HEXISCALE: FACILITATING LARGE LANGUAGE MODEL TRAINING OVER HETEROGENEOUS HARDWARE

Ran Yan [*1] **Youhe Jiang** [*1] **Xiaonan Nie** [2] **Fangcheng Fu** [3] **Bin Cui** [2] **Binhang Yuan** [1]

ABSTRACT

Training large language models (LLMs) is a computationally intensive task, which is typically conducted in data centers with homogeneous high-performance GPUs. In this paper, we explore an alternative approach by deploying training computations across heterogeneous GPUs to enable better flexibility and efficiency for heterogeneous resource utilization. Toward this end, we propose a novel system, HEXISCALE, that can flexibly support asymmetric partition of training computations in the scope of data-, pipeline-, and tensor model parallelism. We further formalize the allocation of asymmetric partitioned training computations over a set of heterogeneous GPUs as a constrained optimization problem and propose an efficient hierarchical graph partitioning algorithm. Our approach effectively allocates training computations across heterogeneous GPUs, fully leveraging the available computational power. We compare the performance of HEXISCALE with state-of-the-art homogeneous and heterogeneous training systems. When training LLMs at different scales (from 7B to 30B), empirical results demonstrate that: (i) compared to state-of-the-art homogeneous baselines running over homogeneous GPUs, HEXISCALE achieves *similar* performance when running over heterogeneous GPUs with the *same* theoretical FLOPS; (ii) compared to state-of-the-art heterogeneous baselines running on the same heterogeneous clusters, HEXISCALE delivers $1.5\times$ to $2.4\times$ higher throughput.

1 INTRODUCTION

Over the past few years, large language models (LLMs) have demonstrated impressive performance and sparked a new wave of exciting AI applications (Bommasani et al., 2021). However, training these LLMs, such as GPT (OpenAI, 2024), Claude (Anthropic, 2024), Gemini (Reid et al., 2024), Llama (Touvron et al., 2023; Dubey et al., 2024), Mixtral (Jiang et al., 2024a), Yi (Young et al., 2024), Falcon (Institute, 2023), etc., can be extremely computation-intensive, often involving thousands of GPUs running for months. The high cost of deploying such training tasks in a cluster with homogeneous GPUs has become an obvious obstacle limiting the evolution of LLMs. We explore an alternative approach by *distributing training computations across heterogeneous GPUs*, enabling greater flexibility in resource utilization and democratizing LLM training.

Distributing parallel training computations across heterogeneous GPUs is a natural option to democratize LLM training. In the current exciting era of generative AI, chip vendors typically release new generations of AI chips every 24 months. For instance, Nvidia introduced the Turing architecture in 2018 (Nvidia, 2018), Ampere in 2020 (Nvidia, 2020), Hopper in 2022 (Nvidia, 2022), and Blackwell is scheduled for Q4, 2024 (Nvidia, 2024). On the other hand, one particular version of an AI chip often remains in use by cloud service platforms, technology companies, or research institutions for a much longer period. For example, K80 GPUs with Tesla architecture, released in 2006 (Nvidia, 2006), are still available on AWS as `p2` instances (Amazon, 2024). This observation highlights the important opportunity to explore effective ways to maximize the efficiency of such widely available yet heterogeneous hardware to facilitate more cost-effective and accessible LLM training.

On the other hand, deploying large-scale training computation for LLM over a set of heterogeneous GPUs with different technique specs would be a challenging task regarding training system design and implementation. To effectively distribute the training computation over thousands of GPUs, the state-of-the-art training systems, like Megatron (Narayanan et al., 2021b) and DeepSpeed (Rajbhandari et al., 2020) usually supports: (i) tensor model parallelism (Narayanan et al., 2021b; Nagrecha, 2021); (ii) pipeline parallelism (Huang et al., 2019; Narayanan et al., 2019; Yang et al., 2021; Narayanan et al., 2021a); and (iii)

[*]Equal contribution [1]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China [2]School of Computer Science, Peking University, Beijing, China [3]School of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Binhang Yuan <biyuan@ust.hk>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026. Copyright 2026 by the author(s).

data parallelism (with potentially sharded implementations of parameters, gradients, and optimizer states across multiple devices, also known as fully sharded data parallelism) (Rajbhandari et al., 2020; Ren et al., 2021; Rajbhandari et al., 2021; Jiang et al., 2023). However, these systems typically only support homogeneous configurations, which require the entire training cluster to operate under a fully symmetric setup. This means all tensor model parallel groups must have the same degree of parallelism, and the same applies to pipeline parallel groups, as well as data or optimizer parallel groups. Such implementation assumes all the GPUs take the *same* amount of computation load, which significantly limits the system efficiency when deploying the training computation over GPUs with *different* computation capability (measured by the peak FLOPS), *different* device memory (i.e., HBM) capacity, and *different* network bandwidth for each pair of GPUs (inter- and intra-node).

Concretely, there are two fundamental challenges stemming from the *heterogeneity*:

- **Different GPU computation capability and memory capacity.** Heterogeneous GPUs can vary significantly in terms of computation capability (i.e., FLOPS) and memory capacity. This disparity poses a challenge in distributing computations across all available resources. Without proper resource management, the most capable GPUs may remain underutilized while less powerful GPUs become performance bottlenecks, resulting in system inefficiencies and prolonged training duration. Therefore, partitioning computations to match the capabilities of each GPU is essential to fully utilize the available hardware.
- **Different GPU-GPU network bandwidth.** The heterogeneous connections between GPUs, ranging from high-speed NVLink and PCIe to standard Ethernet, add another layer of complexity. These varying connection speeds result in uneven communication times. Effective management of communication overhead is essential to prevent faster connections from stalling, waiting for slower ones. This requires sophisticated optimization algorithms to align communication patterns with the underlying hardware capabilities, ensuring smooth data flow across GPUs.

From a system implementation perspective, existing systems such as Megatron (Narayanan et al., 2021b) adopt a fully symmetric partitioning strategy, which often leads to GPU under-utilization due to the inherently asymmetric nature of heterogeneous environments. From a scheduling perspective, existing works—including Alpa (Zheng et al., 2022) and Galvatron (Miao et al., 2022)—design high-complexity algorithms that cannot be trivially adapted to heterogeneous environments (detailed in 2.1). To overcome these challenges, we propose a novel framework, HEXISCALE, that coordinates distributed LLM training over a set of GPUs with different computation capabilities and connections.

Our contributions are summarized as:

Contribution 1. We implement HEXISCALE, a heterogeneous LLM training system, which supports asymmetric partition for the training computation within the scope of data, pipeline, and tensor model parallelism to flexibly accommodate the heterogeneity of various GPUs and diversified network connections. Such designs enable fine-grained partitioning of the original training computation to fully exploit the heterogeneous computational capabilities.

Contribution 2. We formulate the scheduling problem of allocating the LLM training computation over a set of heterogeneous GPUs as a constrained optimization problem. To solve this problem efficiently, we propose a two-phase optimization approach that employs a graph partitioning algorithm to effectively coordinate parallel strategies for the given devices. In the first phase, the algorithm splits available GPUs into multiple groups, each of which forms a pipeline in the second phase. We then iteratively apply the two-phase algorithm to determine optimal parallel strategies for the set of heterogeneous GPUs.

Contribution 3. We evaluate HEXISCALE through experiments. We compare the system efficiency between heterogeneous settings (enabled by HEXISCALE) and standard homogeneous settings within a centralized data center (enabled by Megatron, Galvatron, and FSDP). We conduct these comparisons on training the popular LLM models with different model sizes, including LLAMA-2 (7B), LLAMA-2 (13B), and LLAMA (30B). The results demonstrate that, given the *same* theoretical FLOPS, HEXISCALE achieves performance comparable to state-of-the-art LLM training frameworks running in a homogeneous data center. HEXISCALE also operates efficiently in heterogeneous environments and outperforms state-of-the-art heterogeneous training systems (e.g., AMP (Li et al., 2022), Espresso (Zhou et al., 2025), and Metis (Um et al., 2024)). The results show that HEXISCALE offers $1.5\times$ to $2.4\times$ higher throughput.

The rest of the paper is organized as follows. §2 provides a brief review of optimization techniques for parallelizing LLM training and summarizes related work on auto-parallelization on homogeneous clusters and heterogeneity-aware training scheduling. In §3, we present a case study and introduce the design of HEXISCALE, which utilizes fully asymmetric parallelism to optimize training in heterogeneous environments. In §4, we formalize the allocation of asymmetric partitioned training computations over a set of heterogeneous GPUs as a constrained optimization problem and propose an efficient solution based on a hierarchical graph partitioning algorithm. In §5, we evaluate the system performance of HEXISCALE and the scheduling algorithm.

2 PRELIMINARY AND RELATED WORK

We first discuss representative work on automatic parallelization for homogeneous setups in §2.1, and finally summarize recent efforts on heterogeneity-aware training scheduling from the machine learning community in §2.2.

2.1 Auto Parallelization on Homogeneous Clusters

Significant efforts have been made to automatically identify optimal parallel configurations for LLM training on homogeneous clusters (Zheng et al., 2022; Miao et al., 2022; Santhanam et al., 2021; Chen et al., 2023; Tarnawski et al., 2021; Fan et al., 2021). For example, Alpa (Zheng et al., 2022) proposes a two-level hierarchical scheduling algorithm based on ILP and dynamic programming. Galvatron (Miao et al., 2022) introduces another dynamic programming-based scheduling algorithm for transformer models. These auto parallelization algorithms effectively identify optimal parallel configurations for homogeneous clusters. However, they assume symmetric setups in terms of network connectivity and hardware specifications, and employ high-complexity algorithms. Consequently, in heterogeneous environments, these algorithms often require prohibitively long execution times (e.g., several hours for hundreds of GPUs) while traversing a limited search space and producing parallel configurations that yield suboptimal performance. (See §5.)

2.2 Heterogeneity-aware LLM Training

Significant efforts have been made to build training systems tailored for heterogeneous resources (Miao et al., 2022; Zheng et al., 2022; Unger et al., 2022; Wang et al., 2024a; Yuan et al., 2022; Yi et al., 2020; Miao et al., 2021; Zhang et al., 2022). For example, SDPipe (Miao et al., 2023) implements flexible data parallelism synchronization schemes to address slow data parallel communication issues in (semi)-heterogeneous environments; Whale (Jia et al., 2022) proposes a hardware-aware load-balancing algorithm to accelerate heterogeneous training.

Among recent work, AMP (Li et al., 2022), Espresso (Zhou et al., 2025), and Metis (Um et al., 2024) represent the most relevant efforts toward supporting heterogeneous training and scheduling for the optimal parallel configurations (across data-, pipeline-, and tensor model-parallelism). AMP and Espresso are built on top of DeepSpeed (Rajbhandari et al., 2020); they support heterogeneous layer partitioning and perform scheduling within this design. However, they fall short of fully leveraging the potential of heterogeneous clusters due to limited support for heterogeneity-aware parallelism, which restricts the search space explored by their algorithms. Metis partitions computations into a single pipeline with a variable number of stages, each of which can have a distinct combination of data and tensor model parallelism degrees. To identify high-performance parallelism plans for heterogeneous GPUs, Metis introduces a novel depth-first-search (DFS) and heuristic-based approach (see §4.2 of the Metis paper). Although Metis provides fine-grained heterogeneous-aware parallel support, it has two limitations: (i) the introduced data-parallel-first DFS scheduler prioritizes data parallelism over tensor model parallelism unless an out-of-memory error occurs, whereas optimal throughput often comes from a balanced mix of data- and tensor model-parallelism. Due to these algorithmic limitations, strategy found by Metis's algorithm often only involve data and pipeline parallelism, resulting in inflated pipeline-bubbles, and up to 1.9× lower throughput than HEXISCALE; and (ii) Metis's DFS-based search algorithm scales poorly: it runs for hours and still struggles to find optimal strategies for 240-GPU (see §5.5). In contrast, HEXISCALE is both capable of fully supporting the asymmetric partition of the parallel training computations (see §3, §5) and has a more efficient and effective scheduling algorithm in identifying the optimal parallel strategy (see §4).

