HAPT: Heterogeneity-Aware Automated Parallel Training on Heterogeneous Clusters

Antian Liang*
Fudan University
Shanghai, China
atliang23@m.fudan.edu.cn

Zhigang Zhao*
Fudan University
Shanghai, China
zgzhao21@m.fudan.edu.cn

Kai Zhang†
Fudan University
Shanghai, China
zhangk@fudan.edu.cn

Xuri Shi
Fudan University
Shanghai, China
xrshi23@m.fudan.edu.cn

Chuantao Li
Shandong Computer Science Center (National Supercomputer Center in Jinan)
Jinan, Shandong, China
chuantaolisdsc@gmail.com

Chunxiao Wang
Shandong Computer Science Center (National Supercomputer Center in Jinan)
Jinan, Shandong, China
wangchx@sdas.org

Zhenying He
Fudan University
Shanghai, China
zhenying@fudan.edu.cn

Yinan Jing
Fudan University
Shanghai, China
jingyn@fudan.edu.cn

X. Sean Wang
Fudan University
Shanghai, China
xywangCS@fudan.edu.cn

Abstract

With the rapid evolution of GPU architectures, the heterogeneity of model training infrastructures is steadily increasing. In such environments, effectively utilizing all available heterogeneous accelerators becomes critical for distributed model training. However, existing frameworks, which are primarily designed for homogeneous clusters, often exhibit significant resource underutilization when deployed on heterogeneous accelerators and networks. In this paper, we present Hapt, an automated parallel training framework designed specifically for heterogeneous clusters. Hapt introduces a fine-grained planner that efficiently searches a wide space for the inter-operator parallel strategy, enabling Hapt to alleviate communication overheads while maintaining balanced loads across heterogeneous accelerators. In addition, Hapt implements a heterogeneity-aware 1F1B scheduler that adaptively adjusts the execution timing and ordering of microbatches based on network characteristics, maximizing computation–communication overlap under cross-cluster interconnects while incurring only minimal memory overhead. Our evaluation results show that Hapt can deliver 1.3×–1.6× higher performance on heterogeneous clusters than state-of-the-art training frameworks.

1 Introduction

In the pursuit of higher performance and broader applicability, the scale of deep learning models has rapidly grown to $O(10^{12}–10^{13})$ parameters in recent years (e.g., LLaMA-3 series, GPT models)[1, 2, 20, 21]. Meeting this demand requires massive device memory and computation resources,

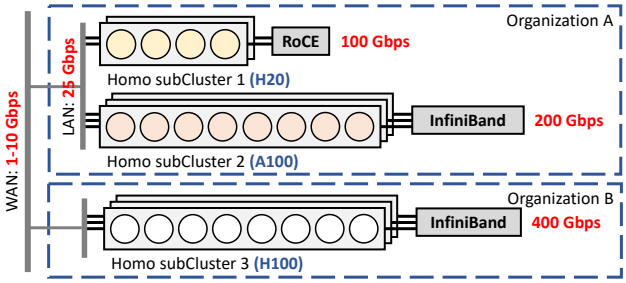

Figure 1. Example heterogeneous cluster composed of multiple homogeneous subclusters, with fast interconnects within subclusters but slower interconnects across them.

which model developers typically address by deploying large homogeneous clusters composed of identical accelerators. However, hardware vendors such as NVIDIA now release new accelerator architectures on an annual cycle[15, 16]. Due to budget constraints, smaller organizations such as startups and research labs typically acquire only a limited number of the latest accelerators at each time. Over time, these organizations gradually accumulate multiple homogeneous subclusters, each equipped with a different type of accelerator. Together, these subclusters constitute heterogeneous computing infrastructures.

For a homogeneous cluster, large-scale models are typically trained using distributed frameworks such as Megatron [7, 14, 26], DeepSpeed [22], and Alpa [37]. However, directly applying these frameworks to heterogeneous clusters consisting of multiple homogeneous subclusters results in severe resource underutilization. The underlying reason for this underutilization is two-fold. First, these frameworks assume all accelerators have identical computation and memory capacities, thereby assigning equal workloads to them,

*Both authors contributed equally to this work.
†Corresponding author

causing significant load imbalancing. Second, as shown in Figure 1, homogeneous subclusters are often distributed across distinct physical locations. In this case, communication across subclusters typically relies on interconnects with limited bandwidth (e.g., 1–25 Gbps [27, 32, 33]), whereas communication within a subcluster usually benefits from high-bandwidth interconnects of 100–800 Gbps [17]. These limitations have been observed in practice, as reported by Um et al. [27], where applying Alpa to train a MoE model on a P100-V100 cluster results in up to a 7.4× performance degradation. Consequently, despite the increase in total resources, heterogeneous clusters can barely be utilized efficiently for model training. In this context, enabling efficient training across multiple homogeneous subclusters is increasingly attractive, as it allows organizations to pool all available resources for larger-scale model training.

Recent studies [27, 32, 36] have focused on automated planning of parallel strategies by exploring the search space along intra-operator parallelism (e.g., data and tensor parallelism) and inter-operator parallelism (e.g., pipeline parallelism) to achieve load balancing on heterogeneous clusters. For example, Metis [27] and HexiScale [32] investigate: (1) intra-operator(op) search space, where model or data tensors can be asymmetrically sliced so that heterogeneous accelerators executing the same operator in parallel are each assigned proportionally balanced workloads; and PipePar [34] focus on (2) inter-op search space, where the computation graph can be partitioned into pipeline stages with different workloads so that these stages are mapped onto accelerators with different capabilities, thereby achieving balanced execution across the pipeline. While these systems achieve higher throughput than directly deploying frameworks originally designed for homogeneous clusters in heterogeneous environments, they still fail to reach the expected utilization (measured as Model FLOP Utilization, MFU) of heterogeneous clusters. For instance, as reported by Yan et al. [32], training LLaMA-2 30B on a heterogeneous cluster yields only 17.1% and 28.0% MFU with Metis and HexiScale, respectively.

We analyze experimental results on heterogeneous clusters and observe that this under-utilization stems from three key factors. ***i)*** *Expensive cross-cluster intra-op communication*. Exploring both intra- and inter-op search spaces can indeed improve load balance across heterogeneous accelerators. However, applying intra-op parallelism across subclusters connected by slow interconnects introduces substantial collective communication overhead [27, 32, 35]. In HexiScale's experiments [32], the overhead of intra-op communication is reported to be about 50% of the total computation overhead. ***ii)*** *Limited inter-op search space*. The search space of inter-op parallel strategies scales with the number of operators in the computation graph, where a large-scale model typically consists of 10^4–10^5 operators. Exhaustively planning across this space is prohibitively expensive [37]. To reduce the cost, existing systems cluster operators into coarse-grained layers for planning. However, planning at a coarse layer granularity often fail to generate a balanced strategy on heterogeneous clusters, as it prevents precise alignment between pipeline stage workloads and the computational capacities of heterogeneous accelerators. For instance, experiments with PipePar [34] on training ResNet-50 using a heterogeneous GPU cluster showed that the best strategy still left the longest stage with 1.5× the compute cost of the shortest stage. Planning at a finer granularity can narrow this gap, but the overhead grows rapidly as the granularity increases, quickly becoming prohibitive. Our experiments show that training a GPT-39B model on our V100–A100 heterogeneous cluster can achieve balanced loads only at a granularity on the order of 10^2 layers. However, it takes more than five days to derive the parallel strategy with such a fine granularity, which is impractical. ***iii)*** *Cross-cluster inter-stage communication stalls computation*. When inter-op parallelism spans multiple subclusters, limited cross-cluster bandwidth often causes inter-stage communication to block computation, leading to substantial pipeline bubbles. Although recent pipeline schedulers can partially hide this latency by launching additional forward microbatches during warmup, they are unaware of network heterogeneity and thus can hide only up to 50% of the theoretical upper bound on inter-stage communication latency, while incurring substantial memory overhead from buffering nearly 2× intermediate results [38].

To bridge this gap, we introduce Hapt, an automated framework for distributed parallel training deep learning models. Our key contributions are:

(1) Fine-grained inter-op parallel strategy planner. Hapt applies intra-op parallelism within homogeneous subclusters and explores a fine-grained inter-op search space among the heterogeneous cluster. To overcome the planning overhead, we introduce a set of aggressive pruning and parallelization techniques. Specifically, we heuristically identifies repeated modules in the computation graph and performs operator clustering only within modules. This design preserves the structural information in the original graph, which can then be exploited to extensively prune the profiling process. Building on this, we design a search algorithm to efficiently generate heterogeneous inter-op parallel strategies at fine layer granularity, enhanced with several system-level optimizations, including sparsity indexing, bidirectional pruning, and batched parallel search, which systematically accelerate the searching process. Together, these techniques enable Hapt to plan at up to 20× finer layer granularity than prior approaches [37], while keeping the overhead tractable and yielding load-balanced parallel strategies.

