HetAuto: Cross-Cluster Auto-Parallelism for Heterogeneous Distributed Training

Guicheng Qi
The University of Hong Kong
guicheng_qi@connect.hku.hk

Junwei Su*
The University of Hong Kong
junweisu.cs@gmail.com

Liqi Yang
Meituan Corporation
yangliqi02@meituan.com

Tao Li
Meituan Corporation
litao122@meituan.com

Tingwen Xie
Meituan Corporation
xietingwen@gmail.com

Yerui Sun
Meituan Corporation
sunyerui@meituan.com

Yuchen Xie
Meituan Corporation
xieyuchen@meituan.com

Chuan Wu*
The University of Hong Kong
cwu@cs.hku.hk

Abstract

As large neural network models (e.g., LLMs) grow in scale, single-cluster resources become insufficient, making cross-cluster distributed training essential. Cross-cluster training is challenging: hardware heterogeneity complicates load balancing and parallelization strategy and introduces hardware compatibility issues in implementation; cross-cluster communication bottlenecks severely impact training throughput. We present HetAuto, an automatic parallelization system for efficient cross-cluster heterogeneous large model training. HetAuto contributes three key innovations: (1) a principle-guided MCTS algorithm with a random forest-enhanced cost model that efficiently searches parallelization strategies and quickly evaluates their performance under heterogeneous configurations; (2) cross-cluster communication optimizations including Virtual-1F1B scheduling that overlaps communication with computation and an optimized resharding strategy for inter-stage communication; and (3) a unified API enabling seamless integration of diverse accelerators. We evaluate HetAuto across 4 different clusters with up to 736 heterogeneous devices. The evaluation results show that HetAuto achieves up to 1.57× training throughput improvement over representative baselines, and strikes an efficient balance between solution quality and search overhead.

CCS Concepts: • **Computing methodologies → Machine learning**.

*Corresponding authors.

EUROSYS '26, Edinburgh, Scotland Uk

ACM ISBN 979-8-4007-2212-7/26/04
https://doi.org/10.1145/3767295.3803590

Keywords: Distributed Training, Heterogeneous Clusters, Auto-Parallelism

ACM Reference Format:
Guicheng Qi, Junwei Su, Liqi Yang, Tao Li, Tingwen Xie, Yerui Sun, Yuchen Xie, and Chuan Wu. 2026. HetAuto: Cross-Cluster Auto-Parallelism for Heterogeneous Distributed Training. In *European Conference on Computer Systems (EUROSYS '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3767295.3803590

1 Introduction

The rapid advancement of large-scale models has fundamentally transformed the landscape of artificial intelligence, with models demonstrating unprecedented capabilities [14, 53, 65]. As these models continue to grow in scale and complexity, single-cluster resources often become insufficient for training [3, 45]. This insufficiency arises from two primary factors: (1) the absolute computational demands may exceed individual cluster's capacities; (2) production clusters typically serve multiple concurrent workloads, often not able to allocate the entire cluster to a single training job. In such scenarios, the alternative is often not being able to train at all rather than training on a single homogeneous cluster. Meanwhile, hardware heterogeneity arises naturally from several practical factors [9, 10, 45, 62] (drawn from both literature and our industrial experience): (a) incremental hardware upgrades — data centers progressively acquire newer accelerators (e.g., from NVIDIA Ampere [49] to Hopper [50] generations) while retaining existing ones; (b) procurement constraints — simultaneously acquiring large amounts of identical high-end GPUs is often infeasible due to supply chain limitations, driving the adoption of alternative accelerators such as AWS Trainium [18, 21]; and (c) cost optimization — mixing accelerator types with different price-performance ratios can reduce the total cost of ownership. These factors make cross-cluster heterogeneous training not merely a theoretical concern but a practical necessity.

Cross-cluster heterogeneous training presents unique opportunities and challenges. Organizations can aggregate computational resources from multiple data centers to achieve better resource utilization in learning larger capacity models. However, several fundamental challenges must be addressed to achieve efficient training.

Challenge #1: Set Complex Parallelization Strategies. The parallelization strategies to run distributed training significantly impact training performance, as suboptimal strategies often lead to substantial throughput degradation compared to optimal configurations [41, 74]. Traditionally, practitioners rely on manual tuning based on expert knowledge and empirical rules to configure hybrid parallelisms including data parallelism (DP) [22], tensor parallelism (TP) [61], pipeline parallelism (PP) [24, 47], and context parallelism (CP) [28, 42]. This manual approach suffers from several critical limitations: it requires deep domain expertise that is scarce and expensive, the tuning process is time-consuming and error-prone, and the resulting configurations are often not transferable across different model architectures or hardware setups [35, 48, 61]. In cross-cluster heterogeneous environments, manual parallelization becomes practically infeasible due to the dramatically expanded solution space. Different clusters exhibit varying computational capabilities, memory configurations, and network topologies, making expert intuition insufficient for navigating the combinatorially complex optimization landscape. Automatic parallelization systems have been proposed for single-cluster training [41, 64, 74, 76]. However, they face scalability and optimality trade-offs when applied to large-scale heterogeneous clusters. The search space grows exponentially with the model size and the number of devices and their heterogeneity, while current approaches either rely on exhaustive search methods that become computationally prohibitive [44, 64, 66, 74, 76], or use overly simplified search spaces that frequently lead to suboptimal solutions [30, 38, 41, 67, 70, 73].