3 SYSTEM DESIGN AND IMPLEMENTATION

In this section, we start with a case study on heterogeneous training with Megatron, then introduce the system design of HEXISCALE and discuss how HEXISCALE improves the training efficiency. We present full comparisons with homogeneous frameworks (Megatron), auto-parallelization work (Galvatron), and heterogeneous frameworks (AMP, Espresso, and Metis) in §5.

3.1 Case Study: Parallelism over Heterogeneity

Consider training a LLAMA-2 (13B) model in a heterogeneous environment on cloud with the following machines: machine *A* is equipped with `3×A800-80G` GPUs connected via NVLINK, which offers a intra-machine bandwidth of 200 GB/s; machine *B* has `3×4090-24G` GPUs connected via PCIe with a bandwidth of 32 GB/s; and machine *C* is equipped with `2×3090-24G` GPUs, connected via PCIe with a 16 GB/s bandwidth.[1] The machines are interconnected using a 1 GB/s Ethernet link. To motivate our system design compared to Megatron, we simulate performance on a training task with a global batch size of 24, and a micro-batch size of 1. Activation recomputation is applied for both systems.

Training with Megatron. Megatron only supports a fully symmetric partition of the training computation. Denote D_{dp}, D_{tp}, D_{pp} as the parallel degrees for data-, tensor

[1] 3-GPU-host exists commonly in practice. For example, cloud providers (e.g., AWS (Amazon Web Services, 2024)) often manage resources via Slurm-based approaches, which may expose leftover 3-GPU when the rest five GPUs are allocated elsewhere.

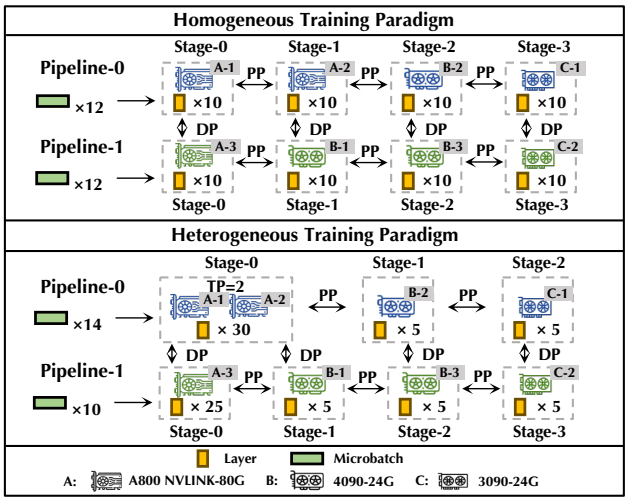

Figure 1. Case study on comparing Megatron and HEXISCALE. Both systems run their optimal parallel strategies.

model-, and pipeline- parallelism. The considered parallel strategies on these 8 GPUs include: (i) plan 1: $(D_{dp} = 1, D_{tp} = 1, D_{pp} = 8)$; (ii) plan 2: $(D_{dp} = 1, D_{tp} = 2, D_{pp} = 4)$; (iii) plan 3: $(D_{dp} = 2, D_{tp} = 1, D_{pp} = 4)$.

Plan 1 employs a pipeline with eight pipeline stages, which is inefficient due to high bubble cost and imbalanced computation. Even if the micro-batch size is set as 1 (which may reduce GPU kernel efficiency), bubble time is still over 22% of the pipeline execution time. The system performance is further compromised by bottleneck machine *C*, which computes 3.9× and 2.1× slower than machines *A* and *B*.

Plan 2 introduces inter-machine tensor model parallelism, significantly degrading the performance with an estimated per-layer communication cost of 1.88s, given 1 GB/s bandwidth and batch size as 24. This is a huge overhead, compared to the 0.01s communication cost via NVLINK.

Plan 3 is a potentially good configuration. We fine-tune this plan by maximizing the number of transformer layers that use high intra-machine bandwidth for data parallel communication. As visualized in Figure 1-(Top). This strategy creates four pipeline stages, each with 10 layers. However, even in this setting, Megatron has at least two deficiencies. First, two GPUs in machine *A* run pipeline parallelism on pipeline-1, which wastes the high NVLINK intra-machine bandwidth and introduces higher pipeline bubble overhead. Second, GPUs in machine *A* handle too few layers, wasting computation power.

When the underlying heterogeneous environment mismatches with the fully symmetric strategy of Megatron, the system treats GPUs with strong computation capability as weak GPUs, and parallel strategies are constrained by the network connections, leading to the underutilization of computation resources. In this case, the state-of-the-art training system Megatron performs poorly, with an estimated end-to-end training iteration time of 41.52s.

3.2 Asymmetric Parallel Support in HEXISCALE

To train efficiently under heterogeneous settings, we implement HEXISCALE with fully asymmetric parallel support and system optimizations, with the essential change as:

Asymmetric partition of the computation load. For each data-parallel replica[2], which can be assigned a *different* batch size, pipeline parallel communication groups are initialized with a *different* tensor model parallel degree and a corresponding, potentially *different* number of allocated transformer layers. Each pipeline stage selects a fixed leader GPU, which minimizes communication latency to GPUs in nearby stages and initializes a tensor model parallel group. During the forward pass, the leader GPU in each stage `sends` the activation to the leader GPU in the next stage. Once the leader GPU in the next stage `receives` the activation, it `broadcasts` this activation among its tensor parallel group to perform computations. During the backward pass, the same logic applies for communicating the gradients w.r.t activations. For cases where two adjacent pipeline stages have the same tensor model parallel degree, HEXISCALE falls back to the Megatron (Narayanan et al., 2021b) scatter/gather communication strategy for collective-communication optimization.

Asymmetric gradient synchronization. Model parameters (and corresponding gradients) of transformer layers across different pipelines may be chunked into different sizes due to differently assigned tensor parallel degrees. In this case, vanilla data parallelism cannot correctly synchronize gradients. To ensure communication correctness, we identify the smallest gradient size and further partition larger gradients into multiple chunks, each matching the size of the smallest identified gradient. Data parallel communication is then performed by synchronizing each gradient chunk among different subsets of data parallel workers, with the same communication volume.

System implementation. we optimize our system in four ways. First, we support gradient accumulation and activation recomputation to amortize data parallel communication cost and reduce memory footprint. Second, we leverage APIs in FlashAttention-2 (Dao, 2024)[3] to create transformer layers with both tensor model parallelism and FlashAttention. We then implement our

[2]For simplicity, in the remainder of the paper, we refer to each data-parallel replica as a pipeline.

[3]HEXISCALE also supports FlashAttention-3 (Shah et al., 2024) on Hopper GPUs.

asymmetric pipeline parallelism—compatible with popular pipeline schedules such as GPipe (Huang et al., 2019) and 1F1B (Narayanan et al., 2019)—using these transformer layers. Third, we support asymmetric gradient synchronization logic by registering custom FSDP communication hooks. Fourth, HEXISCALE incorporates a customized NCCL-based communication layer to support collective-communication across heterogeneous GPUs [4]. HEXISCALE initializes NCCL-communication-groups for data-, tensor model-, and pipeline-parallelism to configure collective-communication across heterogeneous GPUs. GPUs within one communication group can communicate.

3.3 Case Study: Boost with HEXISCALE

With the flexible design, HEXISCALE improves training efficiency in the former heterogeneous setting by creating two pipelines more efficiently as shown in Figure 1-(Bottom). The improvements are as follows:

Computation load is fully partitioned. For each pipeline, the computation load is fully partitioned as follows.

- From the perspective of parallel strategy, we fully utilize the strong computational power and high intra-machine bandwidth in machine *A* by applying tensor parallelism and assigning more transformer layers. Given the high intra-machine bandwidth in machine *A*, tensor model parallelism is found superior to pipeline parallelism. Machine *B* still applies pipeline parallelism as the intra-connection is not high enough for tensor model parallelism.
- From the perspective of layer partition, machine *B* and machine *C* undertake fewer transformer layers, which addresses the computation imbalance issue among pipeline stages. The number of layers that run inter-machine data parallelism is minimized. Megatron and HEXISCALE are both bounded by the slow data parallel communication on machine *B*. Megatron communicates for 10 transformer layers on machine *B*, which can be estimated at 9.90s. HEXISCALE only needs to communicate for 5 transformer layers in 5.07s, which is $1.9\times$ faster than Megatron.
- From the perspective of pipeline efficiency, batch sizes are assigned differently to balance the computation speeds among pipelines. Otherwise, there will be no end-to-end improvement, as HEXISCALE remain bounded by the pipeline-2. The pipeline-1 has to wait for pipeline-2 to run data parallel communication. To unleash the efficiency of pipeline-1, we assign larger batch sizes to pipeline-1. In this way, we balance the execution time of each pipeline and improve the end-to-end performance. In our case study, two pipelines process batch sizes with a 40% difference but exhibit a 7% difference in execution time.

[4]Note that only NVIDIA GPUs are supported in our implementation, which directly supports NCCL for different GPU types.

Run asymmetric data parallelism. For two GPUs in machine *A* on stage-0 of pipeline-1, and GPUs on stage-0 and stage-1 of pipeline-1, although tensor parallelism partitions the parameters to different sizes, data parallel communication among these stages is supported as discussed in §3.2. This flexible design enhanced the performance of pipeline-1 from two aspects. (i) Decreasing the number of pipeline stages reduces pipeline communication and bubble costs. (ii) Adopting tensor parallelism on machine *A* enhances the performance of pipeline-1.

In summary, the end-to-end training iteration time of HEXISCALE for the parallel strategy shown in Figure 1-(Bottom) is estimated to be 25.55s, making it $1.6\times$ faster than Megatron in this hypothetical heterogeneous setting.

4 SCHEDULING WITH HETEROGENEITY

Notations. We summarize the notations used in the scheduling algorithm in Table 1.

Table 1. Summarization of notations.

Symbol	Description
d	GPU device.
m_d	GPU memory of device d.
c_d	Tensor core computation power of device d.
$\alpha_{d,d'}$	Latency between devices d and d'.
$\beta_{d,d'}$	Bandwidth between devices d and d'.
σ	Parallel execution plan of devices in $\mathbf{D}$.
D_{dp}	Number of pipelines.
$\mathbf{D}$	Set of N GPU devices $d_1, d_2, ..., d_N$.
$\mathbf{E}$	Edge set of global graph.
$G = (\mathbf{D}, \mathbf{E})$	Global graph for GPU set $\mathbf{D}$.
$\mathbf{P}$	Global partition containing GPU set $\mathbf{D}_1, ..., \mathbf{D}_{D_{dp}}$.
k_i	Number of GPU groups generated in i-th pipeline.
$\mathbf{D}_i$	GPU set of the i-th pipeline.
$\mathbf{E}_i$	Edge set of the i-th pipeline.
$G_i = (\mathbf{D}_i, \mathbf{E}_i)$	Secondary graph of GPU set $\mathbf{D}_i$.
$\mathbf{P}_i$	Secondary partition containing GPU set $\mathbf{D}_{i,1}, ..., \mathbf{D}_{i,k_i}$.
$\mathbf{D}_{i,j}$	GPU set of the i-th pipeline, j-th GPU group.
τ	Parameter for searching pipeline stage order.