(2) Heterogeneity-aware 1F1B scheduler. HAPT formulates communication–latency hiding as an optimization problem, optimizing the execution timing and ordering of microbatches at each stage. We model pipeline execution latency with a directed acyclic graph (DAG) abstraction and derive the H-1F1B scheduling strategy, which we formally prove to achieve the theoretical upper bound on the amount of communication latency that can be hidden, while incurring only minimal additional memory overhead.

(3) Evaluations. We implement HAPT, evaluating it on clusters of up to 64 GPUs. Across diverse heterogeneous settings, HAPT achieves 1.3×–1.6× higher throughput than state-of-the-art baseline. Moreover, on heterogeneous clusters where cross-cluster interconnects are limited to 5 Gbps, HAPT delivers throughput comparable to Megatron (latest version) running on homogeneous clusters fully connected via 200 Gbps RDMA, under the same theoretical peak FLOP/s. [1]

2 Background and Motivations

2.1 Background on Distributed Parallel Training

Distributed training partitions the model and training data into smaller units for parallel execution across multiple accelerators. To achieve this, the system employs several parallel strategies, which typically includes intra-op parallelism (e.g., data [23] or tensor [26] parallelism) and inter-op parallelism (e.g., pipeline parallelism [6]), and employs a scheduler that orchestrates how forward and backward computations are executed in parallel across the cluster [9, 12, 13, 19].

For parallel strategy planning, manual frameworks such as Megatron [7] require developers to manually configure parallel strategies, which demands deep expertise in the underlying infrastructure and parallelization techniques. This burden has motivated the development of automated planners. For example, Alpa [37] uses a hierarchical planner to automatically apply intra- and inter-op parallelism for homogeneous clusters. It linearizes the model's computation graph into a topologically ordered operator sequence. The cluster is modeled as a $DeviceMesh(N, M)$, with N nodes and M devices per node. The planner proceeds in two steps. (i) Profiling: The operators are first clustered into L layers to shrink the search space. The framework then enumerates all adjacent layer subsequence as candidate pipeline stages, and all possible submeshes (n, m) sliced from the DeviceMesh (N, M). For each candidate stage–mesh pair, the planner determines the optimal intra-op parallel strategy and profiles its execution information, including compute cost and memory footprint. (ii) Searching: Based on the profiled results, the planner formulates the search for inter-op parallel strategies as an optimization problem that minimizes end-to-end latency while balancing compute cost across stages. The search algorithm compares the end-to-end latency for different combinations of stage–mesh pairs and selects the optimal one as the inter-op parallel strategy.

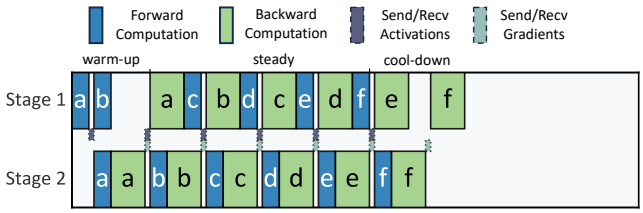

Figure 2. Classic 1F1B pipeline scheduler.

For pipeline scheduling, it accelerates inter-op parallelism by splitting a batch into multiple microbatches that are executed in a pipelined manner [6, 12, 13]. Specifically, each microbatch completes its forward and backward passes by traversing all pipeline stages. Once stage i finishes the forward computation, its output activations are sent to stage i+1; after stage i+1 completes the backward computation, the resulting gradients are propagated back to stage i. The pipeline scheduler determines the timing and ordering of forward and backward computations across all stages. As shown in Figure 2, the classic 1F1B scheduler [12, 13] proceeds in three phases: (i) During the *warm-up* phase, stage i launches $(\#\text{stage} - i + 1)$ forward microbatches (with #stage denoting the total number of stages); (ii) in the *steady* phase, each stage alternates between forward and backward passes in a one-forward–one-backward (1F1B) pattern; (iii) in the *cool-down* phase, the remaining backward passes are drained.

2.2 Load Imbalance of Existing Parallel Strategy Planning on Heterogeneous Device

Planners designed for homogeneous clusters cannot adapt in heterogeneous settings, as they assume identical devices and assign equal workloads, leading to severe load imbalance and low utilization [27, 32].

2.2.1 Limited Search Space Incurs Load Imbalance.

Recent work [8, 27, 30, 32, 35] has begun to explore the search space along intra- and inter-operator(op) parallelism to achieve load balancing on a heterogeneous cluster. For instance, *Metis* assigns different sizes of minibatches to GPUs with different compute capabilities in data parallelism, and distributes pipeline stages with different workloads across heterogeneous device groups in pipeline parallelism [27]. HexiScale extends this direction by enabling asymmetric sharding of tensor dimensions across mixed GPU types [32]. However, intra-op parallelism requires collective communication across all participating devices [8, 26, 31]. In many real-world heterogeneous clusters, cross-cluster communication is limited to slow Ethernet or even WAN links. Such slow networks significantly slow down collective communication. As reported in the experiments of Yan et al.[32], when the inter-cluster bandwidth is limited to 5 Gbps, training LLaMA-2 70B with Metis and HexiScale achieves only 17.1% and 27.2% MFU, respectively.

[1] We open-source HAPT at https://github.com/Lssyes/hapt-hetero.

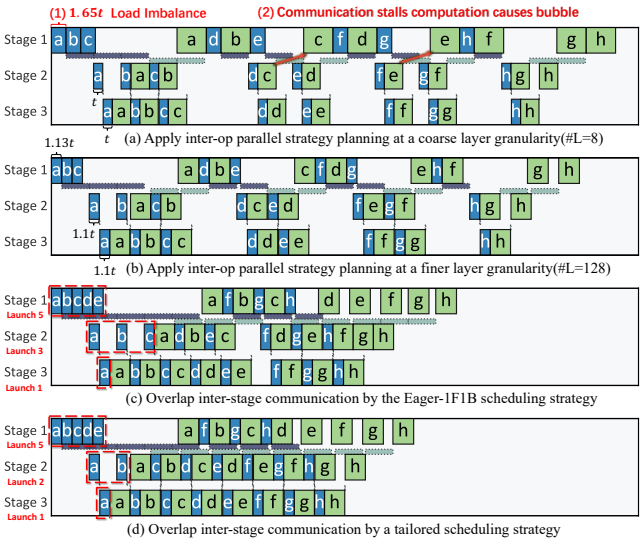

Figure 3. Pipeline timeline of case studies.

Another approach confines intra-op parallelism to homogeneous meshes and introduces heterogeneity only at the inter-op level [24, 34]. This avoids cross-cluster collective communication over slow links, but restricts the search space of candidate meshes. At the same time, inter-op strategy planning is usually performed at a coarse layer granularity to avoid high profiling and searching overhead, which limits the search space of candidate stages [27, 32, 37]. Together, these two restrictions yield a much smaller search space of candidate stage–mesh pairs, making it difficult to balance compute costs across stages. For example, under this restricted search space, experiments with *PipePar* on ResNet-50 using a (4×2) heterogeneous GPU cluster showed that the best strategy left the longest stage with 1.5× the compute cost of the shortest [34]. While planning at finer layer granularity can make a more balanced parallel strategy, however, profiling scales as $\sim O(L^2)$ and search as $\sim O(L^5)$. According to Alpa's experiments [37], even on homogeneous clusters, generating a parallel strategy with $\#L = 8$ layers for GPT-39B already takes about an hour [37]. We evaluate planning overhead at finer granularity(e.g., $\#L = 146$), and it increases dramatically, which can take more than 5 days, even erasing the training-time saved by a more balanced parallel strategy.

2.2.2 Case Study. We conduct a case study to examine how different layer granularities in inter-op parallelism affect load balance, under the constraint that intra-op parallelism is restricted to homogeneous meshes.

We train a GPT model on a heterogeneous cluster consisting of two meshes: $DeviceMesh_{A100}(2, 2)$ with 2×2 A100 GPUs, and $DeviceMesh_{V100}(1, 2)$ with 1×2 V100 GPUs. The A100 nodes are connected by a 200 Gbps InfiniBand network, while cross-cluster communication is limited to a 5 Gbps Ethernet link. We use an off-the-shelf operator clustering tool[10] to group the layer sequence into 8 (same with Alpa) and 128 layers, respectively, with each layer having approximately equal workload. We explore the inter-op parallel strategies in this constrained search space and summarize the optimal result in Table 1.

Table 1. Inter-op parallel strategies for the case study. Notation: $mesh_{V100}(1, 2)$ denotes a homogeneous submesh with (1×2)V100 GPUs; the [16.8%] indicates this submesh's share of the cluster's peak FLOP/s.