Challenge #2: Counter Cross-Cluster Communication Bottlenecks. Inter-cluster communication typically suffers from higher latency and lower bandwidth compared to intra-cluster communication [62]. Unlike high-speed interconnects within clusters (e.g., InfiniBand), cross-cluster links often rely on standard network infrastructure, creating performance bottlenecks that can dominate training time. More critically, cross-cluster communication bottlenecks fundamentally complicate training parallelization strategy design. Previous homogeneous auto-parallelism systems largely rely on a key simplifying assumption: the entire computation graph can be partitioned into independent pipeline stages, with each stage optimized separately [44, 64, 74]. This assumption is valid in homogeneous clusters because inter-stage communication overhead is negligible compared to intra-stage computation and communication costs, enabling significant search space reduction since each sub-graph/stage has a much smaller solution space than the complete graph. However, this independence assumption breaks down in cross-cluster heterogeneous settings. When pipeline stages are connected via cross-cluster links, inter-stage communication incurs substantial overhead that cannot be ignored. Crucially, the intra-parallelism configuration of each stage directly affects the volume and pattern of cross-cluster communication between stages. For instance, different intra-parallelism degrees (i.e., DP, CP, TP degrees) in adjacent stages lead to different volumes of data resharding across stages. This interdependence indicates that stages can no longer be optimized independently — the auto-parallelism system must explore the complete search space of the entire computation graph, leading to exponential complexity growth with graph scale.

Challenge #3: Handle Hardware Compatibility and Unified Interface Requirements. Different clusters may well contain accelerators from various vendors, each requiring distinct software stacks. For instance, NVIDIA GPUs utilize CUDA and NCCL, while other accelerators rely on vendor-specific runtime environments and communication libraries [1, 2, 25, 26]. This hardware heterogeneity creates engineering challenges beyond simple compatibility issues. From a system design perspective, practitioners need a unified programming interface that abstracts vendor-specific details while preserving optimized performance. Without such unification, users must maintain separate codebases for each hardware type, learn multiple programming paradigms, and manually coordinate different communication protocols — an approach that is neither scalable nor practical for large-scale deployments. Achieving true hardware abstraction requires intelligent mapping of high-level operations to hardware-specific optimizations, automatic selection of appropriate communication backends, and dynamic adaptation to mixed-hardware environments. This necessitates a comprehensive framework that provides both user-friendly interfaces and efficient cross-platform execution capabilities.

To address these challenges, we present **HetAuto**, an automatic parallelization system designed for efficient cross-cluster heterogeneous distributed training of large models. HetAuto makes three key technical contributions:

▹ **Intelligent and Efficient Parallelization Strategy Search**: We develop a principle-guided Monte Carlo Tree Search (MCTS) algorithm that efficiently navigates the exponentially large search space through domain-informed pruning strategies. Our approach incorporates four key principles that dramatically reduce search complexity while preserving solution quality, and leverages MCTS to strategically balance exploration of new solution regions with exploitation of promising solutions, enabling efficient discovery of high-quality parallelization strategies without exhaustive enumeration. Additionally, we introduce a novel hybrid cost model that combines analytical modeling with machine learning

techniques. It strikes a balance between computational efficiency and prediction accuracy by using random forest models to capture hardware-specific performance characteristics, while leveraging theoretical analysis for communication overhead estimation.

▹ **Cross-Cluster Communication Optimization**: We propose Virtual-1F1B scheduling that conceptually treats cross-cluster communication as virtual pipeline stages, enabling effective overlap between communication and computation to mitigate bandwidth bottlenecks. Our approach decomposes cross-cluster transfers into device-to-host copy, cross-cluster P2P transfer, and host-to-device copy operations, seamlessly integrating with existing pipeline scheduling strategies. We further introduce an optimized resharding strategy that minimizes inter-stage communication overhead through intelligent tensor layout transformation, balancing network utilization and memory efficiency in heterogeneous cluster environments.

▹ **Unified Hardware Abstraction**: We implement an abstraction layer that enables seamless integration and coordination of diverse accelerators by providing automatic hardware detection and operation routing. Our abstraction layer sits above the existing vendor software stacks, automatically selecting appropriate backends while maintaining unified PyTorch APIs. This approach enables practitioners to deploy the same training code across heterogeneous hardware configurations without manual backend specification or conditional compilation.

We evaluate HetAuto across four heterogeneous clusters with up to 736 devices, demonstrating up to 1.57× training throughput improvements and low strategy search overhead. Our extensive experiments confirm the satisfactory prediction accuracy of our random forest-enhanced cost model, with one-time profiling overhead of 8-20 minutes per cluster. We show that Virtual-1F1B scheduling achieves 1.68-1.84× improvements in training iteration latency compared to standard 1F1B scheduling, and our optimized resharding strategy further reduces communication overhead. HetAuto source code is available at https://github.com/Guicheng-Qi/HetAuto.