4.1 Formalization of the Scheduling Problem

Given a set of heterogeneous GPUs, we aim to identify the optimal heterogeneous parallel execution strategy that minimizes the training iteration time. We formalize the scheduling problem as follows. Let $\mathbf{D} = \{d_1 \dots d_N\}$ be a set of N GPU devices, and the GPU device memory limit be notated as m_d. Given a particular GPU set, the scheduling problem can be defined as identifying the optimal parallel execution plan σ^* that minimizes the training iteration execution time under the constraint of memory consumption:

$$\begin{aligned} \sigma^* = \arg\min_{\sigma} \quad & \textsc{Comm-Cost}(\sigma) + \textsc{Comp-Cost}(\sigma) \\ s.t. \quad & \textsc{Mem-Cumsum}(d) \le m_d \quad \forall d \in \mathbf{D} \end{aligned} \tag{1}$$

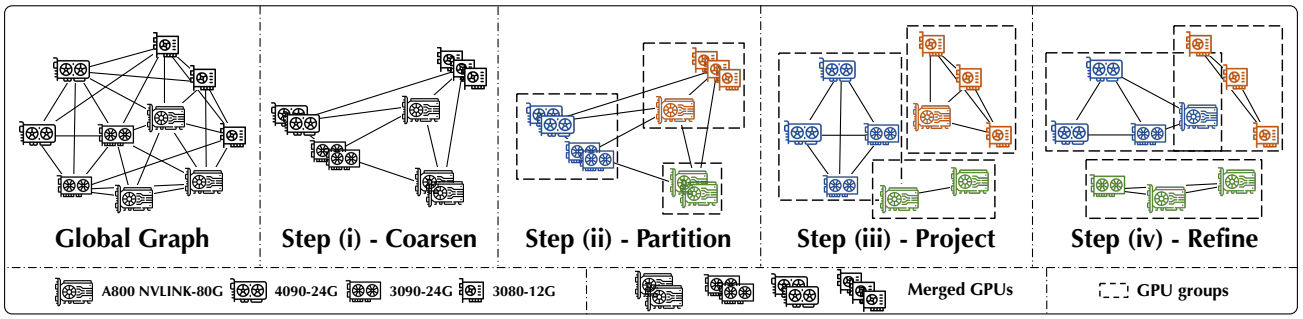

Figure 2. First phase: the global graph is partitioned into three groups of GPUs by four steps: (i)-coarsen, (ii)-partition, (iii)-project, and (iv)-refine. GPUs in the global graph are divided into three groups, which will be constructed as three pipelines.

where COMM-COST (σ) and COMP-COST (σ) represent communication and computation time costs (we present the detailed cost modeling in Appendix B). of one iteration given a parallel execution plan σ. MEM-CUMSUM (d) represents the memory consumption of device d.

One parallel execution plan σ can involve an arbitrary number of pipelines, each with varying global batch sizes, micro-batch sizes, and parallel strategies. Furthermore, a particular parallel strategy can have different configurations for data, pipeline, and tensor model parallel degrees. Each pipeline stage can contain a flexible number of transformer layers. Finding the exact optimal parallel strategy—considering the computation costs, communication costs, and memory consumption of all potential configurations—is NP-hard due to the exponential scale of candidate allocations.

Therefore, in a heterogeneous cluster with varying GPU capacities and network connectivity, it is often only possible to identify a near-optimal parallel execution plan using heuristic-based scheduling algorithms. Naively adopting the dynamic programming-based algorithms proposed in Alpa (Zheng et al., 2022) or Galvatron (Miao et al., 2022) leads to poor scheduling results or requires a significantly longer search time. To address the challenge in efficiently identifying a near-optimal parallel execution plan in heterogeneous clusters, we design a two-phase scheduling algorithm to find a parallel execution plan σ and iteratively optimize to a near-optimal solution σ^*. To be more concrete:

- We introduce the first phase algorithm in §4.2 that partitions the device set $\mathbf{D}$ into multiple GPU groups, each of which will be used to create one pipeline;
- We enumerate the second phase algorithm in §4.3 that identifies parallel execution plan σ for each pipeline;
- We iteratively repeat the two-phase algorithm and optimize the parallel execution plan to σ^*, illustrated in §4.4.

4.2 First Phase of Scheduling Algorithm

The key insight of the first phase algorithm is to partition the GPUs into multiple groups, with each group forming a separate pipeline. Data parallel communication is then performed among these pipelines to synchronize gradients.

Assume we divide devices set $\mathbf{D}$ into D_{dp} groups of GPUs in the current iteration, and *minimize* bandwidth for data parallel communication (we discuss parameter D_{dp} and bandwidth allocation in §4.4). We first organize all GPUs from devices set $\mathbf{D}$ as a global graph $G = (\mathbf{D}, \mathbf{E})$, where each GPU $d \in \mathbf{D}$ is a vertex, and computation power c_d is vertex weight; $\forall d_1, d_2 \in \mathbf{D}$, communication bandwidth β_{d_1,d_2} represent an edge between these two GPUs in set $\mathbf{E}$. Then we partition global graph G into global partition $\mathbf{P} = \{\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_{D_{dp}}\}$, where each set $\mathbf{D}_i \in \mathbf{P}$ contains GPUs with high bandwidth that are used in i-th pipeline , and $\mathbf{D}_i \cap \mathbf{D}_j = \emptyset, \forall i, j$. We adopt a partitioning method based on a D_{dp}-way multi-level graph partition algorithm (Hendrickson et al., 1995) consisting of four steps:

Step (i) - Coarsen. The global graph is coarsened into smaller graphs to simplify graph partition. Directly partitioning a large global graph (i.e., including many vertices) into D_{dp} parts is usually inefficient. In contrast, partitioning a smaller coarsened graph is more efficient. We adopt *heavy edge matching (HEM)* algorithm (Karypis & Kumar, 1998a) for coarsening. To allocate low bandwidth for data parallelism, this coarsening operation merges GPUs with *high* bandwidth. The bandwidth connecting GPUs within super-nodes is considered high when it is relatively higher than the bandwidth connecting GPUs from different super-nodes. As illustrated in Figure 2, step (i), the coarsened graph contains only half as many vertices as the global graph.

Step (ii) - Partition. The coarsened graph is partitioned into D_{dp} GPU groups, ensuring network bandwidth among GPU groups is minimized. With a recursive bisection method (Karypis & Kumar, 1998b), we recursively bisect the coarsened graph until D_{dp} parts partition is obtained.

Graph partition in this step solves a constrained partition problem that *minimizes* CUT objective function (Wei & Cheng, 1989) under two constraints: (i) maintaining strict balance, and (ii) partitioning to exactly D_{dp} parts. CUT objective function is defined by two levels: (i) CUT function between any two sets $\mathbf{D}_i, \mathbf{D}_j \in \mathbf{P}, \forall i, j$ is defined as the sum of edge weights (i.e., bandwidths) that connect them. (ii) CUT objective function for global partition $\mathbf{P}$ is defined as the summation of all cuts between sets $\mathbf{D}_i, \mathbf{D}_j \in \mathbf{P}, \forall i, j$. Formally, the two-level CUT function is defined as:

$$\begin{aligned} \text{CUT}(\mathbf{D}_i, \mathbf{D}_j) &= \sum_{d_i \in \mathbf{D}_i} \sum_{d_j \in \mathbf{D}_j} \beta_{d_i, d_j}, \quad \forall \mathbf{D}_i, \mathbf{D}_j \in \mathbf{P} \\ \text{CUT}(\mathbf{P}) &= \sum_{\mathbf{D}_i, \mathbf{D}_j \in \mathbf{P}} \text{CUT}(\mathbf{D}_i, \mathbf{D}_j) \end{aligned} \tag{2}$$

The constraint that measures the balance of global partition $\mathbf{P} = \{\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_{D_{dp}}\}$ is defined as the maximum sum of vertex weights $\max_{\mathbf{D}_i \in \mathbf{P}} \sum_{d \in \mathbf{D}_i} c_d$, over the average sum of vertex weights $\frac{\sum_{d \in \mathbf{D}} c_d}{D_{dp}}$. This balance factor is always greater than or equal to 1; its maximum value is a hyperparameter. A value closer to 1 indicates more evenly distributed total vertex weights among GPU sets $\mathbf{D}_i$.

Step (iii) - Project & Step (iv) - Refine. Partitioning coarsened graph is not the ultimate goal—as illustrated in Figure 2, step (iii), to find the partition of global graph G, we project the results back, i.e., apply reverse operation of step (i) to recover the D_{dp}-parts of the partition in global graph G. To effectively consider information within coarsened nodes, a refinement algorithm is necessary to enhance partition quality and maintain balance; for this purpose, we employ the Kernighan-Lin algorithm (Kernighan & Lin, 1970) in step (iv) to adjust partition results.

4.3 Second Phase of Scheduling Algorithm

The key insight of this phase is to efficiently generate the pipeline layout based on the graph partition results from the first phase. Thanks to the flexible asymmetric data parallelism design, we can independently determine a parallel strategy for each pipeline. However, the search space remains large, as we must determine the pipeline and tensor model parallelism strategy, as well as the execution order of the pipeline stages. In a fully heterogeneous environment, carefully permuting pipeline stages is necessary due to the heterogeneity of network connections. Formally, given the D_{dp} groups of GPUs, we use GPUs in each GPU set $\mathbf{D}_i \in \mathbf{P}$ to find a parallel execution plan σ_i for each pipeline. Finding the near-optimal layout for the assigned GPUs involves three key steps: (i) grouping GPUs for pipeline stages based on graph partition; (ii) constructing pipeline stages within each GPU group; and (iii) determining the order of pipeline stages under a heterogeneous network.

Step (i) - Group GPUs for pipeline stages. To group GPUs with high bandwidth connections and introduce algorithmic convenience to determine the stage order of the i-th pipeline (introduced shortly), we group GPUs with high bandwidth connections by further splitting GPUs in set $\mathbf{D}_i \in \mathbf{P}$ into multiple groups. Concretely, we first organize each set $\mathbf{D}_i \in \mathbf{P}$ into a secondary graph $G_i = (\mathbf{D}_i, \mathbf{E}_i), i = 1, ..., D_{dp}$, where the edge set $\mathbf{E}_i$ contains communication bandwidths connecting GPUs in set $\mathbf{D}_i$. Next, we partition each secondary graph G_i into secondary partition $\mathbf{P}_i = \{\mathbf{D}_{i,1}, ..., \mathbf{D}_{i,k_i}\}$, where each parameter k_i controls the number of parts that G_i is partitioned into. Set $\mathbf{D}_{i,k} \in \mathbf{P}_i$, contains GPUs to construct pipeline stages, and $\mathbf{D}_{i,k_1} \cap \mathbf{D}_{i,k_2} = \emptyset, \forall k_1, k_2$. The secondary graphs are partitioned using the same multi-level graph partition method as discussed in §4.2. As an illustrative example shown in Figure 3-(Top), GPUs in a secondary graph are first partitioned into three groups after this step. Note that GPUs within the same GPU sets $\mathbf{D}_{i,k} \in \mathbf{P}_i$ have high bandwidth connections. Finding the stage order among these stages has minor effects since pipeline communication costs are not the bottleneck. In contrast, when pipeline stages are generated by different GPU sets $\mathbf{D}_{i,k} \in \mathbf{P}_i$, permuting these stages can significantly reduce the pipeline communication overhead by effectively utilizing the low bandwidth.

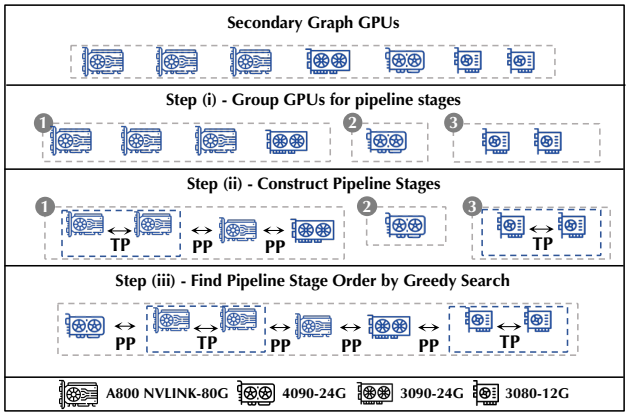

Figure 3. Second phase: each pipeline is created in three steps. (i) GPUs with high bandwidth connections are grouped by graph partition. (ii) *intra-group strategy* is searched separately for each machine, i.e., GPUs in the same machine. (iii) Pipeline stage order is determined by permuting all intra-group strategies by a top-τ greedy search algorithm.

In the following two steps, we first construct pipeline stages within each GPU set $\mathbf{D}_{i,k}$ without considering pipeline stage order. Then, we search the pipeline stage order among pipeline stages created by different GPU sets.

Step (ii) - Construct pipeline stages. Given the secondary graph partition results of each secondary graph G_i, we construct the i-th pipeline as follows. Each GPU set $\mathbf{D}_{i,k} \in \mathbf{P}_i$ separately searches its *intra-group strategy* within each ma-

chine, through simulating different parallelism strategies with cost models defined in the Appendix B and selecting the locally optimal parallel strategy. As shown in Figure 3-(Middle), the first GPU group constructs three stages, while other GPU groups construct one stage in each.