#L	Stage	Submesh	Layers	Layer%	Cost
8	1	$mesh_{V100}(1,2)$[16.8%]	1,2	25.0%	$1.65t$
	2	$mesh_{A100}(1,2)$[41.6%]	3,4,5	37.5%	t
	3	$mesh_{A100}(1,2)$[41.6%]	6,7,8	37.5%	t
128	1	$mesh_{V100}(1,2)$[16.8%]	1–22	17.2%	$1.13t$
	2	$mesh_{A100}(1,2)$[41.6%]	23–76	41.4%	$1.10t$
	3	$mesh_{A100}(1,2)$[41.6%]	76–128	41.4%	$1.10t$

As shown in Figure 3(a), inter-op parallel strategy planning at a coarse layer granularity ($\#L = 8$) leads to severe load imbalance. Layer 1–2 are mapped to stage 1 running on 2× V100 GPUs, where only 16.8% of the total compute capacity is responsible for 25% of the workload. Using the compute cost of a forward microbatch in stages 2 and 3 as a reference t, stage 1 requires $1.65t$ to complete its computation. This imbalance arises because the restricted search space cannot adequately match workloads to the heterogeneous compute capabilities. In contrast, Figure 3(b) shows planning at a finer granularity ($\#L = 128$). Here, stage 1 is assigned less workload, while stages 2 and 3 are assigned more, leading to a computation graph partitioning that more closely matches each mesh's compute capacity. This fine-grained plan achieves improved load balance and higher throughput. When $B = 128$ microbatches, assuming full inter-stage communication overlap, the fine-grained plan achieves a 40.1% throughput improvement over the coarse-grained plan.

2.3 Limited Latency Hiding of Pipeline Schedulers on Heterogeneous Networks

In the previous case study, we observed that communication stalls are another significant source of performance degradation, as the classic 1F1B scheduler enforces strict data dependencies, often forcing computation to wait for slow inter-stage communication, which creates pipeline bubbles during the steady phase.

2.3.1 Overlap-friendly Scheduling Strategy. Communication stalls in the steady phase can be mitigated by overlapping communication with computation. For example, *Eager-1F1B* [38], a variant of the classic 1F1B scheduler, launches $2 \times (\#\text{stage} - i) + 1$ forward microbatches during warm-up at stage i. Concretely, this means that stage i launches two more forward microbatches than stage i+1, thereby creating more

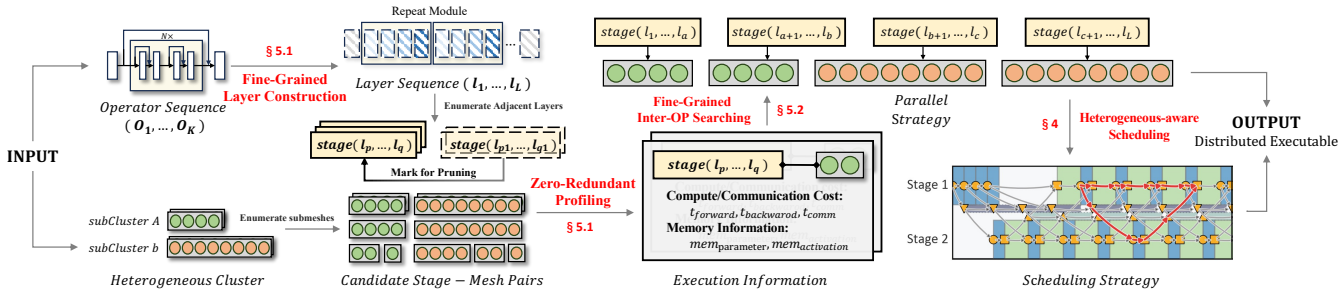

Figure 4. Overview of HAPT workflow.

opportunities to hide communication latency. However, it applies a fixed scheduling strategy, where each stage launches a static number of extra forward microbatches, which is less effective under heterogeneous networks. Specifically: (i) even when inter-stage communication latency is small and requires no hiding, the scheduler still launches two additional forward microbatches. For large-scale models with 16 or more stages, this behavior nearly doubles the activation memory compared to the classic 1F1B; and (ii) its ability to hide communication is fundamentally capped at 50% of the theoretical upper bound. Let t_f and t_b denote the forward and backward compute cost of a stage. With Eager-1F1B, only inter-stage communication whose cost does not exceed $(t_f+t_b)/2$ can be fully overlapped, which represents 50% of the theoretical upper bound.

2.3.2 Case Study. We conduct a case study to illustrate why existing overlap-friendly schedulers perform suboptimally on heterogeneous slow networks and to motivate our tailored scheduling strategy. The setup uses the same model, cluster, and parallel strategy as in Figure 3(b), but applies different pipeline scheduling strategies.

As shown in Figure 3(c), Eager-1F1B launches $\{5, 3, 1\}$ forward microbatches at stages 1–3. In this strategy, stage 2 launches two more microbatches than stage 3, even though the two are connected via InfiniBand and incur negligible communication latency. Overlapping such fast communication is unnecessary and only wastes memory. Similarly, stage 1 launches two more microbatches than stage 2, which is still insufficient to fully hide the communication latency between them. Such delays over slow interconnects are frequently encountered in heterogeneous environments[28, 33].

To further eliminate the steady-phase bubbles observed in Fig. 3(c), we manually configure a tailored scheduling strategy, illustrated in Fig. 3(d). Compared to Eager-1F1B, this strategy launches $\{5, 2, 1\}$ forward microbatches at stages 1–3, respectively. It avoids redundant launches between stages 2 and 3 which are connected by fast link, while increasing the communication-hiding capacity between stages 1 and 2 where communication cost range from $(t_f+t_b)/2$ to (t_f+t_B). In this way, additional launches occur only when necessary. For large-scale models with 16 or more stages on heterogeneous networks, this strategy significantly reduces activation memory consumption compared to Eager-1F1B, while achieving greater communication overlap.

3 Overview

To address these challenges, we propose HAPT, a system that automates parallel training of deep learning models on heterogeneous GPU clusters. Specifically, to address load imbalance, HAPT apply inter-op parallelism by partitioning the computation graph on fine-grained layers and map them onto a heterogeneous cluster, while retaining standard intra-op parallelism within homogeneous clusters. This design avoids heavy collective communication overheads that would otherwise occur on cross-cluster links. To resolve the trade-off between load imbalance (caused by coarse layer granularity) and inter-op parallel strategy planning overhead (caused by fine layer granularity), HAPT iteratively scans the computation graph, heuristically identifies repeated and non-repeated modules, and clusters operators within each module into layers, concatenating to a layer sequence that preserves the repeated structural patterns of the original graph. These structural layers generate multiple identical candidate stage–mesh pairs, which is exploited by our profiler to perform efficient pruning, achieving both fine layer granularity and low profiling overhead. Based on the profiling results, we formulate inter-op parallel strategy searching as an optimization problem and design a dynamic programming (DP) algorithm to derive balanced parallel strategies across stages for the heterogeneous cluster. Furthermore, we incorporate pruning and system-level optimizations to alleviate the search overhead on such fine-grained layers. In addition, to mitigate communication stalls in heterogeneous networks, we formulate the communication–latency hiding problem in pipeline scheduling as an optimization problem which resolves the number of forward microbatches to launch during warm-up for each stage. Building on this formulation, we model the end-to-end latency using a DAG abstraction and adaptively derive an optimal scheduling strategy that maximizes latency hiding while incurring only minimal additional memory overhead.

To achieve these, as shown in Figure 4, HAPT takes a topologically ordered operator sequence and a heterogeneous

cluster as input. It first performs *fine-grained layer construction* (§5.1) on the operator sequence and output structural layers. Next, HAPT slices the heterogeneous cluster into submeshes and enumerates all adjacent layer subsequences as candidate stages. Since many of these stage candidates are structurally identical, HAPT prunes the redundant stages and performs *zero-redundant profiling* (§5.1) to collect execution information for each unique stage–mesh pair, including compute cost, communication cost, and memory footprint. Based on these, we perform *fine-grained inter-op searching* (§5.2) to generate parallel strategies efficiently, applying several system-level optimizations including sparsity indexing, bidirectional pruning, and batched parallel search to systematically accelerate strategy generation. The H-1F1B scheduler then generates a *heterogeneity-aware scheduling* (§4) strategy that based on network characteristics to maximize computation–communication overlap on slow links. Finally, the combined scheduling and parallel strategies are compiled into a distributed executable for model training.

4 Heterogeneity-aware 1F1B Scheduler

In this section, we first introduce the design of the H-1F1B scheduler, as inter-op parallel strategy planning depends on the memory constraints imposed by the pipeline scheduler.

4.1 Design of H-1F1B

Unlike 1F1B [12] and Eager-1F1B [38], which launch a fixed number of forward microbatches at each stage during warm-up, H-1F1B adopts a tailored strategy that adjusts the launch count based on inter-stage communication costs. Specifically, proceeding in descending stage order from S to 1, each stage i launches one more forward microbatch than stage i+1. When the communication cost between stage i and stage i+1 is higher, stage i can launch additional forward microbatches to better overlap communication. The intuition is that launching more forward microbatches enlarges the temporal gap between the forward and backward passes of the same microbatch in the steady phase, creating more slack to hide communication latency. For example, as shown in Figure 5(a,b), although both cases incur the same compute and communication cost per stage, launching four forward microbatches creates a larger gap than three, thereby enabling more inter-stage communication to be overlapped. Thus, larger communication costs are compensated by proportionally larger launch counts. Let N_i denote the number of forward microbatches launched by stage i during warm-up. When deployed in a heterogeneous cluster, H-1F1B adaptively derives N_i as follows and the derived N_i can achieve the theoretical upper bound in hiding the network latency, which will be formally proved in § 4.2.

