20].

Reduce Time Complexity of Search Algorithm Although we have achieved improvements over previous methods, our runtime is still limited by the search algorithm itself. DFS as a graph search algorithm has a O(n) time complexity, which means our search time linearly grows with number of layers in the model. At the same time, there are other search algorithms with lower time complexity that have been leveraged in other subfields of Machine Learning, such as the Successive Halving Algorithm (SHA) proposed by Kevin Jamieson and Ameet Talwalkar [21]. We can consider the search for the best model placement as a type of hyperparameter searching: the placements of model weights on different GPUs leads to different training times. Here, we consider wall-clock time as the optimization metrics, and we can attempt to use SHA to minimize the training time by successively eliminating slower configurations. For example, we generate different model placements based on given GPUs and cluster information. Then, we attempt to train each configuration by 1 iteration and record the time, and eliminate configurations that are considered slower than the majority. Admittedly, this approach spends more time than our approach or Metis by order of magnitude. However, this approach does not require users to provide profiling data beforehand, which saves time. Li et al. has published a work on an asynchronous version of SHA, which can be a potential start on leveraging SHA on GPU clusters without profiling to find the best model placement [22].

Support Asynchronous Training Lastly, another potential future direction is to consider asynchronous training for LLMs. All of our experiments, and related previous works, assume that the produced model placement configuration will be run as a synchronous training job. However, is is not hard to observe that in synchronous training, one or more training processes may run slower than other processes, causing the them to finish a training iteration slower. Thus, other finished processes would have to wait for the stragglers to catch up in order to start the next training iteration. This incurs a major bottleneck in synchronous training. In asynchronous training, the system is allowed to make progress even without all processes' results. Our approach might not directly work on asynchronous training as we do not consider the situation where the system is allowed to make partial progress. We do not consider the cost of slower GPUs catching up to the cluster's progress in the algorithm. We also do not consider the correlated cost of how to place different model layers on different training devices while considering the possible straggling situations. Lastly, our approach prefers homogeneous model composition. For training other large DNN models with different model compositions, the computation model can also affect how the weights should be placed. In general, future research should consider how would asynchronous training affect the best model placement strategy.

6 Conclusion

In this paper, we propose a prototype solution to address the limitations of Metis's naive DFS-based intra-operator search for distributed training on heterogeneous GPU clusters by introducing a prioritized approach that dynamically targets stages prone to out-of-memory (OOM) issues. Building on Metis's heterogeneity-aware framework, our method accelerates the search process by directly modifying Data Parallelism (DP) and Tensor Parallelism (TP) configurations for critical bottleneck stages, reducing memory overhead and unnecessary search steps. Experimental results demonstrated up to a 30% reduction in search steps and wall-clock speedups of 1.09× to 2.01×, consistently matching or surpassing Metis's performance. This work underscores the importance of targeted, memory-aware search strategies for efficient distributed training and lays the groundwork for future advancements in adaptive parallelism planning and broader support for multi-vendor GPU environments.

References

[1] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed Huai hsin Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *ArXiv*, abs/2206.07682, 2022.

[2] OpenAI Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haim ing Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Benjamin Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Sim'on Posada Fishman, Juston Forte, Is abella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Raphael Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Lukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Hendrik Kirchner, Jamie Ryan Kiros, Matthew Knight, Daniel Kokotajlo, Lukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Ma teusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel P. Mossing, Tong Mu, Mira Murati, Oleg Murk, David M'ely, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Ouyang Long, Cullen O'Keefe, Jakub W. Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alexandre Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Pondé de Oliveira Pinto, Michael Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack W. Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario D. Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin D. Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas A. Tezak, Madeleine Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cer'on Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll L. Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qim ing Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report. 2023.

[3] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson,

Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Sax-

ena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024.

[4] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578, 2022.

[5] Taegeon Um, Byungsoo Oh, Minyoung Kang, Woo-Yeon Lee, Goeun Kim, Dongseob Kim, Youngtaek Kim, Mohd Muzzammil, and Myeongjae Jeon. Metis: Fast automatic distributed training on heterogeneous GPUs. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 563–578, Santa Clara, CA, July 2024. USENIX Association.

[6] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[7] Dacheng Li, Hongyi Wang, Eric Xing, and Hao Zhang. Amp: Automatically finding model parallel strategies with heterogeneity awareness. *Advances in Neural Information Processing Systems*, 35:6630–6639, 2022.

[8] Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. Galvatron: Efficient transformer training over multiple gpus using automatic parallelism. *arXiv preprint arXiv:2211.13878*, 2022.

[9] Jinhui Yuan, Xinqi Li, Cheng Cheng, Juncheng Liu, Ran Guo, Shenghang Cai, Chi Yao, Fei Yang, Xiaodong Yi, Chuan Wu, et al. Oneflow: Redesign the distributed deep learning framework from scratch. *arXiv preprint arXiv:2110.15032*, 2021.

[10] Byungsoo Jeon, Mengdi Wu, Shiyi Cao, Sunghyun Kim, Sunghyun Park, Neeraj Aggarwal, Colin Unger, Daiyaan Arfeen, Peiyuan Liao, Xupeng Miao, et al. Graphpipe: Improving performance and scalability of dnn training with graph pipeline parallelism. *arXiv preprint arXiv:2406.17145*, 2024.

[11] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

[12] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

[13] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[14] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020.

[15] Xupeng Miao, Yining Shi, Zhi Yang, Bin Cui, and Zhihao Jia. Sdpipe: A semi-decentralized framework for heterogeneity-aware pipeline-parallel training. *Proceedings of the VLDB Endowment*, 16(9):2354–2363, 2023.

[16] Yabo Duan, Zhiquan Lai, Shengwei Li, Weijie Liu, Keshi Ge, Peng Liang, and Dongsheng Li. Hph: Hybrid parallelism on heterogeneous clusters for accelerating large-scale dnns training. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 313–323. IEEE, 2022.

[17] Jay H Park, Gyeongchan Yun, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jaesik Choi, Sam H Noh, and Young-ri Choi. {HetPipe}: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 307–321, 2020.

[18] Liangxin Liu, Xuebo Liu, Derek F. Wong, Dongfang Li, Ziyi Wang, Baotian Hu, and Min Zhang. Selectit: Selective instruction tuning for large language models via uncertainty-aware self-reflection, 2024.

[19] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, Yonggang Wen, and Tianwei Zhang. Characterization of large language model development in the datacenter, 2024.

[20] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. Megascale: Scaling large language model training to more than 10,000 gpus, 2024.

[21] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization, 2015.

[22] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.