Step (iii) - Find pipeline stage order by greedy search. We consider each *intra-group strategy* as a single vertex, and construct a new graph G'_i for the i-th pipeline. The stage order for the i-th pipeline is thereby searched by a top-τ greedy algorithm. The algorithm runs in two nested loops. First, it selects each GPU group as the starting group. Second, for each neighboring GPU group with τ-highest inter-group bandwidth, the algorithm recursively explores their neighboring GPU groups with τ-highest inter-group bandwidth until a pipeline path is generated. As shown in Figure 3-(Bottom), stage order within each *intra-group strategy* is not changed (as we do not permute them), while three pipeline stages in the first *intra-group strategy* are placed after the one pipeline stage in the second *intra-group strategy*, and before the one pipeline stage in the last *intra-group strategy*.

4.4 Iterative Optimization

Lastly, we introduce the iterative optimization procedure — The parallel execution plan σ is iteratively optimized to the final near-optima σ^* from two aspects:

Optimize from first phase algorithm: The first phase algorithm can be optimized from two aspects.

First, the algorithm partitions the global graph into different numbers of pipelines by enumerating the parameter D_{dp} across iterations to optimize the number of pipelines.

Second, the algorithm carefully allocates network bandwidth for data and pipeline parallelism communication. One of our key observations is that both data parallel communication and pipeline execution time can be the bottleneck in heterogeneous environments. When pipelines have many pipeline stages and handle a large batch size, pipeline execution time accounts for most of the training iteration time. In this case, minimizing bandwidth for data parallelism and maximizing bandwidth for pipeline execution can effectively improve the system performance. The system performance benefits from the reduced communication overhead of pipeline execution. On the other hand, when pipelines have few pipeline stages and handle a small batch size, system performance benefits from maximizing bandwidth for data parallelism and minimizing bandwidth for pipeline execution. The system performance is boosted by the reduced data parallel communication overhead. Based on this observation, we implement two partition options: either (i) **maximize** or (ii) **minimize** the inter-group (i.e., GPU groups for each pipeline) edge weights (i.e., bandwidth). Maximizing the inter-group edge weights corresponds to maximizing the CUT objective function in Equation 2, which in turn results in allocating high communication bandwidth for data parallelism; conversely, minimizing the edge weights results in allocating low communication bandwidth for data parallelism. At each iteration, we adaptively select the potentially optimal partition option based on the historical moving average of costs to optimize system performance. Specifically, we simulate data parallel communication and pipeline execution costs at the end of each iteration and update the historical average costs. In the next iteration, the partition decision is made using this historical information. With this design, our first-phase algorithm effectively allocates network bandwidth, enhancing system performance.

Optimize from second phase algorithm: Given the global graph partition result in the first phase, varying the parameters k_i across iterations results in a distinct intra-group strategy, which, in turn, determines the configuration of pipeline stages and their order. By fine-tuning k_i, we can construct pipelines that achieve high efficiency.

Batch and layer assignment: For a given parallel execution plan generated by our two-phase algorithm, our algorithm automates batch and layer distribution using *greedy strategies*. For batch assignment, we distribute batches proportionally to the estimated training efficiency of each pipeline. For layer assignment, we prioritize assigning layers to pipeline stages in proportion to their estimated computational efficiency. If out-of-memory issues are detected on certain stages, we reduce the number of layers assigned to those stages and reassign the corresponding layers to stages with the most abundant memory resources.

Evaluate by simulation: at the end of each iteration, we simulate the execution costs for the generated parallel execution plan using our cost model, given the communication bandwidth matrix and hardware computation capabilities. The communication bandwidth matrix serves as an abstraction of the entire cloud network, and we obtain this matrix through profiling communication status between every GPU-pair; for computation capabilities, we represent them by theoretical peak FLOPS of each GPU type. When a generated parallel execution plan results in an out-of-memory error, its execution cost is set to infinity. If the plan results in poor performance due to imbalanced computation, inefficient memory usage, or suboptimal network utilization, the cost is assigned a high penalty value. Conversely, if the plan efficiently utilizes the given heterogeneous cluster, a low cost is assigned. Finally, the parallel strategy with the lowest estimated cost is selected. Our cost model closely predicts actual execution costs. We provide a detailed evaluation on the simulation accuracy in §5.4 and Appendix C, results show that simulation deviations are less than 2.9%. To enhance simulation accuracy, we further incorporate net-

work latency ($\alpha_{d,d'}$) into the simulation. As the number of collective communication operations increases, system performance degrades due to high linking costs. Network latency is particularly critical in heterogeneous environments, where a large number of micro-batches and long pipeline schedules lead to an increased number of NCCL operations.

5 EVALUATION

5.1 Experimental Setups

Experimental setup. To thoroughly compare the end-to-end performance of HEXISCALE and state-of-the-art frameworks, we include Megatron, Galvatron, and FSDP as baseline frameworks, and LLAMA models in different scales as representative models[5]. We evaluate HEXISCALE based on the following experiment settings:

Homogeneous settings. We rent two `8×A800 PCIe-80G` with or without RDMA, to test the maximum throughput (measured by the achieved PFLOPS) on LLAMA-2 (7B), LLAMA-2 (13B), and four `8×A800 PCIe-80G` to test the maximum throughput on LLAMA (30B).

Heterogeneous settings. We rent GPUs from cloud provider `Ucloud`. To evaluate system performance in various heterogeneous environments, we test and compare the performance of Megatron, Galvatron, and HEXISCALE under three settings (inter-connections at 5.6 Gbps) as follows:

- *Heterogeneous setting 1*: one `8×3080Ti`, one `8×3090`, and three `8×4090`. With 1.36% higher total FLOPS and 1.48× smaller total memory, we compare the maximum throughput gap with two `8×A800 PCIe-80G` in training LLAMA-2 (7B)/(13B). `3080Ti` has 12 GB memory, `3090` and `4090` each have 24 GB memory. Intra-machine connections of `3080Ti` and `3090` are 24 GB/s.
- *Heterogeneous setting 2*: one `8×3080Ti`, one `8×3090`, one `8×4090`, and one `8×A800 NVLINK-80G` (200 GB/s NVLINK). With 1.59% less total theoretical FLOPS and 1.14× smaller total memory, we compare the maximum throughput gap with two `8×A800 PCIe-80G` in training LLAMA-2 (7B)/(13B).
- *Heterogeneous setting 3*: one `8×3090`, two `4×3090`, four `8×4090`, and one `8×A800 NVLINK-80G`. With 4.67% less total theoretical FLOPS and 1.43× smaller memory, we compare the maximum throughput gap with four `8×A800 PCIe-80G` in training LLAMA (30B).

5.2 End-to-end Performance

Results and discussions. *First,* HEXISCALE *exhibits performance comparable to other high-performance systems in homogeneous settings.* As shown in Figure 4 and Figure 5 [6], HEXISCALE achieves comparable performance as the baseline frameworks Megatron and Galvatron under both RDMA and Ethernet inter-machine connections. FSDP is unsuitable for clusters with low-speed inter-connections; even with RDMA bandwidths of 10 GB/s, using FSDP alone exhibits poor performance.

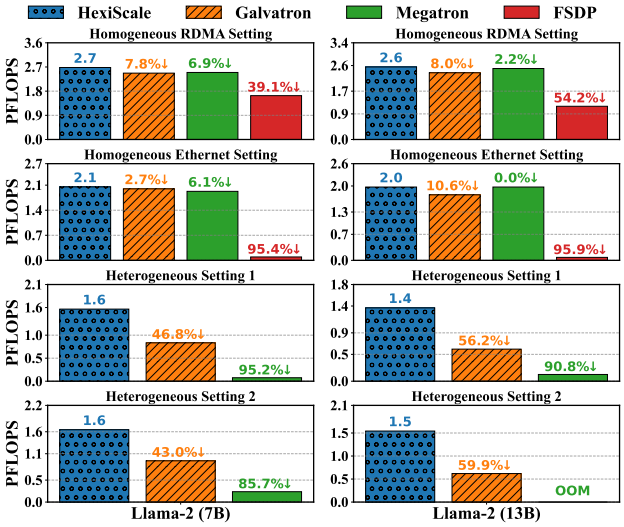

Figure 4. End-to-end experiments of HEXISCALE compared with other systems under various settings on LLAMA-2 (7B / 13B).

Second, comparisons across frameworks in heterogeneous settings highlight the adaptability of HEXISCALE. While Megatron and Galvatron perform well in homogeneous environments, they cannot be easily adapted to heterogeneous settings. As shown in Figure 5, Megatron can not run in heterogeneous setting 3 when training LLAMA (30B). Despite tuning various parallel strategies, out-of-memory issues persist. Megatron enforces a fully symmetric parallel strategy, leading to significant imbalance issues. Different types of GPUs cannot fully utilize computational capabilities and intra-machine bandwidth when restricted to a uniform parallel strategy. For example, GPUs with high intra-machine bandwidth must adopt a low tensor model parallel degree to accommodate GPUs with lower bandwidth, resulting in underutilized resources. Galvatron offers greater flexibility than Megatron by allowing flexible transformer layer assignment. However, parallel strategies remain suboptimal for heterogeneous clusters, leading to performance degradation. Compared to Galvatron, HEXISCALE achieves up to a 2.5× higher throughput and, on average, a 2.1× improvement.

Finally, comparisons between homogeneous settings and heterogeneous settings demonstrate the strong competitiveness of HEXISCALE. In homogeneous settings with RDMA,

[5] LLM often differ on model scales, instead of model structure.

[6] We present the detailed parallel execution configurations in Appendix G for each system.

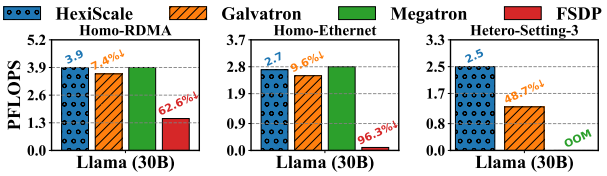

Figure 5. End-to-end experiments of HEXISCALE compared with other systems under various settings on LLAMA (30B) model.

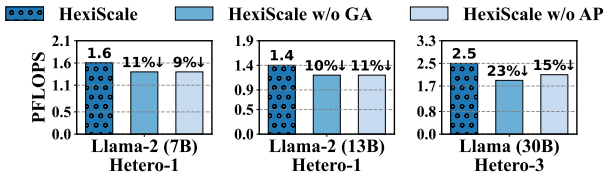

Figure 6. Breakdown experiments of HEXISCALE with LLAMA-2 (7B / 13B / 30B) models under hetero-setting 1 and 3.

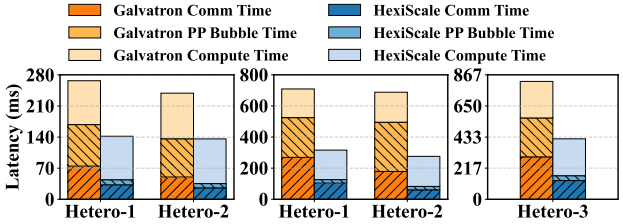

Figure 7. Breakdown of end-to-end time across different heterogeneous experimental settings and models. We benchmark the per-batch communication time, computation time, and pipeline bubble time for HEXISCALE and Galvatron.

the achieved maximum throughput of Megatron and Galvatron is significantly higher than when training with HEXISCALE in heterogeneous environments. This performance gap arises from inter-machine connections: Megatron and Galvatron utilize high-performance RDMA, whereas HEXISCALE runs over slower Ethernet-based inter-connections. The performance of HEXISCALE can be further improved by replacing Ethernet with RDMA connections (This point is validated through simulation, as presented in §5.5). To fairly evaluate heterogeneous training performance, we conduct baseline experiments using Megatron, Galvatron, and FSDP in homogeneous settings with Ethernet inter-machine connections. As shown in Figure 4 and Figure 5, despite the challenges posed by imbalanced GPU computation, limited total memory capacity, and more frequent inter-machine communication in heterogeneous clusters, HEXISCALE exhibits up to $1.01\times$ throughput and an average throughput gap of $0.83\times$ compared to homogeneous scenarios [7]. By asymmetric system design and effective scheduling, HEXISCALE better utilizes fragmented GPU resources and exhibits strong potential in diverse heterogeneous settings.