Considering a pipeline with S stages under a given parallel strategy. Let t_i denote the per-microbatch computation cost (forward plus backward) of stage i, and let c_i denote the

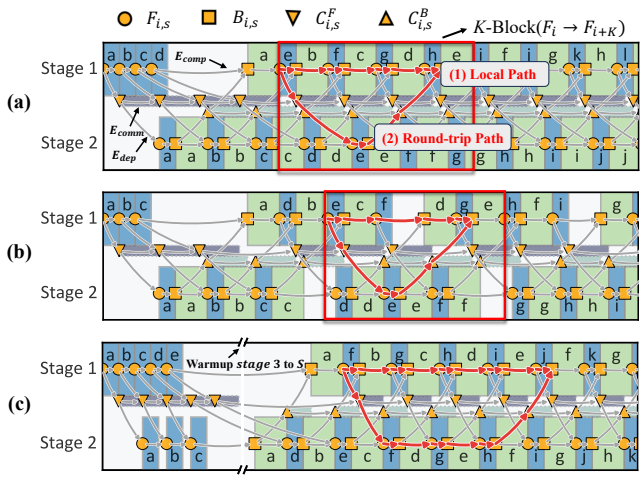

Figure 5. DAG representation of the pipeline execution. (a) Two-stage homogeneous case where the local path dominates K-block latency. (b) Two-stage homogeneous case where the round-trip path dominates K-block latency. (c) S-stage homogeneous case, reduced to an equivalent two-stage subgraph by pruning nodes & edges unrelated to stages 1–2.

per-microbatch communication cost of transferring activations or gradients between stage i and stage i+1 (with $c_S = 0$). Define $t_{\max} = \max_{i\in\{1,\dots,S\}} t_i$ as the maximum per-stage computation cost. We assume $c_i \le t_{\max}$ for all i; otherwise, full communication–computation overlap would be impossible. The last stage always launches a single microbatch ($N_S = 1$). For all $i \in \{1,\dots,S-1\}$, we define

$$N_i = 1 + \sum_{k=i}^{S-1} \delta_k, \tag{1}$$

where δ_i denotes the number of additional forward microbatches launched by stage i relative to stage i+1. The value of δ_i is determined by the ratio of c_i to $t_{\max}$:

$$\delta_i = \begin{cases} 1, & 0 < c_i \le \varepsilon \cdot t_{\max}, \\ 2, & \varepsilon \cdot t_{\max} < c_i \le t_{\max}/2, \\ 3, & t_{\max}/2 < c_i \le t_{\max} \end{cases} \tag{2}$$

where ε is a small positive constant.

4.2 Proof of Effectiveness and Generality

We formally prove that H-1F1B achieves the optimal communication–computation overlap, starting with a formal representation of pipeline execution.

4.2.1 DAG Formulation. We abstract the execution of 1F1B-style pipelines as a DAG, defined as follows.

Nodes. As shown in Figure 5, for each microbatch i at stage s, the DAG includes four types of nodes: $\boldsymbol{F_{i,s}}$ (forward compute), $\boldsymbol{B_{i,s}}$ (backward compute), $\boldsymbol{C^F_{i,s}}$ (forward communication from stage s to s+1), and $\boldsymbol{C^B_{i,s}}$ (backward communication from stage s+1 to s). We also define a *sink node* that connects to

all terminal nodes via zero-weight edges, ensuring the DAG is fully connected.

Edges. The DAG includes three categories of edges. i) $\boldsymbol{E}_{\mathbf{comp}}$: enforce sequential execution of computations within the same stage; ii) $\boldsymbol{E}_{\mathbf{comm}}$: enforce sequential transmissions in the same direction over a link (assuming full-duplex interconnects); iii) $\boldsymbol{E}_{\mathbf{dep}}$: capture the ordering between computation and communication within each microbatch.

Durations. Each edge $(u \to v)$ is weighted by the execution cost of node u, enforcing the standard finish-to-start precedence constraint:

$$s(v) - s(u) \geq d(u), \tag{3}$$

where $s(u)$ is the start time of node u and $d(u)$ its execution cost[2]. We initialize the schedule with $s(F_{1,1}) = 0$.

4.2.2 Two-stage Homogeneous Case (S=2).

We begin the proof with the base case of a two-stage pipeline where both stages have identical per-microbatch compute costs. The argument will then be extended to pipelines with S stages via mathematical induction, and finally generalized to uneven per-stage compute costs.

Let the compute costs of stage $s \in \{1, 2\}$ be $d(F_{i,s}) = f$ and $d(B_{i,s}) = b$, with symmetric communication such that $d(C^F_{i,1}) = d(C^B_{i,1}) = c$. During warm-up, stage 1 launches $K = 1 + \delta_1$ forward microbatches, while stage 2 launches only one. The steady phase dominates the overall latency of a 1F1B-style pipeline, with its duration being at least $(B - K)(f + b)$. The minimum value is attained when all inter-stage communications are fully overlapped with computation, i.e., when no pipeline bubbles remain in the steady phase. This lower bound also determines the theoretical upper bound of communication cost that can be hidden by overlap. We therefore restrict our analysis to

$$0 \leq c \leq f+b, \tag{4}$$

since full communication–computation overlap is impossible when outside this range. Given this condition, the key question becomes: *what is the minimum K that minimizes the steady-phase latency?* To answer this, we first derive an expression for the steady-phase latency as a function of K. Specifically, we begin by analyzing the temporal distance between $F_{i,1}$ and $B_{i,1}$ of the same microbatch.

Lemma 1: Precedence Bound within a Microbatch. For any microbatch i at stage 1 in the steady phase,

$$s(B_{i,1}) = s(F_{i,1}) + \max\left\{K \cdot f + (K-1)\, b,\ 2f + b + 2c\right\}. \tag{5}$$

Proof. Between $F_{i,1}$ and $B_{i,1}$, there are two possible paths, shown in Fig. 5(a). **(1) Local path.** With K forward microbatches launched in warm-up, stage 1 accumulates a backlog of K forward and $(K-1)$ backward computations ahead of $B_{i,1}$ in the steady phase. Serializing these nodes gives a path length of $Kf + (K-1)b$. **(2) Round-trip path.** To obtain the gradient required by $B_{i,1}$, microbatch i must traverse the path $F_{i,1} \to C^F_{i,1} \to F_{i,2} \to B_{i,2} \to C^B_{i,1}$, whose total length is $2f + b + 2c$. Since every path from $F_{i,1}$ to $B_{i,1}$ must satisfy the precedence constraint in Eq. (3), we obtain

$$s(B_{i,1}) \ \geq \ s(F_{i,1}) \ + \ \max\left\{K \cdot f + (K-1)b,\ 2f + b + 2c\right\}. \tag{6}$$

It remains to prove that this lower bound is tight in the steady phase. We consider two cases. **(1)** When $K \cdot f + (K-1)b > 2f + b + 2c$, the local path becomes dominant, as illustrated in Fig. 5(a). For nodes on this path: (i) $F_{i,1}$ has only E_{comp} incoming edges and therefore starts immediately after $B_{i-K,1}$ finishes. (ii) $B_{i,1}$ has incoming edges from both E_{comp} and E_{dep}, but since the local path dominates and Eq.(4), its start time is not delayed by the round-trip path. Thus, $B_{i,1}$ begins execution immediately after $F_{i+K-1,1}$ completes. In this case, all nodes meet the precedence constraint in Eq.(3) with equality, so Eq. (6) is tight. **(2)** When $K \cdot f + (K-1)b \leq 2f + b + 2c$, the round-trip path becomes the critical path, as illustrated in Fig.5(b). By a similar argument, one can show that every node on the round-trip path meets the precedence constraint in Eq.(3) with equality. Combining both cases, we conclude that Eq. (6) is tight.

Lemma 2: K-Block. In the steady phase, we define a *K-block* on stage 1 as the execution segment that begins at $F_{i,1}$ and ends at $F_{i+K,1}$. The start times of these nodes satisfy

$$s(F_{i+K,1}) \ = \ s(F_{i,1}) \ + \ \max\{\, K(f + b),\ \ 2(f + b + c)\,\}\,. \tag{7}$$

Proof. On stage 1, $F_{i+K,1}$ can only begin after $B_{i,1}$ finishes, so $s(F_{i+K,1}) - s(B_{i,1}) = b$. Substituting the bound on $s(B_{i,1})$ from Eq. (5) gives Eq. (7). Thus, Lemma 2 holds.

End-to-End Latency Analysis. For B microbatches, the steady phase can be partitioned into $\lfloor (B-K)/K \rfloor$ disjoint K-blocks, each of duration

$$\Lambda_K = \max\{K(f + b),\ 2(f + b + c)\}. \tag{8}$$

Hence, when B is sufficiently large, the end-to-end latency is approximated by

$$T(K) \ \approx \ \tfrac{B}{K} \cdot \Lambda_K \ + \ O(1) \ \approx \ B \cdot \max\left\{f + b,\ \tfrac{2(f+b+c)}{K}\right\} \tag{9}$$

where $O(1)$ denotes lower-order overhead.