2 Background and Motivation

2.1 Distributed Training Parallelization Strategies

Modern large-scale model training relies on sophisticated combinations of parallelization strategies to achieve scalability and efficiency. Data parallelism [22] replicates the model across devices, each processing a different data batch. Tensor parallelism [61] partitions individual operations across devices for fine-grained parallel computation. Pipeline parallelism [24, 47] divides models into sequential stages across devices for pipelined processing. Context parallelism [28, 42] distributes long sequences among devices along the sequence dimension. Manual configuration of these strategies requires deep expertise and extensive trial-and-error experimentation. Practitioners must navigate complex trade-offs between computation efficiency, memory utilization, and communication overhead while considering model architecture specifics and hardware characteristics.

2.2 The Complexity of Cross-Cluster Heterogeneous Training

Cross-cluster heterogeneous training involves distributing model training across multiple separated clusters with potentially different hardware configurations. We target scenarios where multiple distributed clusters are connected with limited inter-cluster bandwidth, with each cluster being homogeneous internally (hosting devices of a single hardware type) while the clusters may use different device types, varying numbers of devices, and distinct software stacks.

2.2.1 Exponential Search Space in Heterogeneous Environments.

The search space for optimal parallelization strategies of training a large model over heterogeneous clusters exhibits exponential complexity that stems from multiple dependent combinatorial decisions.

Graph Partitioning Complexity. For pipeline parallelism, the system must partition the model computational graph into pipeline stages. For a computational graph with O operators, the number of possible ways to partition these operators into p pipeline stages is $\binom{O-1}{p-1}$. Even with domain-specific simplifications (such as restricting partitions to layer boundaries in transformer models), the combinatorial explosion remains substantial for large models (which is often the case for cross-cluster training).

Device Mesh Allocation Complexity. Each pipeline stage requires allocation of device meshes from available clusters. A device mesh is a logical 2D arrangement of devices where devices share uniform compute capabilities within the mesh and consistent communication performance along each dimension. In homogeneous settings, this involves selecting the device mesh shape from a single device pool. In heterogeneous cross-cluster environments, the complexity explodes due to two factors: (1) Device type combinations: each stage can utilize devices from different clusters; (2) Device mesh shape selection: for each device type, the system must determine the corresponding device mesh shape. With C heterogeneous clusters, each stage can choose from $O(2^C)$ possible device type combinations, with mesh shape selection for each device type following an established $N + \log M$ complexity [74], where N represents the maximum number of nodes and M represents the maximum number of devices per node. This complexity arises from the preference hierarchy of device allocation: within a single node, devices are allocated in power-of-two increments $(1, 2, 4, \ldots, M)$; expanding to multiple nodes adds N options, yielding configurations like $(1, 1), (1, 2), \ldots, (1, M), (2, M), \ldots, (N, M)$.

Intra-Parallelism Strategy Selection. Within each allocated device mesh for each pipeline stage, the system must determine optimal data parallelism (DP), tensor parallelism

(TP), and context parallelism (CP) degrees. The number of valid parallelism combinations for a mesh of size $N \times M$ is $d_3(N \times M)$, where $d_3(n)$ denotes the number of ways to write n as an ordered product of three positive integers, corresponding to DP, CP, TP degrees.

Compounded Exponential Growth. The above three sets of decisions combine multiplicatively, creating a search space of exponential complexity. For example, training an 80-layer Llama model across three clusters (256+256+512 devices) results in over 10^{110} possible parallelism configurations, making exhaustive exploration computationally impossible (see Appendix A for detailed analysis). Existing auto-parallelism approaches like Alpa [74] already require hours for computing parallelization strategies of a 32-device homogeneous cluster. The heterogeneous cross-cluster setting increases this complexity by orders of magnitude, rendering current approaches impractical for realistic deployments.

2.2.2 Communication Hierarchy and Pipeline Stage Interdependence.

Cross-cluster training introduces a fundamental two-tier communication hierarchy that breaks key assumptions of existing auto-parallelism methods. Within clusters, devices communicate via high-bandwidth interconnects such as NVLink (e.g., 300-600 GB/s for NVLink 3.0/4.0) and InfiniBand (e.g., 100-400 Gb/s). Between clusters, communication relies on standard networking infrastructure with significantly lower bandwidth (typically 10-100 Gb/s) and higher latency. This hierarchy invalidates the stage independence assumption that enables scalable optimization in homogeneous settings. Previous homogeneous-cluster auto-parallelism approaches [44, 64, 74] decompose the problem by assuming pipeline stages can be optimized independently, reducing search complexity from $O(S^p)$ to $O(S \cdot p)$ where S is the size of the strategy space per stage and p is the number of stages. However, when pipeline stages span clusters, their parallelism configurations become tightly coupled through cross-cluster communication requirements. For example, if stage i uses TP degree 8 while stage $i+1$ uses TP degree 2 and the two stages are on two different clusters, the system must perform expensive tensor resharding across the low-bandwidth inter-cluster link. The communication volume and pattern depend critically on both stages' configurations, creating interdependencies that force exploration of the full $O(S^p)$ search space.

2.2.3 Hardware Diversity and Integration Challenges.

Heterogeneous clusters present integration challenges that extend beyond simple performance differences. Each accelerator type requires distinct software stacks: NVIDIA GPUs use CUDA kernels and NCCL for collective communication [51], while other accelerators require vendor-specific runtime environments and communication libraries [1, 2, 25, 26]. These software stack differences create development and maintenance complexities. While different hardware types can communicate through host-based backends like Gloo [17], practitioners currently face the burden of maintaining separate codebases for each hardware type. This includes writing hardware-specific operator implementations, managing different memory allocation patterns, and manually coordinating communication backends. Without unified abstraction, supporting H hardware types requires practitioners to maintain H separate codebases, each with hardware-specific optimizations and communication logic. This fragmented development approach increases maintenance overhead and hinders the adoption of cross-cluster heterogeneous training.