5.3 Ablation Studies

System design breakdown. HEXISCALE is implemented with asymmetric pipeline parallelism, under the support of asymmetric data parallelism, and other system optimizations, including gradient accumulation (GA). We evaluate the effectiveness of each design separately, the results are shown in Figure 6. We analyze the ablation results as follows:

Disabling our asymmetric parallel support, system performance at most degrades 23% (in training LLAMA (30B)), and 15% on average. Without our asymmetric parallel support, all pipeline stages must have the same tensor model parallelism degrees, which is often suboptimal due to the underutilization of distinct hardware features. For example, in heterogeneous setting 3, a higher tensor model parallelism degree on $8\times$A800 is beneficial. Conversely, GPUs with lower intra-machine bandwidths should be assigned lower tensor model parallelism degrees (detailed in §5.4).

Disabling gradient accumulation, system performance at most degrades 15% (for LLAMA (30B)), and 12% on average. Heterogeneous GPUs with limited memory can only accommodate a small batch size. As a result, the execution time between two data parallel gradient synchronizations is short. Without gradient accumulation, frequent data parallel communication degrades system performance.

Training iteration latency breakdown. We present breakdown of training iteration latency of HEXISCALE and Galvatron under various heterogeneous settings in Figure 7. HEXISCALE outperforms Galvatron by reducing communication overhead and pipeline bubbles. With a symmetric parallel strategy, Galvatron runs a high degree of pipeline parallelism, which increases both communication overhead and pipeline bubbles. Furthermore, Galvatron experiences additional bubbles due to imbalanced computation across pipeline stages. With asymmetric parallel support, HEXISCALE addresses these challenges by reducing the number of pipeline stages and carefully balancing computation.

5.4 Scheduling Algorithm Evaluation

Another major concern is the performance of our algorithm, including the scheduling results, simulation accuracy, effectiveness of graph partitioning, and running efficiency.

Evaluate the effectiveness of graph partition. We evaluate our graph partition algorithm by comparing the algorithm convergence gap when running the carefully designed graph partition algorithm and a random graph partition in multiple rounds. As shown in Figure 8, our algorithm outperforms

[7] The throughput gaps are calculated by dividing the throughput of HEXISCALE on heterogeneous clusters by the throughput of Megatron and Galvatron on homogeneous clusters interconnected via Ethernet.

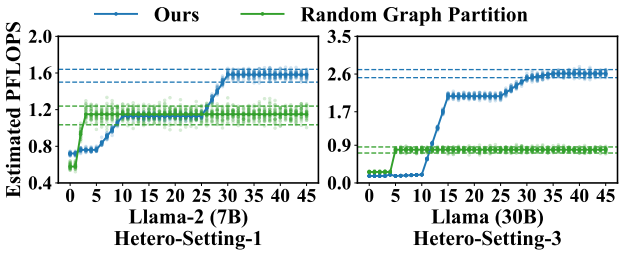

Figure 8. Convergence comparison of our algorithm and random graph partition with LLAMA-2 (7B) (left) and (30B) (right) models, where both run 20 times.

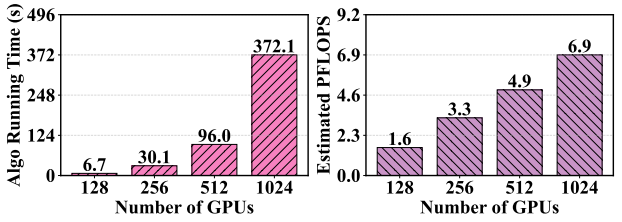

Figure 9. Algorithm running time and estimated throughput for GPU cluster sizes (ranging from 4× to 32× GPUs compared to Hetero-Setting 2)

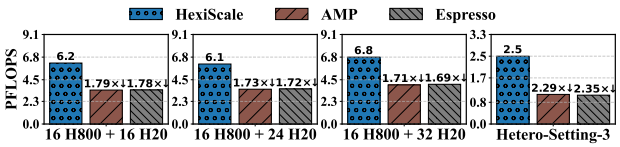

Figure 10. Achieved PFLOPS of HEXISCALE/AMP/Espresso.

random graph partition when searching parallel execution plan for LLAMA-2 (7B) model in heterogeneous setting 1, and LLAMA (30B) model in heterogeneous setting 3. Despite fluctuations across different rounds, our algorithm converges to 1.4× and 3.3× higher estimated throughput, for LLAMA-2 (7B) and LLAMA (30B). Furthermore, our algorithm continues to improve over iterations, whereas the random graph partitioning method typically converges early without significant improvement. The performance edge of our algorithm becomes more pronounced as the cluster complexity increases and the model size grows. One contributing factor is that random graph partitioning is more likely to result in out-of-memory errors.

Evaluate the effectiveness of batch and layer assignment. Under heterogeneous setting 1 with the LLAMA (13B) model, we conduct additional ablation studies comparing our greedy strategy against uniform batch and layer distribution. Disabling the greedy layer distribution leads to an 11.46% degradation in throughput (from 1.57 PFLOPS to 1.39 PFLOPS), while disabling the greedy batch distribution results in a 9.55% performance degradation (from 1.57 PFLOPS to 1.42 PFLOPS).

Evaluate the algorithm scalability. Our graph partition-based scheduling algorithm efficiently handles large graphs, thanks to the introduced heuristics, which prune unnecessary parts of the search space. For example, our coarsening operation (Step (i) in §4.2) merges graph vertices based on network bandwidth, effectively reducing the complexity of the partitioning process while preserving partition quality. To evaluate the speed of our algorithm, we run 50 iterations (where our algorithm generally converges) on 12-core CPUs to search for the optimal parallel execution plan of different numbers of GPUs (128 to 1024). As shown in Figure 9, our algorithm scales well with an increasing number of GPUs, maintaining a runtime of less than seven minutes even with 1024 GPUs. The scheduling overhead remains manageable, compared to the months-long training time required for large language models. Additionally, the simulated throughput exhibits stable scaling on larger clusters, increasing from 1.6 PFLOPS with 128 GPUs to 6.9 PFLOPS with 1024 GPUs, demonstrating the robustness of our algorithm.

5.5 Case Studies

To further evaluate HEXISCALE, we present case studies in this section, comparing HEXISCALE with state-of-the-art heterogeneity-aware training systems. We present detailed discussions as follows, and present the detailed experimental configurations for each system in Appendix G.

Compare with AMP/Espresso. To compare HEXISCALE with AMP/Espresso, we conduct experiments for LLAMA (30B) under heterogeneous settings with both low and high inter-machine network bandwidth. The low-bandwidth scenario is enabled by heterogeneous setting 3; the high-bandwidth scenarios are enabled by three heterogeneous settings with InfiniBand. Compared to AMP/Espresso, results in Figure 10 demonstrate that: (i) with low inter-connections, HEXISCALE achieves 2.29×/2.35× higher throughput; (ii) with high inter-connections, HEXISCALE achieves on average 1.75×/1.73× higher throughput. In both favorable and unfavorable network conditions, HEXISCALE consistently delivers superior performance through flexible asymmetric parallel support and effective scheduling.

Compare with Metis. To fairly compare HEXISCALE with Metis, one of the state-of-the-art heterogeneous training systems (Um et al., 2024), we conduct experiments for LLAMA (30B) on heterogeneous settings with both low (enabled by heterogeneous setting 3) and high (enabled by 16 H800 and 32 H20 with InfiniBand) inter-machine network bandwidth. Since Metis does not open-source its training system and Metis's strategy is deployable via HEXISCALE, we run its open-source scheduling algorithm, and deploy the searched parallelism strategy on HEXISCALE. Experimental results in Figure 11 (left, middle) demonstrate that Metis

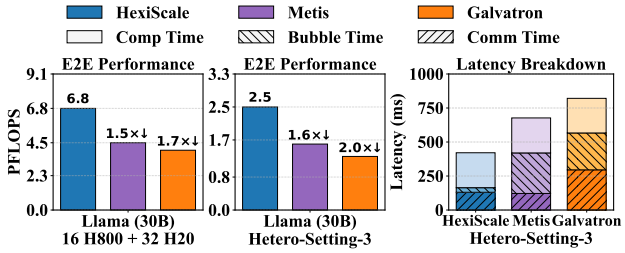

Figure 11. End-to-end performance comparison (left, middle), and per-batch latency breakdown (right) of three systems.

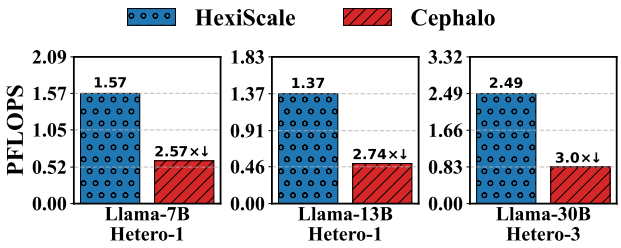

Figure 12. Achieved PFLOPS of HEXISCALE and Cephalo.

achieves a throughput of 4.53 and 1.56 PFLOPS on the given two settings, which is 1.5× and 1.6× lower than that of HEXISCALE. This provides strong evidence supporting the design of HEXISCALE: the limitation of Metis's algorithmic design (see details in §2.2) overlooks opportunities where tensor model parallelism can further enhance system performance. Although tensor model parallelism is more communication-intensive than pipeline parallelism, it incurs no bubble overhead, which can be a potential bottleneck in heterogeneous environments. (i) For heterogeneous settings 3, Metis recommends using data parallelism within each machine and pipeline parallelism across the 8 machines (detailed strategy is listed in Appendix G). (ii) For 16 H800 and 32 H20, Metis suggests using data parallelism within each machine and pipeline parallelism across the 6 machines. In both cases, the long pipeline configuration suggested by Metis incurs significant pipeline bubble overhead. In contrast, HEXISCALE demonstrates superior scheduling across different forms of parallelism. Compared to Metis, the key difference lies in the tensor and data parallelism search space: HEXISCALE explores $O(D^2)$ combinations, whereas Metis considers only $O(D)$, where D represents the number of GPUs assigned to pipeline stages. For example, tensor model parallelism is additionally incorporated in the strategy suggested by HEXISCALE (detailed in Table 7) to boost system performance.

Compare with Cephalo. Cephalo's design includes support for uneven assignment of batches and training states (i.e., parameters, gradients, and optimizer states), and optimized gradient accumulation that leverages activation offloading. In contrast, HEXISCALE enables fully asymmetric parallel support, i.e., asymmetric pipeline- and data-parallelism. Compared to Cephalo, HEXISCALE accommodates stringent network conditions, e.g., when low-speed Ethernet is used for inter-machine communication, and delivers superior performance. Since Cephalo is closed-source, we conduct additional comparisons between HEXISCALE and Cephalo through simulations implemented by us over our heterogeneous settings 1 and heterogeneous setting 3 (5.6Gbps inter-machine bandwidth). Experimental results presented in Figure 12 show that HEXISCALE achieves up to 3.0× higher PFLOPS and, on average, 2.8× higher PFLOPS than Cephalo. Note that under practical low inter-machine bandwidth, the data-parallel communication overhead in Cephalo significantly degrades performance. In contrast, HEXISCALE incurs moderate communication overhead due to its hybrid and asymmetric parallelism design.

Latency breakdown comparison. For heterogeneous setting 3, we present a detailed breakdown of training iteration latency in Figure 11 (right). Compared to Metis, HEXISCALE incurs slightly higher communication overhead, primarily due to its increased demand for tensor model and data parallel communication. However, HEXISCALE significantly reduces pipeline bubble time. Metis's design necessitates a high degree of pipeline parallelism, inevitably increasing bubble time. In contrast, HEXISCALE efficiently mitigates this problem by incorporating tensor model parallelism and reducing the number of pipeline stages, thus achieving superior performance.

More experimental results. We compare our scheduling algorithm with an MILP-based scheduling approach in Appendix E. Experimental results show that our algorithm achieves comparable performance to the MILP approach.