To minimize $T(K)$, K must be large enough so that $2(f + b + c)/K < f + b$. Equivalently, the required number of extra forward launches can be expressed as

$$\delta_1 \ = \ K - 1 \ = \ \left\lceil 1 + \tfrac{2c}{f+b} \right\rceil, \tag{10}$$

which ensures bubble-free execution in the steady phase by fully overlapping communication with computation.

4.2.3 S-stage Homogeneous Case.

We now extend the result to a pipeline with S stages via mathematical induction. Assume that for a pipeline with $(S-1)$ stages, the extra forward launches δ_i at stage i follow the H-1F1B rule:

$$\delta_i \ = \ N_i - N_{i+1} \ = \ \left\lceil 1 + \tfrac{2c_i}{f+b} \right\rceil, \quad \forall i \in \{1, \ldots, S-1\}, \tag{11}$$

[2]If activation recomputation is used, the overhead is included in the corresponding backward compute node.

which guarantees minimum steady-phase latency and bubble-free execution. Now consider a pipeline with S stages, where stages 2 through S are scheduled according to the $(S-1)$-stage case, leaving only stage 1. The remaining question is: *what is the minimum δ_1 achieves the same guarantee?*

The key observation is that the start times of stage 3 through stage S are determined recursively by the start time of stage 2. Thus, as long as the relative start times of stage 2 nodes (with respect to $F_{1,2}$) remain unchanged, the schedules of all later stages are also unaffected. As illustrated in Fig.5(c), this allows us to prune all nodes and edges unrelated to stages 1 and 2, reducing the analysis to an equivalent two-stage subgraph. In this reduced setting, the steady-phase latency is approximated by

$$T(N_1) \;\approx\; B \cdot \max\left\{f + b,\; \frac{N_2(f+b)+f+b+2c_1}{N_1}\right\} \;+\; O(1)\,. \quad (12)$$

Minimizing $T(N_1)$ requires N_1 to satisfy Eq. (11). Thus, by induction, the H-1F1B scheduler achieves the minimum steady-phase latency and guarantees bubble-free execution for pipelines with an arbitrary number of stages.

4.2.4 S-stage Heterogeneous Case. For heterogeneous pipelines where stages have different compute times, we conceptually *stretch* faster stages so that their compute cost matches $t_{\max}$. This reduction transforms the heterogeneous pipeline into a homogeneous one with uniform stage cost, to which our earlier proof directly applies. Moreover, since our fine-grained inter-op planner (§5) produces nearly balanced stage workloads, the stretching introduces only minor deviation, ensuring that the reduction remains accurate in practice and preserving the correctness of H-1F1B's guarantees.

When stages have different compute times, the pipeline throughput is determined by the stage with the longest compute time. We conceptually "stretch" every faster stage so that its compute cost matches $t_{\max}$, thereby reducing the heterogeneous pipeline to a homogeneous one with uniform stage cost. Under this reduction, our earlier proof for the homogeneous case applies directly. Moreover, because our fine-grained inter-op planner (§5) produces nearly balanced compute costs across stages, the stretching introduces only minor adjustments, ensuring the reduction is tight in practice and preserving the correctness of H-1F1B's guarantees.

5 Fine-grained Heterogeneous Inter-op Parallel Strategy Planner

5.1 Zero-Redundant Profiler

Before planning inter-op parallel strategies, the topologically ordered operator sequence is typically clustered into coarse-grained layers to reduce the profiling and search overhead. As illustrated in Figure 6(a), existing methods simply cluster operators by compute cost, aiming to balance workload across layers [27, 37]. However, this discards the model's structural information, which limits opportunities for pruning during profiling. Consequently, planners are restricted to coarse-layer granularity to keep profiling time practical.

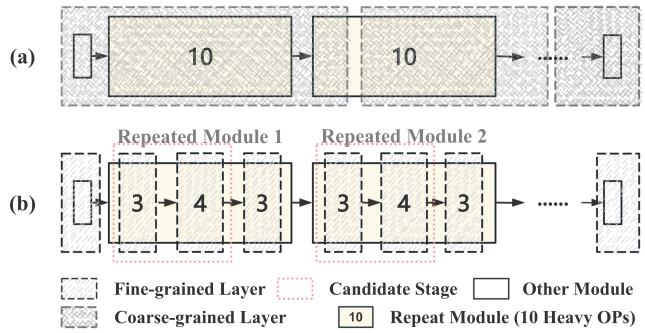

Figure 6. (a) Behaviour of layer construction in Alpa. (b) Behaviour of layer construction in Zero-Redundant Profiler.

To address this limitation, we propose the Zero-Redundant Profiler. First, it constructs a fine-grained layer sequence while preserving structural information from the model. As illustrated in Figure 6(b), this is done by partitioning the operator sequence into *repeated modules* and *non-repeated modules* through an iterative procedure: we initially treat the entire operator sequence as a single non-repeated module, and then repeatedly perform the following steps: **(i)** among all current non-repeated modules, identify the most frequent contiguous sub-sequence that contains at least z heavy operators (e.g., convolution or GEMM); **(ii)** if such a sub-sequence exists, designate it as a repeated module and continue with step (i); otherwise, terminate. For each module, we independently cluster its operators using existing algorithms [37], and then concatenate the results across modules to form a fine-grained layer sequence.

Next, once the layers are constructed, the profiler enumerates all candidate stage–mesh pairs and applies two forms of pruning. First, infeasible candidates are removed, such as those that would trigger out-of-memory (OOM) errors or whose workloads are severely imbalanced relative to the compute capacity of the submesh. Second, candidates duplicated across repeated modules are marked and pointed to their counterparts (e.g., in Figure 6(b), a candidate stage in repeated module 2 is pointed to its equivalent in repeated module 1). After pruning, we apply an off-the-shelf intra-op planner[31, 37] to determine the intra-op parallel strategy for each remaining pair, since intra-op parallelism is confined to homogeneous meshes. We then profile these candidates to obtain execution information, such as compute and communication costs, and memory footprints, which are stored for subsequent inter-op parallel strategy search.

5.2 Heterogeneous Inter-op Parallel Strategy Searching Algorithm

We now present an algorithm for searching inter-op parallel strategies to minimize end-to-end training latency on heterogeneous clusters with slow interconnects. We first model

the end-to-end latency of pipeline training under H-1F1B scheduler, then formulate the problem as a DP search to minimize this latency. To make fine-grained layer–level search tractable, we further apply several system-level optimizations, including pruning and parallelization.

5.2.1 Cost Model and DP Formulation. Assume the computational graph is clustered into a layer sequence $(l_1, l_2, \ldots, l_L)$. We partition this sequence into S stages $(s_1, s_2, \ldots, s_S)$, where each stage s_i is a contiguous subsequence $(l_q, \ldots, l_p)$. Each stage s_i is then mapped to a submesh $\text{Mesh}_{A \text{ or } B}(n_i, m_i)$, sliced from a heterogeneous cluster consisting of $\text{DeviceMesh}_A(N_A, M_A)$ and $\text{DeviceMesh}_B(N_B, M_B)$.[3] Following the same rationale as prior work [37], the size of each submesh is constrained to either $(1,1), (1,2), \ldots, (1,2^p)$ with $2^p = M$, or $(2, M), \ldots, (N, M)$. Let $t_i = t(s_i, \text{Mesh}_{A \text{ or } B}(n_i, m_i))$ denote the per-microbatch compute cost of executing s_i on its assigned mesh, and let c_i denote the communication cost between stages i and i+1, obtained from the profiler. When every c_i satisfies $c_i \le \max_{1\le j\le S} t_j$, the end-to-end training latency under the H-1F1B scheduler can be modeled as

$$T^* = \min_{\substack{s_1,\ldots,s_S;\\ (n_1,m_1)_A,\ldots,(n_S,m_S)_B}} \left\{ \sum_{i=1}^{S} (t_i + 2c_i) + (B-1) \cdot \max_{1\le j\le S} t_j \right\}, \tag{13}$$

where the first term captures the cost of a single microbatch (forward and backward through all S stages), and the second term accounts for the remaining $(B-1)$ microbatches, dominated by the slowest stage with latency $t_{\max} = \max_{1\le j\le S} t_j$.