2.3 Limitations of Existing Auto-Parallelism Approaches

While auto-parallelism has been useful in distributed training, existing approaches face fundamental scalability and optimality trade-offs that become prohibitive in cross-cluster heterogeneous environments.

Exhaustive Search Lacks Scalability. Methods like Alpa [74], Piper [64], Galvatron [44], Mist [76], and Metis [66] employ dynamic programming or exhaustive enumeration to find optimal parallelism solutions. While they can guarantee optimality in small-scale settings, they become computationally intractable when facing the exponential search spaces of cross-cluster heterogeneous environments. Even with optimization techniques, these methods require hours for modest cluster sizes and cannot scale to the 10^{110}-scale search spaces we identified (see Appendix A).

Simplified Search Space Leads to Suboptimal Solutions. To achieve scalability, many existing works [30, 38, 41, 70, 73] dramatically simplify the search space through restrictive assumptions or heuristics, such as limiting parallelism combinations to predefined templates or assuming stage independence. While computationally tractable, these simplifications exclude potentially superior configurations, leading to suboptimal performance in complex heterogeneous environments where the optimal solution may require unconventional parallelism combinations. For instance, a common simplification is to enforce that all pipeline stages adopt identical intra-parallelism strategies to avoid combinatorial explosion [38, 70]. However, this assumption is fundamentally flawed in heterogeneous settings: different accelerator types have distinct memory hierarchies, computational capabilities, and communication topologies, making their optimal intra-parallelism strategies inherently different even for the same training workload.

In Table 1, we evaluate 16-device training of a 24-layer Llama model (hidden size 4096, 32 attention heads, sequence length 8192, global batch size 64) by enumerating intra-parallelism degrees (DP d, CP c, TP t) and micro-batch sizes (b) to maximize throughput (TDS: tokens per device per second). Table 1 summarizes the optimal strategies across different hardware configurations. For instance, on 16 NVIDIA

Table 1. Optimal intra-parallelism strategies

Device (Count)	Optimal (d,c,t), b	Thpt. (TDS)
A100-80GB (2 × 8)	(8, 2, 1), 1	3395.6
H20-141GB (2 × 8)	(16, 1, 1), 1	2218.6
Ascend A2-64GB (1 × 16)	(4, 1, 4), 2	2286.7

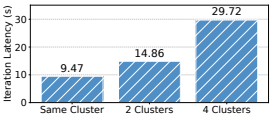


Figure 1. Iteration latency across deployment scenarios.

A100-80GB GPUs (2 nodes × 8 GPUs/node), the optimal strategy is $(d, c, t) = (8, 2, 1)$ with micro-batch size 1. In contrast, 16 Ascend A2-64GB NPUs (1 node × 16 devices/node) achieve optimal throughput at $(d, c, t) = (4, 1, 4)$ with micro-batch size 2. These results confirm that optimal intra-parallelism configurations vary across device types.

Inadequate Cross-Cluster Communication Optimization. Even works that consider heterogeneous clusters [30, 38, 63, 66, 70] typically treat cross-cluster communication as a fixed overhead rather than an optimization target. They lack specialized scheduling mechanisms and resharding strategies designed for the unique characteristics of multi-tier communication hierarchies. In practice, cross-cluster communication can incur substantial overhead and significantly prolong training iteration time. We demonstrate this phenomenon in Fig. 1, where a 24-layer Llama model (hidden size 4096, 32 attention heads, sequence length 8192, global batch size 64) is trained using 32 NVIDIA A100-80GB GPUs with a parallelism strategy of $(p, d, c, t) = (4, 2, 2, 2)$ (p is the pipeline parallelism degree) and standard 1F1B pipeline scheduling. We evaluate three deployment scenarios: (1) all 4 nodes (8 GPUs each) within a single cluster, (2) 2 nodes in one cluster and 2 nodes in another cluster, and (3) each node in a different cluster. Intra-cluster connectivity uses 100 Gbps InfiniBand, while inter-cluster bandwidth is 10.2 Gbps as measured using `iperf3` [16]. As shown in Fig. 1, cross-cluster communication introduces substantial overhead. This degradation stems not only from reduced inter-cluster bandwidth but also from the additional communication pipeline: each tensor must undergo device-to-host (D2H) copy, cross-cluster peer-to-peer transfer, and host-to-device (H2D) copy, where host-device transfers incur considerable latency.

2.4 Our Approach: Practical Optimization for Cross-Cluster Training

Our approach is motivated by the observation that while cross-cluster heterogeneous training presents exponential search complexity in identifying optimized parallelization strategies, this complexity contains exploitable structure that enables practical optimization through three key insights.

Domain-Knowledge Guided Search Space Pruning. Rather than exploring the full exponential search space, we can leverage domain knowledge of distributed training patterns to eliminate configurations that are fundamentally unlikely to yield good solutions (§ 4.1). For instance, configurations that create excessive cross-cluster communication or cause obvious computational inefficiencies (such as extreme load imbalance across stages) can be pruned early, dramatically reducing the effective search space while preserving promising solution regions.