6 CONCLUSION

HEXISCALE is a novel system that enhances LLM training on heterogeneous GPUs by asymmetric parallelism and effective scheduling. Empirical studies show that when running over heterogeneous GPUs, HEXISCALE achieves comparable throughput to state-of-the-art homogeneous training systems, and outperforms state-of-the-art heterogeneous training systems with the same heterogeneous GPUs.

ACKNOWLEDGMENT

This work is supported by the HKUST startup grant R9895 from CSE; RGC-ECS project 26218024; RGC-NSFC project CRS_HKUST601/24; Webank-HKUST joint lab project (WEB24EG01-E-CSE. We thank the gift credit from Ucloud that supports this project. We also thank the support from the National Supercomputer Center in Guangzhou Nansha Sub-Center.

REFERENCES

Ai, X., Wang, Q., Cao, C., Zhang, Y., Chen, C., Yuan, H., Gu, Y., and Yu, G. Neutronorch: Rethinking sample-based gnn training under cpu-gpu heterogeneous environments. *Proceedings of the VLDB Endowment*, 17(8): 1995–2008, 2024.

Amazon. Amazon ec2 instance types, 2024. URL https://aws.amazon.com/ec2/instance-types/.

Amazon Web Services. Aws parallelcluster documentation, 2024. https://docs.aws.amazon.com/parallelcluster/.

Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024. URL https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Cai, Z., Yan, X., Ma, K., Wu, Y., Huang, Y., Cheng, J., Su, T., and Yu, F. Tensoropt: Exploring the tradeoffs in distributed dnn training with auto-parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 33(8): 1967–1981, 2021.

Chen, J., Li, S., Ran, G., Yuan, J., and Hoefler, T. Autoddl: Automatic distributed deep learning with asymptotically optimal communication. *memory*, 1:2, 2023.

Chen, P., Zhang, W., He, S., Gu, Y., Peng, Z., Huang, K., Zhan, X., Chen, W., Zheng, Y., Wang, Z., et al. Optimizing large model training through overlapped activation recomputation. *arXiv preprint arXiv:2406.08756*, 2024.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Fan, S., Rong, Y., Meng, C., Cao, Z., Wang, S., Zheng, Z., Wu, C., Long, G., Yang, J., Xia, L., et al. Dapple: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 431–445, 2021.

Hendrickson, B., Leland, R. W., et al. A multi-level algorithm for partitioning graphs. *SC*, 95(28):1–14, 1995.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Institute, T. I. Falcon 180b, 2023. URL https://falconllm.tii.ae/falcon-180b.html.

Jia, X., Jiang, L., Wang, A., Xiao, W., Shi, Z., Zhang, J., Li, X., Chen, L., Li, Y., Zheng, Z., et al. Whale: Efficient giant model training over heterogeneous {GPUs}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pp. 673–688, 2022.

Jia, Z., Lin, S., Qi, C. R., and Aiken, A. Exploring hidden dimensions in parallelizing convolutional neural networks. In *ICML*, volume 2279, pp. 2288, 2018.

Jia, Z., Zaharia, M., and Aiken, A. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024a.

Jiang, Y., Fu, F., Miao, X., Nie, X., and Cui, B. Osdp: Optimal sharded data parallel for distributed deep learning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 2142–2150, 2023.

Jiang, Y., Yan, R., Yao, X., Zhou, Y., Chen, B., and Yuan, B. Hexgen: Generative inference of large language model over heterogeneous environment. In *Forty-first International Conference on Machine Learning*, 2024b.

Jiang, Y., Fu, F., Yao, X., He, G., Miao, X., Klimovic, A., Cui, B., Yuan, B., and Yoneki, E. Demystifying cost-efficiency in llm serving over heterogeneous gpus. *arXiv preprint arXiv:2502.00722*, 2025a.

Jiang, Y., Fu, F., Yao, X., Wang, T., Cui, B., Klimovic, A., and Yoneki, E. Thunderserve: High-performance and cost-efficient llm serving in cloud environments. *arXiv preprint arXiv:2502.09334*, 2025b.

Jiang, Y., Yan, R., and Yuan, B. Hexgen-2: Disaggregated generative inference of llms in heterogeneous environment. *arXiv preprint arXiv:2502.07903*, 2025c.

Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998a.

Karypis, G. and Kumar, V. Multilevel algorithms for multi-constraint graph partitioning. In *SC'98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, pp. 28–28. IEEE, 1998b.

Kernighan, B. W. and Lin, S. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

Li, D., Wang, H., Xing, E., and Zhang, H. Amp: Automatically finding model parallel strategies with heterogeneity awareness. *Advances in Neural Information Processing Systems*, 35:6630–6639, 2022.

Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., et al. Pytorch distributed: experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13 (12):3005–3018, 2020.

Li, Z., Jian, X., Wang, Y., Shao, Y., and Chen, L. Daha: Accelerating gnn training with data and hardware aware execution planning. *Proceedings of the VLDB Endowment*, 17(6):1364–1376, 2024.

Mei, Y., Zhuang, Y., Miao, X., Yang, J., Jia, Z., and Vinayak, R. Helix: Distributed serving of large language models via max-flow on heterogeneous gpus. *arXiv preprint arXiv:2406.01566*, 2024.

Miao, X., Nie, X., Shao, Y., Yang, Z., Jiang, J., Ma, L., and Cui, B. Heterogeneity-aware distributed machine learning training via partial reduce. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 2262–2270, 2021.

Miao, X., Wang, Y., Jiang, Y., Shi, C., Nie, X., Zhang, H., and Cui, B. Galvatron: Efficient transformer training over multiple gpus using automatic parallelism. *Proceedings of the VLDB Endowment*, 16(3):470–479, 2022.

Miao, X., Shi, Y., Yang, Z., Cui, B., and Jia, Z. Sdpipe: A semi-decentralized framework for heterogeneity-aware pipeline-parallel training. *Proceedings of the VLDB Endowment*, 16(9):2354–2363, 2023.

Nagrecha, K. Model-parallel model selection for deep learning systems. In *Proceedings of the 2021 international conference on management of data*, pp. 2929–2931, 2021.

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 1–15, 2019.

Narayanan, D., Phanishayee, A., Shi, K., Chen, X., and Zaharia, M. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*, pp. 7937–7947. PMLR, 2021a.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021b.

Nvidia. Gpu computing solutions for hpc, 2006. URL https://www.nvidia.com/docs/IO/43395/tesla_product_overview_dec.pdf.

Nvidia. Nvidia reinvents computer graphics with turing architecture, 2018. URL https://nvidianews.nvidia.com/news/nvidia-reinvents-computer-graphics-with-turing-architecture.

Nvidia. Nvidia's new ampere data center gpu in full production, 2020. URL https://nvidianews.nvidia.com/news/nvidias-new-ampere-data-center-gpu-in-full-production.

Nvidia. Nvidia announces hopper architecture, the next generation of accelerated computing, 2022. URL https://nvidianews.nvidia.com/news/nvidia-announces-hopper-architecture-the-next-generation-of-accelerated-computing.

Nvidia. Nvidia blackwell platform arrives to power a new era of computing, 2024. URL https://nvidianews.nvidia.com/news/nvidia-blackwell-platform-arrives-to-power-a-new-era-of-computing.

OpenAI. Openai gpt-4o, 2024. URL https://platform.openai.com/docs/models/gpt-4o.

Park, J. B., Mailthody, V. S., Qureshi, Z., and Hwu, W.-m. Accelerating sampling and aggregation operations in gnn frameworks with gpu initiated direct storage accesses. *Proceedings of the VLDB Endowment*, 17(6):1227–1240, 2024.

Qi, P., Wan, X., Huang, G., and Lin, M. Zero bubble (almost) pipeline parallelism. In *The Twelfth International Conference on Learning Representations*, 2024.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Rajbhandari, S., Ruwase, O., Rasley, J., Smith, S., and He, Y. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–14, 2021.

Reid, M., Savinov, N., Teplyashin, D., Lepikhin, D., Lillicrap, T., Alayrac, J.-b., Soricut, R., Lazaridou, A., Firat, O., Schrittwieser, J., et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564, 2021.

Santhanam, K., Krishna, S., Tomioka, R., Fitzgibbon, A., and Harris, T. Distir: An intermediate representation for optimizing distributed neural networks. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pp. 15–23, 2021.

Shah, J., Bikshandi, G., Zhang, Y., Thakkar, V., Ramani, P., and Dao, T. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08608*, 2024.

Tarnawski, J. M., Narayanan, D., and Phanishayee, A. Piper: Multidimensional planner for dnn parallelization. *Advances in Neural Information Processing Systems*, 34: 24829–24840, 2021.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Um, T., Oh, B., Kang, M., Lee, W.-Y., Kim, G., Kim, D., Kim, Y., Muzzammil, M., and Jeon, M. Metis: Fast automatic distributed training on heterogeneous {GPUs}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 563–578, 2024.

Unger, C., Jia, Z., Wu, W., Lin, S., Baines, M., Narvaez, C. E. Q., Ramakrishnaiah, V., Prajapati, N., McCormick, P., Mohd-Yusof, J., et al. Unity: Accelerating {DNN} training through joint optimization of algebraic transformations and parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 267–284, 2022.

Wang, Y., Jiang, Y., Miao, X., Fu, F., Zhu, S., Nie, X., Tu, Y., and Cui, B. Improving automatic parallel training via balanced memory workload optimization. *IEEE Transactions on Knowledge and Data Engineering*, 2024a.

Wang, Y., Wang, S., Zhu, S., Fu, F., Liu, X., Xiao, X., Li, H., Li, J., Wu, F., and Cui, B. Data-centric and heterogeneity-adaptive sequence parallelism for efficient llm training. *arXiv preprint arXiv:2412.01523*, 2024b.

Wei, Y.-C. and Cheng, C.-K. Towards efficient hierarchical designs by ratio cut partitioning. In *1989 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, pp. 298–301. IEEE, 1989.

Wu, K., Park, J. B., Zhang, X., Hidayetoğlu, M., Mailthody, V. S., Huang, S., Lumetta, S. S., and Hwu, W.-m. Tba: Faster large language model training using ssd-based activation offloading. *arXiv preprint arXiv:2408.10013*, 2024.

Yang, B., Zhang, J., Li, J., Ré, C., Aberger, C., and De Sa, C. Pipemare: Asynchronous pipeline parallel dnn training. *Proceedings of Machine Learning and Systems*, 3:269–296, 2021.

Yi, X., Zhang, S., Luo, Z., Long, G., Diao, L., Wu, C., Zheng, Z., Yang, J., and Lin, W. Optimizing distributed training deployment in heterogeneous gpu clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pp. 93–107, 2020.

Young, A., Chen, B., Li, C., Huang, C., Zhang, G., Zhang, G., Li, H., Zhu, J., Chen, J., Chang, J., et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024.

Yuan, B., He, Y., Davis, J., Zhang, T., Dao, T., Chen, B., Liang, P. S., Re, C., and Zhang, C. Decentralized training of foundation models in heterogeneous environments. *Advances in Neural Information Processing Systems*, 35: 25464–25477, 2022.

Zhang, Z., Zheng, S., Wang, Y., Chiu, J., Karypis, G., Chilimbi, T., Li, M., and Jin, X. Mics: near-linear scaling for training gigantic model on public cloud. *Proceedings of the VLDB Endowment*, 16(1):37–50, 2022.

Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C.-C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., et al. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12): 3848–3860, 2023.

Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Xing, E. P., 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)*, pp. 559–578, 2022.

Zhou, Q., Xu, F., Weng, L., Li, R., Wu, X., Chen, L., Zhou, Z., and Liu, F. Espresso: Cost-efficient large model training by exploiting gpu heterogeneity in the cloud. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*, pp. 1–10. IEEE, 2025.

Zhuang, Y., Zheng, L., Li, Z., Xing, E., Ho, Q., Gonzalez, J., Stoica, I., Zhang, H., and Zhao, H. On optimizing the communication of model parallelism. *Proceedings of Machine Learning and Systems*, 5, 2023.

A PARALLELIZE LLM TRAINING

Parallel training strategies. To distribute LLM training computation over thousands of GPUs, three main categories of parallel strategies have been proposed.