To compute T^* in Eq. (13), we design a heterogeneous DP search algorithm. The algorithm enumerates all feasible candidate stage–mesh pairs as a fixed $t_{\max}$ and minimizes the first term in Eq. (13), Specifically, we define the DP state $F(s, k, d_A, d_B; t_{\max})$ as the minimum execution latency under fixed $t_{\max}$ when partitioning the layer subsequence $(l_k, \ldots, l_L)$ into s stages, with d_A GPUs from DeviceMesh_A and d_B GPUs from DeviceMesh_B allocated across these stages. We further define $N(s, k, d_A, d_B; t_{\max})$ as the warm-up launch count of the first stage in this s stage determined by the optimal substructure of $F(s, k, d_A, d_B; t_{\max})$. Finally, let $C(i)$ denote the communication cost of transferring the intermediate activations from $(l_1, \ldots, l_i)$ to the subsequent stages, as obtained from the profiler. The initialization is $F(0, L+1, 0, 0; t_{\max}) = 0$ and $N(0, L+1, 0, 0; t_{\max}) = 0$, and the optimal substructure of F can be expressed in two cases:

(1) When $d_A > 0$ (resources available on DeviceMesh_A),

$$F(s, k, d_A, d_B; t_{\max}) = \tag{14}$$
$$\min_{\substack{k\le i\le L\\ n\cdot m\le d_A}} \left\{ \begin{array}{l} F(s-1, i+1, d_A - n\cdot m, d_B;\ t_{\max}) + \ 2\cdot \mathrm{C}(i) \\ + t((l_k, \ldots, l_i), \text{mesh}_A(n,m), K) \end{array} \right\};$$

(2) When $d_A = 0$ (all GPUs on DeviceMesh_A allocated):

$$F(s, k, 0, d_B; t_{\max}) = \tag{15}$$
$$\min_{\substack{k\le i\le L\\ n\cdot m\le d_B}} \left\{ \begin{array}{l} F(s-1, i+1, 0, d_B - n\cdot m;\ t_{\max}) + \ 2\cdot \mathrm{C}(i) \\ + t((l_k, \ldots, l_i), \text{mesh}_B(n,m), K) \end{array} \right\}.$$

The optimal total latency under a fixed $t_{\max}$ is then

$$T(t_{\max}) = \tag{16}$$
$$\min_s \Big\{ F\big(s,\ 0,\ N_A \cdot M_A,\ N_B \cdot M_B;\ t_{\max}\big) \Big\} + (B-1)\, t_{\max}.$$

In Eq. (14), $t((l_k, \ldots, l_i), \text{mesh}_A(n,m), K)$ (abbreviated as $t(\cdot)$) denotes the compute cost of executing the candidate stage $(l_k, \ldots, l_i)$ on $\text{mesh}_A(n,m)$ with K launching counts during warmup. K is a symbolic placeholder, it can be substituted with the following expression:

$$K \ = \left\lceil \frac{2C(i)}{t_{\max}} \right\rceil + 1 + N(s-1, i+1, d_A - n\cdot m, d_B; t_{\max}), \tag{17}$$

for the case $d_A > 0$, with a symmetric definition when $d_A = 0$. For each substructure of F, the following constraints must hold: (i) $t(\cdot) < t_{\max}$, (ii) $C(i) \le t_{\max}$ (the H-1F1B communication constraint), and (iii) the H-1F1B memory constraint as followed:

$$\text{mem}_p + K \cdot \text{mem}_a \le \text{mem}_{\text{device}}, \tag{18}$$

where mem_p denote the memory for model parameters and buffers, $\text{mem}_{\text{device}}$ denote the total device memory, and mem_a denote the activation memory per microbatch.

Complexity Analysis: Assume the heterogeneous cluster consists of C homogeneous DeviceMeshes. Ignoring variations in DeviceMesh sizes, the slicing step of our DP algorithm under a fixed $t_{\max}$ requires $O\big(CL^3NM(N + \log M)\big)$ time. The number of candidate $t_{\max}$ values produced by the profiler is upper-bounded by $O\big(CL(N + \log M)\big)$. Since the profiling across different homogeneous DeviceMeshes can be parallelized, the effective complexity of this step is reduced to $O(L(N + \log M))$. Thus, the overall complexity of the search algorithm is $O\big(CL^4NM(N + \log M)^2\big)$.

5.2.2 Search Overhead Optimizations. Running the DP search at fine-grained layer granularity (e.g., $L \sim 10^2$) leads to a combinatorial explosion of the state space, requiring hundreds of hours to find an optimal strategy. To make the search tractable, we introduce three optimizations. **(1) DP sparsity index.** The theoretical state space of F scales as $O(L^2N^CM^C)$. However, the Zero-Redundant Profiler prunes a large fraction of infeasible candidate stage-mesh pairs in advance. We exploit this by pre-computing a sparse index of feasible candidate stage-mesh pairs and restricting DP transitions to this reduced set, thereby avoiding redundant states.

[3] While our planner in principle supports an arbitrary number of heterogeneous clusters, we illustrate the design of search using two clusters for clarity of exposition.

Algorithm 1: DP Search.

Input: Compute cost, communication cost and memory footprint information.
Output: Minimum end-to-end training latency T^*

$T^* \leftarrow \infty$
$[t_S, \ldots, t_E) \leftarrow \text{SortedAndFilter}(t_A, t_B)$
for $t_{\max} \in batch[t_S, \ldots, t_E)$ **do**
  $F(0, L+1, 0, 0; t_{\max}) \leftarrow 0$
  for $s \leftarrow 1$ **to** L **do**
    for $l \leftarrow L$ **to** 1 **do**
      for $d_B \leftarrow 1$ **to** $N_B M_B$ **do**
        Update $F(\cdot)$ using Eq. (17) and (15)
      for $d_A \leftarrow 1$ **to** $N_A M_A$ **do**
        Update $F(\cdot)$ using Eq. (17) and (14)
  $T_{\text{cand}} \leftarrow \min_s\{\cdots\} + (B-1)\, t_{\max}$ (Eq. (13))
  if $T_{cand} < T^*$ **then**
    $T^* \leftarrow T_{\text{cand}}$

(2) Bidirectional pruning of $t_{\max}$. Candidate $t_{\max}$ values are first sorted and deduplicated from profiler outputs. We then locate the smallest feasible value t_S via binary search and discard all smaller ones. From the solution at t_S with latency $T(t_S)$, we derive an upper bound $t_E = \lceil T(t_S)/B \rceil$. Any $t_{\max} > t_E$ is pruned, as larger values cannot further reduce latency. **(3) Batched parallel evaluation.** The remaining $t_{\max}$ candidates are evaluated in parallel using Ray actors. Instead of launching one task per candidate—which causes imbalance and JIT overhead—we batch candidates by grouping them according to the number of valid stage–mesh pairs activated. This balances the per-actor workload and amortizes warm-up costs.

6 Evaluation

We implement Hapt on top of Alpa, extending it with 3,000+ lines of Python and C++ code. Beyond extending Alpa's compiler with our planner and pipeline scheduler, we implement a heterogeneity-aware runtime on top of Ray actors [11], XLA [29], and NCCL [18], which adapts to diverse interconnects by automatically selecting the fastest available Network Interface Card (NIC) based on the network topology.

Our evaluation is conducted on the ShanHe platform [25], which provides a heterogeneous cluster with a total of 64 GPUs: 4 nodes with each equipped with 8×A100s and 4 nodes with each equipped with 8×V100s[4]. Table 2 summarizes the detailed specifications of the heterogeneous cluster. Each GPU is allocated 8 CPU cores with 8 GB of memory per core. Communication within each homogeneous subcluster is performed via RDMA over Mellanox ConnectX-6 NICs, while communication across clusters is conducted over a 10 Gbps Ethernet NIC.

Models. We evaluate Hapt on GPT models ranging from 15B to 39B parameters, scaling the hidden dimension with the number of available devices. The global batch size is fixed at 1024 and the context length at 1K.

6.1 Comparison under Heterogeneous Clusters

6.1.1 End-to-End Performance. We evaluate end-to-end training latency under eight heterogeneous configurations and three GPT model sizes (Figure 7), comparing Hapt against three representative systems. All baselines are tuned with their best hyperparameter settings, and experiments are conducted with cross-cluster bandwidth fixed at 5 Gbps[5]. Note that Megatron and HexiScale leverage FlashAttention optimizations [3, 4], whereas Alpa and Hapt rely solely on JAX's JIT compilation [5]. As a result, Hapt is at a computational disadvantage in attention-intensive blocks. These attention optimizations, however, are orthogonal to the framework-level techniques studied in this work.

As shown in Figure 7(a), neither Alpa nor Megatron supports configurations where the number of devices differs across nodes, as both are designed for homogeneous clusters with identical GPUs per node. In Figure 7(d) and (e), *Megatron* fails to identify a valid parallel strategy that fully utilizes all available devices due to its limited parallel strategy search space. Overall, despite lacking kernel-level optimizations, Hapt consistently outperforms the best baseline (HexiScale) by 1.3×–1.6× across all evaluated heterogeneous configurations, demonstrating its effectiveness in heterogeneous environments.

6.1.2 Pipeline Breakdown Analysis. Figure 8 shows the stage-wise breakdown of computation, communication, and idle time (pipeline bubbles) under the same configuration as Figure 7(h). Communication latency reflects only active data transfers on NICs, while idle waiting is attributed to bubbles. This breakdown analysis reveals the sources of Hapt's performance advantage.

Load Balance Analysis. To quantify the degree of imbalance, we define the load-balancing metric η:

$$\eta = 1 - \frac{\sum_{i=1}^{d} (td_{\max} - td_i) \cdot \text{PeakFlops}_i}{td_{\max} \cdot \sum_{i=1}^{d} \text{PeakFlops}_i} \times 100\%, \qquad (19)$$

where d is the number of devices, td_i is the actual compute time of device i, PeakFlops_i is its theoretical peak FLOP/s, and $td_{\max} = \max_{i=1}^{d} td_i$ is the maximum compute time among all devices. η ranges from $(0\%, 100\%]$, with $\eta = 100\%$ indicating perfect balance, i.e., identical compute times across all devices. We adopt η in our evaluation to measure how well different strategies balance heterogeneous workloads.