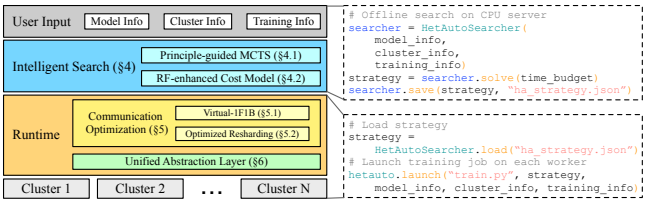


Figure 2. System overview of HetAuto.

Intelligent Search Over Exhaustive Enumeration. For the pruned search space, we employ Monte Carlo Tree Search (MCTS) guided by a hybrid cost model to efficiently explore the solution space and converge to high-quality configurations (§ 4.1). Our cost model combines machine learning techniques for hardware-specific performance modeling with analytical approaches for communication overhead estimation, providing accurate yet computationally efficient evaluations (§ 4.2). This approach balances solution quality with computational tractability, allowing practical optimization for realistic large-scale deployment scenarios.

Cross-Cluster Communication-Aware Optimization with Unified Execution. We address the fundamental bottleneck of cross-cluster communication through specialized scheduling mechanisms and resharding strategies tailored to multi-tier communication hierarchies (§ 5). By understanding the communication patterns inherent in different parallelism configurations, we can reduce cross-cluster overhead through optimized data movement and overlapping. Additionally, we provide a unified execution abstraction that enables seamless integration of diverse accelerator types through automatic hardware detection and operation routing (§ 6). Our abstraction layer sits above existing vendor software stacks, automatically selecting appropriate backends while maintaining unified PyTorch APIs, thereby eliminating the need for manual cross-hardware integration efforts.

3 HetAuto Overview

Fig. 2 gives a system overview of HetAuto, which comprises an intelligent search module that runs offline on a single CPU server to generate optimal parallelization strategies, and an optimized runtime system that executes across multiple heterogeneous clusters for distributed training.

The workflow of HetAuto proceeds as follows. Users provide: (1) model information including model architecture, number of layers, hidden size, attention heads, and sequence length; (2) cluster information including hardware specifications, number of nodes, devices per node, intra-cluster bandwidth, and inter-cluster bandwidth; and (3) training configuration such as global batch size, recomputation level (full or selective recomputation of activations or no recomputation), and pipeline scheduling method (e.g., 1F1B, interleaved-1F1B [48], zero bubble [57], or Virtual-1F1B to be introduced later).

Then the intelligent search module of HetAuto identifies an optimized parallelization strategy across the heterogeneous cluster (§ 4). This module incorporates a principle-guided MCTS algorithm that automatically and efficiently discovers high-quality solutions (§ 4.1), along with a random forest-enhanced cost model that enables fast and accurate performance evaluation (§ 4.2). Subsequently, the discovered parallelization strategy is applied to HetAuto runtime system to launch the distributed training job accordingly.

The runtime is specifically optimized for cross-cluster heterogeneous training. It supports Virtual-1F1B scheduling to overlap cross-cluster communication with computation (§ 5.1), and optimized inter-stage resharding to balance network utilization and memory efficiency (§ 5.2). Additionally, the runtime exploits a unified abstraction layer to detect hardware type and automatically route operations to corresponding implementations, thereby hiding vendor-specific implementation details across different hardware platforms (§ 6). The main APIs implementing this workflow are illustrated in Fig. 2.

4 Intelligent Parallelism Strategy Search

The Parallelism Strategy Search Problem. We consider the cross-cluster auto-parallelism problem as a two-level nested optimization, similar to the inter- and intra-parallelism framework from Alpa [74]. At the outer level, we determine the pipeline parallelism structure by deciding the number of pipeline stages p, partitioning the model into these stages, and selecting the number of micro-batches q per training iteration. At the inner level, we allocate device resources from each cluster to pipeline stages and determine the optimal intra-parallelism strategy (data, context, and tensor parallelism degrees) in each allocation. The goal is to minimize the training iteration time (equivalently, maximize training throughput), subject to device capacity constraints, memory limitations, and so on (detailed optimization formulation provided in Appendix B).

Search Space Complexity. This optimization problem constitutes a complex integer non-linear programming challenge that becomes computationally intractable in large-scale scenarios. As analyzed in § 2.2, cross-cluster training scenarios require global optimization because adjacent stages' parallelism strategies significantly affect cross-cluster communication costs which are non-negligible, transforming search complexity from additive (sum over stage search spaces) to multiplicative (product over stage search spaces).

4.1 Principle-Guided MCTS

We address this computational challenge through a principle-guided Monte Carlo Tree Search framework. Our approach begins by establishing domain-informed principles that effectively reduce the exponential search space based on theoretical insights and empirical observations. We then leverage MCTS [6, 78] to navigate the pruned search space and discover high-quality parallelization strategies.