Data parallelism (Li et al., 2020) distributes computation by dividing training batch across devices, where each GPU hosts a model replica for forward and backward propagation, and the gradients are synchronized by `AllReduce` operations. To further optimize memory usage, gradients, Zero Redundancy Optimizer (ZeRO) (Rajbhandari et al., 2020) and Fully Sharded Data Parallel (FSDP) (Zhao et al., 2023) further shard the optimizer states and parameters across multiple devices and gather them through additional communication when necessary.

Pipeline parallelism (Huang et al., 2019; Narayanan et al., 2019) partitions the model's computation across multiple layers into a series of stages, where each GPU handles one stage. During the forward propagation, the GPU serving stage-(j) sends the activations to the GPU serving stage-(j+1); during the backward pass, communication reverses direction to transfer the gradients.

Tensor model parallelism (Narayanan et al., 2021b) further partitions each transformer layer over multiple GPUs, where weight matrices are distributed row- or column-wisely. Two `AllReduce` operations are required to aggregate the layer output activations in forward pass and corresponding gradients in backward pass, respectively. Tensor model parallelism is communication intensive, but can effectively parallelize the computation when the connection between GPUs is fast (e.g., through NVLink).

System optimization for LLM training. Significant efforts have been made to *optimize parallel LLM training*, and various system optimizations (Park et al., 2024; Li et al., 2024; Ai et al., 2024; Jia et al., 2019; 2018; Cai et al., 2021) have been proposed to improve LLM training throughput. For example, Zero bubble parallelism enhances computational efficiency by reducing bubble overhead (Qi et al., 2024); A cross-mesh resharding mechanism was introduced to minimize communication overhead in tensor parallelism (Zhuang et al., 2023); Additionally, system optimizations such as gradient bucketing, gradient accumulation, and computation-communication overlap have been integrated into data parallel implementations (Li et al., 2020).

To improve training throughput, *optimizations for memory footprint* are also necessary. For example, activation recomputing significantly reduces memory footprint by recomputing the desired activation during backward propagation instead of storing it after forward computation (Chen et al., 2024). On the other hand, offloading activations to CPU RAM or SSD can effectively reduce GPU memory usage without impacting performance by adaptively overlapping data transfers with computation (Wu et al., 2024).

B DETAILS OF COST MODEL

In this section, we present our cost models. We first model the cost for each transformer layer, and then derive the end-to-end cost.

Modeling Cost Layer-wisely

- **Tensor model parallel communication cost**. Suppose activation recompute is enabled. A transformer layer runs over a set of GPUs $\mathbf{d}_{i,j}$. The tensor model parallel communication cost for a micro-batch—with a micro-batch size as B_{mb}—is:

$$\begin{aligned}&\textsc{Comm-TP-Fwd}(\mathbf{d}_{i,j}) = \textsc{Comm-TP-Bwd}(\mathbf{d}_{i,j})\\&= 4 \cdot \max_{d\in\mathbf{d}_{i,j}} \sum_{d'\in\mathbf{d}_{i,j}\setminus\{d\}} \left(\alpha_{d,d'} + \frac{B_{mb}SHB_{\text{type}}}{|\mathbf{d}_{i,j}|\beta_{d,d'}}\right)\end{aligned} \tag{3}$$

- **Computation cost**. Assume tensor model parallelism runs over the same type of device d, for one micro-batch, the computation cost for forward and backward passes is:

$$\begin{aligned}\textsc{Comp-TP-Fwd}(\mathbf{d}_{i,j}) &= \frac{24B_{mb}SH^2\left(1+\frac{S}{6H}\right)}{c_d|\mathbf{d}_{i,j}|}\\ \textsc{Comp-TP-Bwd}(\mathbf{d}_{i,j}) &= 2\times\textsc{Comp-TP-Fwd}(\mathbf{d}_{i,j})\end{aligned} \tag{4}$$

- **Integrate data parallel communication cost**. We model data parallel strategies (DDP, ZeRO-2, ZeRO-3) as a *per layer overhead* that captures communication and its overlap with computation. For consistency with the tensor-parallel model, we explicitly define the forward and backward computation time of a layer as:

$$\begin{aligned}T_f(\mathbf{d}_{i,j}) &= \textsc{Comp-TP-Fwd}(\mathbf{d}_{i,j}) + \textsc{Comm-TP-Fwd}(\mathbf{d}_{i,j})\\ T_b(\mathbf{d}_{i,j}) &= \textsc{Comp-TP-Bwd}(\mathbf{d}_{i,j}) + \textsc{Comm-TP-Bwd}(\mathbf{d}_{i,j})\end{aligned} \tag{5}$$

Let T_{comm} denote the communication cost of ReduceScatter and AllGather, which are data parallel collectives. We model T_{comm} as follows. Notice that for the AllReduce operation, we model the cost as $2T_{\text{comm}}$ since the AllReduce operation can be implemented via one ReduceScatter and one AllGather operation.

$$T_{\text{comm}}(k,\sigma) = \max_{d\in\mathbf{D}^k_{\text{dp}}} \sum_{d'\in\mathbf{D}^k_{\text{dp}}-\{d\}} \left(\alpha_{d,d'} + \frac{12H^2B_{\text{type}}}{\left|\mathbf{D}^k_{\text{dp}}\right|\beta_{d,d'}}\right) \tag{6}$$

Combining the above formulations, we define the per-layer cost under different data parallel strategies as follows. Since computation-communication overlapping would influence system efficiency (e.g., competing GPU SMs), we introduce constants θ_1, θ_2 for DDP and ZeRO-2's backward-pass and θ_{3f}, θ_{3b} for ZeRO-3's forward backward passes that represent the corresponding slow-

down. The constants are obtained through profiling.

$$\text{DP-OVERHEAD-LAYER}(k,\sigma) = \begin{cases} n_{mb}T_f + (n_{mb}-1)\,T_b + \theta_1 \cdot \max(T_b, 2T_{\text{comm}}), & \text{DDP} \\ n_{mb}T_f + (n_{mb}-1)\,T_b + \theta_2 \cdot \max(T_b, 2T_{\text{comm}}), & \text{ZeRO-2} \\ n_{mb}\theta_{3f} \cdot \max(T_f, T_{\text{comm}}) & \\ \quad + (n_{mb}-1)\,\theta_{3b} \cdot \max(T_b, T_{comm}) & \text{ZeRO-3} \\ \quad + \theta_{3b} \cdot \max(T_b, 2T_{comm}) & \end{cases} \tag{7}$$

Explainations for the above formulations. (i) For DDP, which replicates training states, one AllReduce operation is used for gradient communication. The last micro-batch can overlap backward-pass computation with gradient synchronization. (ii) For ZeRO-2, which shards gradients and optimizer states, after gradient computation, one ReduceScatter operation is used to synchronize gradients, followed by one AllGather operation to collect updated parameters. The last micro-batch can overlap backward-pass computation with gradients and updated parameters synchronization. (iii) For ZeRO-3, which fully shards training states, one AllGather operation is required to collect parameters before the forward-pass and backward-pass computation, and one ReduceScatter operation is used to synchronize gradients. All micro-batches can overlap forward-pass and backward-pass computation with AllGather of parameters, while the last micro-batch can further overlap computation with gradients synchronization.

This formulation ensures that tensor parallel computation and communication are consistently incorporated into data parallel modeling, while avoiding double-counting and explicitly capturing computation–communication overlap.

- **Pipeline parallel communication cost**. Pipeline communication occurs between adjacent stages. Define $\text{COMM-PP-HOP}(\mathbf{d}_{i,D^i_{pp}}, \mathbf{d}_{i,D^i_{pp}+1}) = 0$. Then:

$$\begin{aligned} &\text{COMM-PP-HOP}(\mathbf{d}_{i,j}, \mathbf{d}_{i,j+1}) = \\ &2 \cdot \min_{d \in \mathbf{d}_{i,j}, d' \in \mathbf{d}_{i,j+1}} \left[\left(\alpha_{d,d'} + \frac{B_{mb} S H B_{\text{type}}}{\beta_{d,d'}} \right) \right. \\ &\left. + \sum_{d'' \in \mathbf{d}_{i,j+1} \setminus \{d'\}} \left(\alpha_{d',d''} + \frac{B_{mb} S H B_{\text{type}}}{|\mathbf{d}_{i,j+1}| \beta_{d',d''}} \right) \right] \end{aligned} \tag{8}$$

Modeling Cost for Each Parallel Strategy

- **Pipeline stage cost**. For stage j in pipeline i:

$$\text{STAGE}(\mathbf{d}_{i,j}) = \sum_{k=1}^{l_{i,j}} \left[\text{DP-OVERHEAD-LAYER}(k,\sigma) \right] \tag{9}$$

- **Pipeline parallelism cost**. The total execution time for pipeline i is:

$$\begin{aligned} \text{PIPELINE-TIME}(i) = &\sum_{j=1}^{D^i_{pp}} \Big(\text{STAGE}(\mathbf{d}_{i,j}) \\ &+ \text{COMM-PP-HOP}(\mathbf{d}_{i,j}, \mathbf{d}_{i,j+1}) \Big) \\ &+ (n_{mb}-1) \cdot \max_{j=2,\ldots,D^i_{pp}} \Big(\text{STAGE}(\mathbf{d}_{i,j}) \\ &+ \text{COMM-PP-HOP}(\mathbf{d}_{i,j}, \mathbf{d}_{i,j+1}) \Big) \end{aligned} \tag{10}$$

Modeling end-to-end time The iteration time is determined by the slowest pipeline:

$$\text{TOTAL-TIME}(\sigma) = \max_{i=1,\ldots,D_{dp}} \{\text{PIPELINE-TIME}(i)\} \tag{11}$$

Modeling Memory Cost: Suppose full activation recompute and distributed data parallelism (DDP) are applied. The memory cost of parameters and activations for one layer can be estimated as follows:

$$\text{MEM-LAYER}\,(\sigma) = \frac{48H^2 B_{type}}{|\mathbf{d}_{i,j}|} + B_{mb} S H B_{type} \tag{12}$$

Under the 1F1B schedule, the peak memory is given by:

$$\text{MEM-PEAK}\,(\sigma) = L \cdot \text{MEM-LAYER}\,(\sigma)\,. \tag{13}$$

C DETAILED VALIDATIONS ON COST MODEL

Evaluate the simulation accuracy. We evaluate the accuracy of our simulation in Table 2. The simulation results (deviating less than 2.9%) closely match the actual outcomes across various GPU settings. Our algorithm can thereby accurately search for an effective parallel execution plan. To further validate our cost model, we present additional experiments in Appendix C. The results demonstrate that our cost model closely predicts both memory usage and per-batch communication overhead.

Comparisons on predicted and real memory costs (see 3) and communication costs (see 4) across different models validate our cost model. The predicted values closely align with the observed values during real execution.

D DETAILS OF EXPERIMENT SETUPS

We simulate the performance of HEXISCALE, Metis, and Galvatron on a large-scale cluster consisting of 240 heterogeneous GPUs, the results are summarized in Table 5:

Table 2. Comparison of the real and simulated achieved PFLOPS across different experimental settings.

Model	Setting	Real (PFLOPS)	Simulated (PFLOPS)	Deviation (%)
LLAMA-2 (7B)	Homo-Ethernet	2.05	2.11	2.84%
	Homo-RDMA	2.68	2.70	0.74%
	Hetero-Setting-1	1.57	1.61	2.48%
	Hetero-Setting-2	1.65	1.66	0.60%
LLAMA-2 (13B)	Homo-Ethernet	1.97	2.02	2.48%
	Homo-RDMA	2.55	2.61	2.30%
	Hetero-Setting-1	1.37	1.40	2.14%
	Hetero-Setting-2	1.54	1.56	1.28%
LLAMA (30B)	Homo-Ethernet	2.78	2.86	2.80%
	Homo-RDMA	3.89	3.99	2.51%
	Hetero-Setting-3	2.49	2.54	1.97%

Table 3. Estimated vs. Observed Peak Memory Usage.