[4]To emulate V100 performance on A100s, we throttle the core frequency and cap the maximum memory allocation of the A100s.

[5]Lower-bandwidth links are emulated by using Linux trafic control (`tc`).

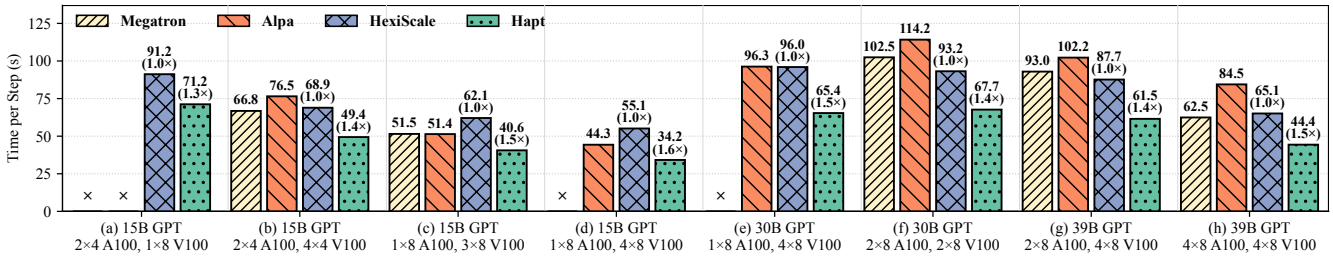

Figure 7. End-to-end training latency of Hapt compared with baselines under different heterogeneous configurations.

Table 2. Hardware specifications of the heterogeneous GPU cluster.

	Number	FP16 Tensor Core	GPU Memory	Intra-Host BW	Inter-Host BW	Cross-Cluster BW
8×A100	4	312 TFLOP/S	40 GB	300 GB/s	200 Gbps	1-10 Gbps
8×V100	4	125 TFLOP/S	32 GB	150 GB/s	200 Gbps	1-10 Gbps

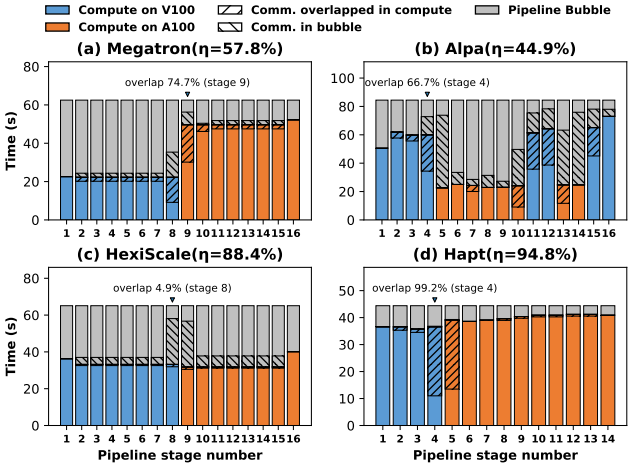

Figure 8. Stage-wise breakdown of computation, communication, and idle bubbles within one iteration for the configuration in Figure 7(h). For each framework, we annotate the communication–latency overlap ratio at the stage located on the subcluster boundary.

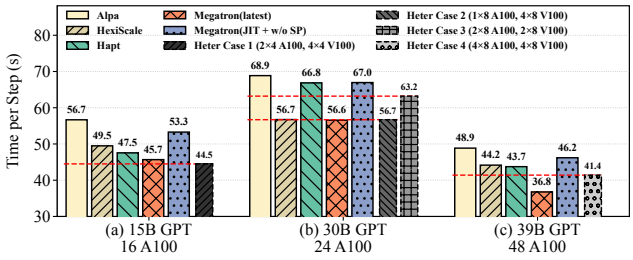

Figure 9. Comparison of end-to-end training latency across homogeneous and heterogeneous GPU clusters.

From the results, both Megatron and Alpa overlook the performance gap between A100 and V100 GPUs. For example, in Figure 8(a), stages mapped to V100s in Megatron take up to 2.6× longer than those on A100s. In contrast, in Figure 8(c) and (d), HexiScale and Hapt allocate workloads proportionally to device capabilities, thereby reducing idle time. However, HexiScale performs inter-op planning at coarse layer granularity; in this example, it aggregates the embedding and output layers into boundary stages, thereby inflating their workload and reducing load-balancing score η to 88.4%. By comparison, Hapt deliberately assigns fewer layers to slower V100s to improve the throughput of faster A100s, striking a better balance with $\eta = 94.8\%$ and thereby achieving higher end-to-end performance.

Overlap Analysis. Beyond load balancing, differences in communication–computation overlap also significantly impact performance. As shown in Figure 8(a), *Megatron* overlaps 74.7% of communication latency. This is because, under data-parallel training, where multiple `processGroups` manage different pipelines, inter-stage communication from one pipeline can overlap with computation from another. In contrast, Figure 8(c) shows that *HexiScale* overlaps only 4.9%. To support heterogeneous pipeline parallelism, it enforces strict synchronization across pipelines, thereby losing the opportunity for overlap provided by data parallelism. The small amount of overlap observed is implicitly contributed by TCP buffer effects. Figure 8(b) illustrates that *Alpa*, which uses SPMD-style data parallelism, cannot exploit data parallelism for overlap either. However, by employing Eager-1F1B scheduling, it achieves 66.7% overlap. Finally, as shown in Figure 8(d), Hapt leverages the adaptive H-1F1B scheduler. In this case, since $c < t_{\max}$, almost all communication is overlapped. Notably, unlike other frameworks, *Alpa* uses Ethernet NICs for communication across homogeneous subclusters but RDMA within each subcluster; this benefit comes from its heterogeneous runtime, which adaptively selects the appropriate network interface.

6.2 Performance Comparison on Heterogeneous and Homogeneous Clusters

We evaluate the end-to-end training latency of Megatron, Alpa, HexiScale, and Hapt on homogeneous clusters, and use these results as the baseline. We then compare Hapt's performance on heterogeneous clusters (Figure 7(b,e,f,h)) against this homogeneous baseline to assess its ability to fully exploit heterogeneous resources. Each heterogeneous cluster is chosen such that its theoretical peak FLOP/s lies within 87–93% of that of the corresponding homogeneous cluster. Latency results in Figure 9 are normalized by this ratio to ensure fairness. For completeness, we further distinguish two versions of Megatron: (i) the latest fully optimized release, and (ii) a conservative lower-bound configuration using only PyTorch JIT with sequence parallelism disabled.

As shown in Figure 9(c), we compare Hapt on a heterogeneous cluster with its homogeneous baseline. Although the heterogeneous cluster provides only 93% of the peak FLOP/s of the homogeneous one, Hapt still sustains 82.7% of Megatron's throughput. In an even more extreme configuration (Figure 9(a)), Hapt on a heterogeneous cluster even outperforms all homogeneous baselines. This advantage arises from the larger aggregate memory of the heterogeneous cluster, which allows for higher degrees of data parallelism. These results highlight that Hapt can fully exploit heterogeneous compute resources, even in the presence of slow network bottlenecks.

6.3 Sensitivity to Cross-Cluster Bandwidths

To evaluate the impact of cross-cluster bandwidth, we use the heterogeneous configuration in Figure 7(h) and throttle the cross-cluster links from 10 Gbps down to 3 Gbps using Linux `tc`. As shown in Figure 10, HexiScale is highly sensitive to bandwidth, with latency growing roughly inversely proportional to the available bandwidth, since it cannot explicitly overlap communication with computation. Megatron relies only on data parallelism and TCP buffering to implicitly overlap communication. As a result, when bandwidth falls below 7 Gbps, its trend converges with that of HexiScale. In contrast, Hapt maintains stable step time until the bandwidth drops to around 3 Gbps, at which point $c > t_{\max}$ and the inter-stage communication overhead surpasses the total computation cost. Interestingly, Alpa exhibits a similar inflection point as Hapt, suggesting that under this configuration, Eager-1F1B and H-1F1B have comparable sensitivity to cross-cluster bandwidth. However, this does not imply equivalent overlap capability. As shown in Figure 8, Alpa suffers from substantial load imbalance that doubles $t_{\max}$ compared to Hapt, which masks the limitations of its weaker overlap strategy in this experiment.

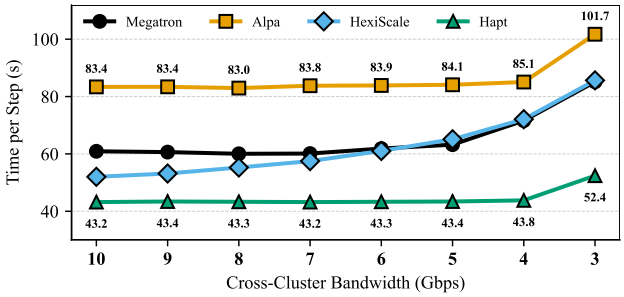

Figure 10. End-to-end training latency of different systems under varying cross-cluster bandwidths (3–10 Gbps).