Principle #1: Partition the model at the granularity of Transformer layers. Given that transformer-based models [4, 14, 53, 65, 68] have been dominant in modern AI development, we focus on cross-cluster auto-parallelism for transformer-based large model training. The repetitive layer structure of transformer architectures provides natural partitioning boundaries: we partition at the layer level, ensuring that each pipeline stage contains an integer number of complete transformer layers. This design choice is motivated by the communication characteristics of transformer architectures: intra-layer operations between consecutive operators (e.g., GeMM, GeMV) require significantly more intermediate data transfer than inter-layer communication, as evidenced by prior work on pipeline parallelism optimization as well [24, 47, 61]. Consequently, partitioning individual transformer layers across different pipeline stages would be even more counterproductive in bandwidth-constrained cross-cluster training scenarios.

Principle #2: Restrict cross-cluster parallelism to pipeline parallelism only. In other words, we must use the same type of devices within a pipeline stage. This principle is grounded in both communication efficiency and empirical evidence. Context parallelism and tensor parallelism require significantly more frequent inter-device communication than data parallelism and pipeline parallelism, making them impractical for cross-cluster deployment where bandwidth is much more limited. Prior work has demonstrated that pipeline parallelism achieves superior efficiency compared to data parallelism in bandwidth-constrained cross-cluster training scenarios [8]. That is, cross-cluster pipeline parallelism consistently outperforms cross-cluster data parallelism (i.e., each cluster has a model replica and cross-cluster all-reduce is needed for gradient synchronization) under various inter-cluster bandwidth configurations. This is because pipeline parallelism's sequential communication pattern better tolerates high latency (through overlapping with stage computation), while data parallelism's cross-cluster all-reduce operation renders a blocking performance bottleneck.

Principle #3: Minimize inter-cluster communication frequency. Inter-cluster communication typically constitutes the primary bottleneck in cross-cluster training, directly impacting iteration latency, as demonstrated in Fig. 1. Minimizing the need of cross-cluster communication is essential for improving training efficiency. A direct corollary of this principle is that each cluster should accommodate consecutive pipeline stages rather than discrete, non-contiguous stages. A non-consecutive stage allocation that interleaves stages across clusters can result in excessive inter-cluster communication.

Principle #4: Enforce uniform intra-parallelism strategies within each cluster. Within the same cluster, all pipeline

stages should adopt identical intra-parallelism strategies. This principle is motivated by a balanced trade-off between empirically validated near-optimal performance and search efficiency: *First*, for transformer-based models characterized by highly repetitive layer structures, uniform intra-parallelism within homogeneous clusters has been empirically proven to deliver near-optimal performance [35, 48, 61]. This is because: (1) pipeline stages in the same cluster process isomorphic transformer layers, so a shared strategy aligns with the model's structural regularity; (2) identical configurations ensure consistent tensor layouts across stages, enabling seamless data transfer without resharding, thus reducing communication overhead that would otherwise degrade performance. *Second*, this uniformity drastically prunes the search space of valid parallelism configurations. Without this constraint, the number of possible intra-parallelism combinations scales linearly with the number of pipeline stages, leading to exponential growth in the solution space and prohibitive search latency. Enforcing uniformity reduces the search complexity from stage-specific optimization to cluster-specific optimization, significantly accelerating the search process.

The four principles above can significantly reduce the search space of the original nested optimization. However, this pruned search space remains computationally intractable for large-scale systems involving high-capacity clusters with thousands of devices and increasing model layers. We employ Monte Carlo Tree Search (MCTS), a powerful heuristic approach that excels at navigating large discrete search spaces. MCTS strategically balances exploration of new solution regions and exploitation of known promising solutions. We adopt MCTS not as a generic black-box, but as a domain-specialized planner over the pruned space defined by our four principles, with the following key designs.

Problem-Specific Decision Tree Structure. We formulate the pruned optimization problem as a sequential decision process represented by a decision tree (Fig. 3), which has a natural causal order that mirrors the two-level optimization structure and enables early pruning. The tree structure captures our hierarchical decision making: starting from the root node, we determine pipeline stages p, followed by selecting the number of micro-batches q. Next, we establish a cluster permutation σ for C clusters, where C is the total number of available clusters, and the permutation σ determines the order for sequential decision making. Sequentially for each cluster i in the permutation order σ, we decide: (1) the number of stages to allocate to cluster i, (2) the layer distribution across stages within cluster i, and (3) the intra-parallelism strategy for stages in cluster i. The arrow at the bottom of Fig. 3 indicates this iterative process: after completing decisions for cluster i (with OOM check for early pruning), we proceed to make the same three types of decisions for the next cluster $i+1$ in the permutation order,

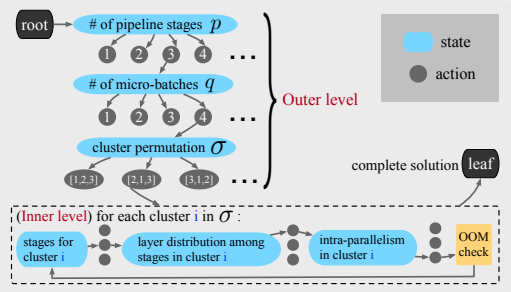


Figure 3. Decision tree representation of the pruned optimization for MCTS-based parallelization strategy search.

continuing until all C clusters have been processed. Once decisions are made for all clusters, we obtain a complete parallelization strategy. This decision tree structure defines the search space: each root-to-leaf path represents one complete parallelization strategy. Rather than exhaustively enumerating all possible paths (which would be computationally prohibitive), MCTS selectively explores promising branches using the UCB formula to guide tree traversal. The algorithm iteratively builds and refines this decision tree by strategically expanding nodes that are likely to lead to better solutions, ultimately converging on high-quality parallelization strategies.