Model	Estimated Peak Memory (GB)	Observed Peak Memory (GB)
LLAMA-2 (7B)	48.2	52.3
LLAMA-2 (13B)	95.8	98.4
LLAMA (30B)	239.4	246.2

E ALGORITHM OPTIMALITY

Due to the NP-hard nature of this complex problem, determining an upper bound is particularly challenging, as it requires either a provably optimal solution or a heuristic guarantee under specific constraints. To evaluate the optimality of our scheduling algorithm, we compare it against the MILP (Mixed-Integer Linear Programming) approach, a widely adopted method in resource allocation problems. ILP/MILP-based solvers have been integrated into prominent systems such as Alpa (Zheng et al., 2022), FlexSP (Wang et al., 2024b), and Helix (Mei et al., 2024), demonstrating their reliability and effectiveness in resource allocation and parallel strategy determination for LLM training and inference.

We run both a MILP algorithm and our algorithm, and compare the performance gap. The MILP algorithm guarantees optimal results, except its running time grows exponentially.

In Table 6, we simulate the system performance for heterogeneous settings that appear in §5.2. As shown in the table, the scheduling algorithm of HEXISCALE achieves performance comparable to MILP. Additionally, the graph partitioning-based scheduling algorithm used by HEXISCALE excels in scalability for complex and large environments, significantly outperforming MILP in terms of search time (HEXISCALE completes the search within minutes, whereas MILP requires hours). In summary, our algorithm delivers performance close to the optimal solution while demonstrating exceptional efficiency.

Table 4. Estimated vs. Observed Communication Latency in Heterogeneous Settings. (following Figure 7)

Model	Estimated Latency (ms)	Observed Latency (ms)
LLAMA-2 (7B)	32.31	32.66
LLAMA-2 (13B)	103.24	104.28
LLAMA (30B)	129.27	130.51

Table 5. GPU compositions in the simulated large-scale cluster.

GPU Type	Num Instances	Bandwidth (GB/s)
8×3090	5	24
4×3090	6	24
8×4090	20	32
8×A800	2	200

F SIMULATION EXPERIMENTS

Large-scale simulation. To further evaluate HEXISCALE on larger models and cluster configurations, we simulate performance of HEXISCALE, Metis, and Galvatron on a 240 heterogeneous GPU large-scale cluster with LLAMA (30B) and LLAMA-2 (70B) under both low (5.6 Gbps) and high (80 Gbps) inter-machine network bandwidth. Detailed cluster information is provided in Appendix D. As illustrated in Figure 13, under both low and high inter-machine network connections, HEXISCALE exhibits superior performance on the large-scale cluster with different model sizes, achieving performance improvements of 1.6× to 1.9× relative to Metis, and 2.1× to 3.3× relative to Galvatron. Additionally, Metis's and Galvatron's search algorithms fail to efficiently identify high-performance parallelism strategies in large-scale clusters, requiring hours to complete. In contrast, HEXISCALE demonstrates superior algorithmic scalability, as discussed in §5.4.

Performance under varied network. Network connectivity plays a critical role in heterogeneous training. To evaluate how HEXISCALE performs under varying network conditions, we simulate system performance by adjusting inter-machine bandwidths—from 4 Gbps to 80 Gbps—for LLAMA-2 (7B), LLAMA-2 (13B), and LLAMA (30B) models in heterogeneous settings described in §5.2. The simulation results are summarized in Figure 14. When inter-machine bandwidth degrades from 5.6 Gbps to 4 Gbps, HEXISCALE maintains robust performance with only moderate degradation. Conversely, as bandwidth increases to 80 Gbps, HEXISCALE achieves a notable boost in system performance. At 80 Gbps, HEXISCALE delivers throughput comparable to the homogeneous baseline with RDMA—up to 0.95×, and 0.80× on average—despite operating in a heterogeneous environment. Importantly, upgrading the net-

Table 6. Throughput comparison of MILP and HEXISCALE across different models and heterogeneous settings.

Model	Setting	Method	Achieved PFLOPS
Llama-2 (7B)	Hetero-Setting 1	MILP	1.62
		HEXISCALE	1.62
Llama-2 (13B)	Hetero-Setting 1	MILP	1.46
		HEXISCALE	1.44
Llama-2 (7B)	Hetero-Setting 2	MILP	1.70
		HEXISCALE	1.66
Llama-2 (13B)	Hetero-Setting 2	MILP	1.63
		HEXISCALE	1.56
Llama (30B)	Hetero-Setting 3	MILP	2.69
		HEXISCALE	2.54

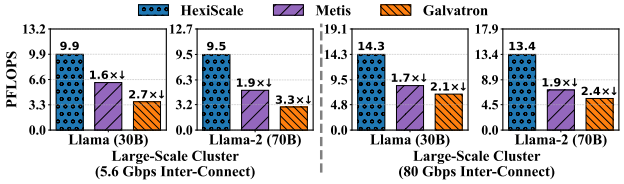

Figure 13. Large scale simulation of HEXISCALE, Metis, and Galvatron on LLAMA (30B) and LLAMA-2 (70B) models.

work fabric (e.g., enabling RoCE at 40 Gbps or RDMA at 80 Gbps) is significantly more cost-effective than replacing GPUs, suggesting that heterogeneous training—enabled by HEXISCALE—offers a promising and economical alternative to conventional homogeneous setups.

G EVALUATION CONFIGURATION DETAILS

G.1 Case study

In this subsection, we present the case study of scheduling in the heterogeneous setting 3 to compare the parallel strategy across HEXISCALE, Metis, and Galvatron

Strategy of HEXISCALE. Table 7 illustrates the parallel strategy discovered by our algorithm. Within each pipeline, distinct hardware characteristics are effectively considered by applying locally high-performance parallel strategies and strategically assigning transformer layers to balance computation across pipeline stages. The number of pipelines is fine-tuned to four, leveraging the benefits of data parallelism while avoiding out-of-memory issues and too much communication overhead. Data parallel communication overhead remains manageable, as pipeline execution time is much longer. These results confirm that HEXISCALE effectively accounts for hardware heterogeneity, generating high-performance parallel strategies.

The parallel strategies for Metis and Galvatron are as follows:

Strategy of Metis. Metis prefers using intra-machine data parallelism, thereby constructing one pipeline with 8 stages; the results are summarized in Table 8.

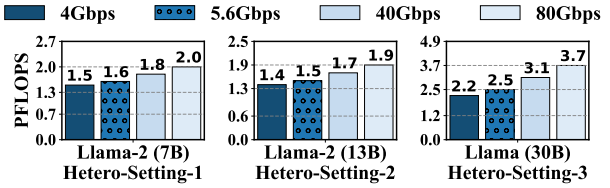

Figure 14. Simulated throughput achieved by HEXISCALE under varied inter-machine network connections.

Table 7. Parallelism strategy discovered by HEXISCALE when training LLAMA (30B) in heterogeneous setting 3. There are four pipelines (DP=4) with various pipeline layouts.

Pipeline Index	Stage Index	GPU Allocation	Layer Count	TP Degree
0	{0}	8×A800	60	8
1	{0,1,2}	4×4090	18	4
	{3}	2×3090	6	2
2	{0,1,2}	4×4090	18	4
	{3}	2×3090	6	2
3	{0,1}	4×4090	18	4
	{2,3,4,5,6,7}	2×3090	4	2

Strategy of Galvatron. The strategy for Galvatron is fine-tuned as $(D_{dp}, D_{tp}, D_{pp}) = (4, 2, 7)$. With a higher data parallel degree, the out-of-memory issue persists. With a lower data parallel degree, the system performance is compromised by either significant pipeline bubbles or communication overhead. Tensor model parallel degree is fine-tuned to 2; otherwise, for higher tensor model parallel degrees, 3090 incurs significant communication overhead and compromises the overall system performance.

G.2 Other Evaluation Details

Pipeline schedules. For all systems, we adopt 1F1B when enabling pipeline parallelism.

Data-parallel sharding strategies. Megatron/Metis uses DDP, Galvatron/AMP/Espresso employs ZeRO-3. HEXISCALE adopts ZeRO-3 for heterogeneous clusters comprising Hopper GPUs, and adopts ZeRO-2 for other settings.

FSDP configurations. FSDP uses ZeRO-3 when RDMA is available and ZeRO-2 when only Ethernet is available. We do not enable activation offloading for FSDP, since empirically it degrades performance on A800 GPUs.

Batch size selection criteria. We tune the global-batch and micro-batch size that leads to the highest throughput for each system.

Parallel Strategy and batch size details. For each system, we summarize the parallel strategy and batch size used in

Table 8. Parallel Strategy of Metis under Heterogeneous Setting 3.

Stage Index	GPU Allocation	Layer Count	(DP, TP) Degree
0	8xA800	20	(8, 1)
1,2,3,4	8x4090	8	(8, 1)
5	8x3090	4	(8, 1)
6	4x3090	2	(4, 1)
7	4x3090	2	(4, 1)

our evaluations in Table 9, Table 10, Table 11, and Table 12.

Table 9. Batch Configuration for HEXISCALE and Metis on 30B (Heterogeneous Setting 3).

System	Batch per Pipeline	Micro Batch
HEXISCALE (4 pipelines)	80,64,64,48	2
Metis (1 pipeline)	256	8

Table 10. Parallelism Strategy and Batch Sizes for Megatron, Galvatron, FSDP, and AMP/Espresso.

System	Model	Cluster	(DP, TP, PP)	Global Batch	Micro Batch
Megatron	Llama-7B	2x8xA800	(8,1,2)	384	4
Megatron	Llama-13B	2x8xA800	(8,1,2)	192	2
Megatron	Llama-30B	4x8xA800	(4,4,2)	192	2
Megatron	Llama-7B	Hetero Setting 1	(5,1,8)	120	1
Megatron	Llama-13B	Hetero Setting 1	(2,1,20)	128	1
Megatron	Llama-7B	Hetero Setting 2	(2,1,16)	128	1
Megatron	Llama-13B	Hetero Setting 2	(2,1,16)	128	1
Galvatron	Llama-7B	2x8xA800	(8,1,2)	384	4
Galvatron	Llama-13B	2x8xA800	(8,1,2)	192	2
Galvatron	Llama-30B	4x8xA800	(4,4,2)	192	2
Galvatron	Llama-7B	Hetero Setting 1	(8,1,5)	128	2
Galvatron	Llama-13B	Hetero Setting 1	(4,1,10)	128	2
Galvatron	Llama-7B	Hetero Setting 2	(8,1,4)	128	2
Galvatron	Llama-13B	Hetero Setting 2	(8,1,4)	128	2
Galvatron	Llama-30B	Hetero Setting 3	(7,1,8)	96	2
FSDP	Llama-7B	2x8xA800	(16,1,1)	384	4
FSDP	Llama-13B	2x8xA800	(16,1,1)	192	4
FSDP	Llama-30B	4x8xA800	(32,1,1)	192	2
AMP/Espresso	Llama-30B	16xH800+16xH20	(2,4,4)	384	4
AMP/Espresso	Llama-30B	16xH800+24xH20	(2,4,5)	384	4
AMP/Espresso	Llama-30B	16xH800+32xH20	(2,4,6)	384	4
AMP/Espresso	Llama-30B	Hetero Setting 3	(7,1,8)	128	2

Table 11. HEXISCALE Parallel Strategy for 7B/13B Models.

Model	Cluster	#Pipelines	TP per Stage	GPU Allocation	Global Batch	Micro Batch
Llama-7B	Hetero Setting 1	8	1,1,2,1	1x3090+1x3080+3x4090	576	4
Llama-13B	Hetero Setting 1	4	1,1,1,1,2,2,2	2x3090+2x3080+6x4090	384	2
Llama-7B	Hetero Setting 2	8	1,1,1,1	1x3090+1x3080+1x4090+1xA800	576	4
Llama-13B	Hetero Setting 2	8	1,1,1,1	1x3090+1x3080+1x4090+1xA800	384	2

Table 12. HEXISCALE Strategy for 30B on Hopper Heterogeneous Clusters.

Cluster	Parallel Strategy	Batch per Pipeline	Micro Batch
16xH800+16xH20	P0-1: 8xH800; P2-3: 8xH20	160,160,32,32	4
16xH800+24xH20	P0-1: 8xH800; P2-4: 8xH20	160,160,32,32,32	4
16xH800+32xH20	P0-1: 8xH800; P2-5: 8xH20	160,160,32,32,32,32	4