6.4 Ablation Study of Layer Granularity

We examine the effectiveness of applying heterogeneous inter-op parallel strategy planning at different layer granularities. Using the heterogeneous configuration in Figure 7(h), we evaluate three granularities with Hapt's runtime: #layer-8 and #layer-16 (similar to Alpa), and #layer-48 (similar to HexiScale). As shown in Figure 11(a), strategies planned on Hapt's fine-grained layer sequence consistently outperform all baselines by 1.2×–1.6×, demonstrating the necessity of fine-grained decomposition.

6.5 Ablation Study of Joint Optimization

We evaluate the effectiveness of jointly optimizing parallel strategy planning and pipeline scheduling under the heterogeneous configuration in Figure 7(e). Specifically, we compare Hapt with a baseline that *ignores communication cost*, where parallel strategies are planned assuming zero communication overhead, i.e., by setting $C(i) = 0$ for all $i \in \{1, \ldots, s\}$. As shown in Figure 11(b), the performance of baseline is substantially worse than Hapt, resulting in a 1.4×–3.3× slowdown. This is because the size of tensor transferred across stages can vary substantially depending on the chosen inter-op parallel strategy. We inspected the inter-op parallel strategy generated by this baseline, and observed that it achieved a high load-balancing score ($\eta = 94.8\%$). However, it assigned two large tensors to the bottleneck link, incurring 5× higher communication overhead than our method. In contrast, Hapt's strategy transmits only the smaller (Batch_Size, Seq_Len, Hid_Dim) tensor over the bottleneck. Thus, even with a lower load-balancing score ($\eta = 93.3\%$), Hapt delivers markedly better end-to-end performance. These results demonstrate that explicitly accounting for communication cost is essential for planning on heterogeneous networks.

6.6 Search Overhead

We evaluate the planning overhead of a heterogeneous configuration in Figure 7 (h). Without our optimizations (Alpa's default mechanisms), profiling alone takes more than 10 hours and the DP search exceeds 100 hours, making planning impractical. In contrast, Hapt completes profiling in

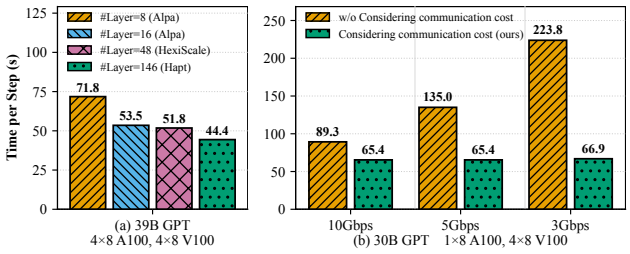

Figure 11. (a) Effect of fine layer granularity, (b) Effect of joint optimization of planning and scheduling.

1248 seconds and finishes the DP search in just 133 seconds, reducing the total overhead to about 23 minutes. We further evaluate the scalability of our planner under heterogeneous configurations, as shown in Figure 7(e–h). As the model and cluster scale up, the planning overhead actually decreases (47 min → 44 min → 33 min → 23 min). This trend arises because both GPT-30B and GPT-39B share the same layer granularity of $\#Layer = 146$. While larger clusters introduce additional profiling and searching complexity, they also increase the proportion of candidate stage–mesh pairs that can be pruned. In addition, the number of CPU cores available for parallel profiling and searching grows with cluster size, further reducing the overhead. These results demonstrate that our layer-construction method, zero-redundant profiler, and system-level optimizations for dynamic programming search make fine-grained heterogeneous planning practical and efficient for large-scale model training.

7 Conclusion

We present Hapt, an automated parallel training framework for heterogeneous clusters. Hapt introduces an efficient fine-grained parallel strategy planner that mitigates communication overhead while maintaining balanced loads across heterogeneous accelerators. With a heterogeneity-aware pipeline scheduler, Hapt achieves the theoretical upper bound of computation–communication overlap with minimal memory overhead. Our evaluation demonstrates that Hapt delivers 1.3×–1.6× higher performance than state-of-the-art frameworks designed for model training on heterogeneous clusters.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[3] Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations (ICLR)*.

[4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[5] Roy Frostig, Matthew James Johnson, and Chris Leary. 2018. Compiling machine learning programs via high-level tracing. *Systems for Machine Learning* 4, 9 (2018).

[6] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems* 32 (2019).

[7] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems* 5 (2023).

[8] Haoyang Li, Fangcheng Fu, Hao Ge, Sheng Lin, Xuanyu Wang, Jiawen Niu, Xupeng Miao, and Bin Cui. 2025. Hetu v2: A General and Scalable Deep Learning System with Hierarchical and Heterogeneous Single Program Multiple Data Annotations. *arXiv preprint arXiv:2504.20490* (2025).

[9] Shigang Li and Torsten Hoefler. 2021. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[10] Lianmin Zheng. 2022. Github repository: alpa-projects/alpa. https://github.com/alpa-projects/alpa, Last accessed on 2025-01-05.

[11] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*. 561–577.

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

[13] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2021. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*. PMLR, 7937–7947.

[14] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[15] Nvidia. 2025. Hopper Architecture. https://www.nvidia.cn/data-center/technologies/hopper-architecture/, Last accessed on 2025-09-21.

[16] Nvidia. 2025. Hopper Architecture. https://www.nvidia.cn/data-center/technologies/blackwell-architecture/, Last accessed on 2025-09-21.

[17] Nvidia. 2025. Nvida SuperNIC. https://www.nvidia.cn/networking/products/ethernet/supernic/, Last accessed on 2025-09-21.

[18] Nvidia. 2025. The nvidia collective communication library. https://github.com/openxla/xla, Last accessed on 2025-09-21.

[19] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. 2023. Zero bubble pipeline parallelism. *arXiv preprint arXiv:2401.10241* (2023).

[20] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[21] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

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

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

[24] Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. 2023. Swarm parallelism: Training large models can be surprisingly communication-efficient. In *International Conference on Machine Learning*. PMLR, 29416–29440.

[25] ShanHe Team. 2025. ShanHe SuperComputing Platform. https://www.shanhe.com/, Last accessed on 2025-09-21.

[26] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[27] Taegeon Um, Byungsoo Oh, Minyoung Kang, Woo-Yeon Lee, Goeun Kim, Dongseob Kim, Youngtaek Kim, Mohd Muzzammil, and Myeongjae Jeon. 2024. Metis: Fast Automatic Distributed Training on Heterogeneous {GPUs}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 563–578.

[28] Xiaofeng Wu, Jia Rao, and Wei Chen. 2024. ATOM: Asynchronous Training of Massive Models for Deep Learning in a Decentralized Environment. *arXiv preprint arXiv:2403.10504* (2024).

[29] XLA and TensorFlow teams. 2017. XLA — TensorFlow, compiled. https://tensorflow.google.cn/xla?hl=zh-cn#inspect_compiled_programs, Last accessed on 2024-05-07.

[30] Si Xu, Zixiao Huang, Yan Zeng, Shengen Yan, Xuefei Ning, Haolin Ye, Sipei Gu, Chunsheng Shui, Zhezheng Lin, Hao Zhang, et al. 2024. HetHub: A Heterogeneous distributed hybrid training system for large-scale models. *arXiv e-prints* (2024), arXiv–2405.

[31] Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, et al. 2021. GSPMD: general and scalable parallelization for ML computation graphs. *arXiv preprint arXiv:2105.04663* (2021).

[32] Ran Yan, Youhe Jiang, Xiaonan Nie, Fangcheng Fu, Bin Cui, and Binhang Yuan. 2024. HexiScale: Accommodating Large Language Model Training over Heterogeneous Environment. *arXiv preprint arXiv:2409.01143* (2024).

[33] Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy S Liang, Christopher Re, and Ce Zhang. 2022. Decentralized training of foundation models in heterogeneous environments. *Advances in Neural Information Processing Systems* 35 (2022), 25464–25477.

[34] Jinghui Zhang, Geng Niu, Qiangsheng Dai, Haorui Li, Zhihua Wu, Fang Dong, and Zhiang Wu. 2023. PipePar: Enabling fast DNN pipeline parallel training in heterogeneous GPU clusters. *Neurocomputing* 555 (2023), 126661.

[35] Shiwei Zhang, Lansong Diao, Chuan Wu, Zongyan Cao, Siyu Wang, and Wei Lin. 2024. HAP: SPMD DNN Training on Heterogeneous GPU Clusters with Automated Program Synthesis. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 524–541.

[36] WenZheng Zhang, Yang Hu, Jing Shi, and Xiaoying Bai. 2024. Poplar: Efficient Scaling of Distributed DNN Training on Heterogeneous GPU Clusters. *arXiv preprint arXiv:2408.12596* (2024).

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

[38] Yonghao Zhuang, Lianmin Zheng, Zhuohan Li, Eric Xing, Qirong Ho, Joseph Gonzalez, Ion Stoica, Hao Zhang, and Hexu Zhao. 2023. On optimizing the communication of model parallelism. *Proceedings of Machine Learning and Systems* 5 (2023).