MCTS Algorithm. We employ the MCTS framework (Algorithm 1) with four iterative phases: (1) Selection: Starting from the root, we traverse the tree by selecting the most promising child nodes according to the UCB formula until reaching an expandable node (i.e., a non-leaf node with untried actions). (2) Expansion: We expand the selected node by randomly choosing an untried action, which creates a new child node. (3) Simulation: We perform a random rollout from the newly expanded node to a complete solution and evaluate its performance using our cost model to obtain a reward signal. (4) Backpropagation: We propagate the obtained reward back through all ancestor nodes in the decision path, updating their accumulated reward and visit counts. The Upper Confidence Bound (UCB) score [6, 78] is computed by:

$$\mathrm{UCB}(s,a) = \underbrace{\frac{Q(s,a)}{N(s,a)}}_{\substack{\text{exploitation}\\ \text{term}}} + \underbrace{\lambda \cdot \sqrt{\frac{\ln(N(s))}{N(s,a)}}}_{\substack{\text{exploration}\\ \text{term}}} \tag{1}$$

where $Q(s,a)$ is the total reward accumulated so far when taking action a in state s, $N(s,a)$ is the number of times action a is selected in state s, $N(s)$ is the total number of visits to state s, and λ is the exploration weight. The state encodes a prefix of decisions, and the actions at each node are feasible extensions of that prefix. For example, a state s can represent "p=4 pipeline stages, q=8 micro-batches, cluster permutation [2, 1, 3]", and an action a is "allocate 2 stages to cluster 2". UCB effectively balances two competing objectives: the exploitation term favors actions with historically high rewards,

Algorithm 1 MCTS for Pruned Cross-Cluster Auto-Parallelism Optimization

Input: Model information I_{model}, cluster information $I_{cluster}$, training information $I_{training}$, time budget T_{budget}
Output: Best-found parallelization strategy $\mathcal{P}^*$ and corresponding training iteration latency ℓ^*

```
root ← CreateRoot(I_model, I_cluster, I_training)   ▷ Empty state
ℓ* ← +∞, P* ← ∅, start_time ← CurrentTime()
while CurrentTime() − start_time < T_budget do
    node ← Select(root)                              ▷ UCB-based selection
    if IsExpandable(node) then
        node ← Expand(node)                          ▷ Add new child node
    end if
    reward ← Simulate(node)                          ▷ Random rollout to leaf
    Backpropagate(node, reward)                      ▷ Update ancestors
    if IsLeaf(node) and node.latency < ℓ* then
        ℓ* ← node.latency
        P* ← node.strategy
    end if
end while
return ℓ*, P*
```

while the exploration term encourages the investigation of less-visited actions that may lead to better solutions.

Custom Reward and Constraint Handling. During the simulation phase, we employ our latency cost model to compute the training iteration latency ℓ for each complete parallelization strategy. The reward is then calculated as $\frac{1}{1+\ell}$ (lower latency yields higher reward). Additionally, we use our memory cost model to estimate device memory consumption in each cluster when the decisions for current cluster are made. If any cluster encounters OOM issues, we immediately assign a reward of 0 and terminate the simulation early, ensuring that infeasible solutions are properly removed.

Constraint-Aware Action Space Pruning. At each decision node, we dynamically prune the action space based on problem constraints. For instance, when deciding stage allocation to a cluster, we only consider allocations that respect remaining pipeline stages and device availability, reducing branching factor and accelerating search convergence.

Rollout Variance Mitigation. While random rollouts during simulation can introduce variance in reward estimates, several aspects of our design mitigate this concern: (1) UCB's exploration term (Eqn. (1)) naturally accounts for uncertainty by favoring less-visited branches, thus accumulating more samples where estimates are noisy; (2) constraint-aware action space pruning drastically reduces the branching factor, making random rollouts more likely to reach reasonable complete solutions.

Theoretically, MCTS converges to exhaustive search given infinite time, guaranteeing increasingly better solutions with larger time budgets [34]. In practice, users can configure the time budget based on their problem complexity and performance requirements, trading off search time and solution quality.

4.2 Random Forest-Enhanced Cost Model

To guide the MCTS search process, we need accurate and efficient cost models to evaluate the performance of specific parallelism strategies. Purely theoretical modeling struggles with accuracy because they cannot capture the complex relationships between parallelism strategies and actual hardware performance. Modern accelerators exhibit intricate behaviors — memory hierarchy effects, kernel fusion optimizations — that are difficult to model analytically [15, 72, 75]. This is particularly problematic for core computation modules like transformer layers, where the relationship between intra-parallelism configurations and execution latency exhibits strong non-linearity due to hardware-specific optimizations and resource contention patterns. On the other hand, pure profiling can achieve high accuracy but incurs prohibitive overhead. Exhaustively profiling all possible configurations across multiple device types would require an unacceptable number of measurements, making cost model construction itself a bottleneck.

We address these limitations through a hybrid approach that combines profiling and analytical modeling, based on the characteristics of different system components. Our key insight is that for auto-parallelism, we do not need perfect absolute accuracy, but sufficient relative accuracy to distinguish good solutions from poor ones in the search process. We identify that different components exhibit different complexity patterns: (1) Core computational modules (e.g., transformer layers): Non-linear relationships between parallelism strategies and performance, requiring profiling for accuracy; (2) Communication operations: Linear relationships with communication volume and bandwidth, suitable for analytical modeling; (3) Auxiliary modules (e.g., embedding layers): Minimal performance impact, not requiring profiling.

Profiling-based single-layer latency prediction. We extract the core module of an entire model (e.g., a single transformer layer) and profile the actual forward execution latency of the single layer under representative intra-parallelism strategies and input tensor shapes. The backward pass latency is estimated based on the forward pass latency, accounting for potential activation recomputation. Especially, we vary context parallelism (CP) and tensor parallelism (TP) degrees during profiling. Data parallelism primarily incurs communication overhead via all-reduce operations in the backward pass; we estimate the corresponding latency using theoretical modeling. This design reduces profiling overhead by limiting the total number of strategy combinations that need to be measured. The profiling is hardware-dependent rather than job-dependent, so we profile once for each device type. The profiled data can then be reused across different training jobs that contain the same core module.

To predict the actual latency for unseen parallelism[1] and input shape configurations, we use the profiled data to train a random forest model [5, 7, 40]. The random forest model is an ensemble learning method that combines multiple decision trees to achieve robust and accurate predictions. As input to the model, we use basic features (e.g., tensor shapes, parallelism degrees) and domain-specific informative features that capture key performance characteristics, including computational intensity, memory requirements, communication overhead, and their trade-offs. Representative basic and derived features are given in Appendix C.1.

Hybrid latency modeling. The trained random forest model predicts the latency for a single layer. For other communication overheads (e.g., all-reduce in data parallelism for gradient synchronization, P2P communication between pipeline stages) and other modules (e.g., embedding layer), we use theoretical modeling to estimate the latency. Communication latency is modeled as startup_latency + message_size / bandwidth, following the standard alpha-beta communication model [8]. For embedding layers, being memory-intensive operations, latency is estimated based on memory access volume and memory bandwidth.

Memory usage modeling. The memory cost can be more readily estimated. We follow previous work [37, 41, 66, 74] to build our memory cost model through theoretical analysis (detailed in Appendix C.2).

5 Cross-Cluster Communication Optimization

Cross-cluster communication must traverse heterogeneous network infrastructure with significantly lower bandwidth and higher latency, presenting two challenges in distributed training with hybrid parallelisms. *First*, the synchronization point in conventional pipeline scheduling leads to severe bubble time when pipeline stages span clusters, as slow cross-cluster transfers block the entire pipeline. *Second*, different clusters may adopt heterogeneous parallelism configurations, requiring tensor resharding operations, that further amplify communication overhead across cluster boundaries. We propose two communication optimizations, Virtual-1F1B and optimized resharding, enabling efficient pipeline parallelism across distributed clusters.

5.1 Virtual-1F1B

1F1B scheduling [19, 47] is the most widely used pipeline parallel training strategy today. In conventional 1F1B, each pipeline stage alternates between forward and backward passes, with synchronous P2P communication between adjacent stages. However, when pipeline stages span across clusters, cross-cluster communication becomes a bottleneck, which must traverse multiple network layers including device-to-host transfer, inter-cluster networking, and host-to-device transfer. As shown in Fig. 4 (1F1B), the communication between stage 2 and stage 3 would take longer time than intra-cluster communication, creating pipeline bubbles that substantially increase training iteration latency.

We propose Virtual-1F1B, an optimized scheduling approach to address the communication bottleneck in cross-cluster training. As illustrated in Fig. 4, Virtual-1F1B treats cross-cluster communication as a virtual pipeline stage and enables better overlap with computation. The key insight is to decompose cross-cluster communication into three sequential operations that naturally map to pipeline stage semantics: (1) Device-to-host (D2H) copy: Analogous to the 'send' operation of a pipeline stage; (2) Cross-cluster P2P transfer: Analogous to the 'computation' (forward/backward) of a pipeline stage; (3) Host-to-device (H2D) copy: Analogous to the 'receive' operation of a pipeline stage. We enable the cross-cluster P2P transfer (the most time-consuming component in a virtual stage) to overlap with forward/backward computation in other pipeline stages, effectively hiding communication latency behind computation. It is important to distinguish Virtual-1F1B from standard intra-cluster communication–computation overlap using multiple CUDA streams [52]. Within a cluster, devices communicate via high-speed interconnects (e.g., NVLink or InfiniBand), and overlapping is achieved by issuing communication and computation kernels on separate CUDA streams within the same device memory domain. Cross-cluster communication, however, involves a fundamentally different three-phase data path: device-to-host (D2H) memory copy, host-to-host network transfer across cluster boundaries, and host-to-device (H2D) memory copy at the destination. Standard CUDA stream overlap cannot address this multi-phase, multi-memory-domain transfer. Virtual-1F1B's contribution is abstracting this three-phase transfer as a virtual pipeline stage, which naturally integrates with pipeline scheduling semantics — for example, the number of warmup micro-batches increases by the number of cross-cluster boundaries.

We design asynchronous P2P communication operations for cross-cluster communication, enabling Virtual-1F1B. In distributed computing, an asynchronous operation returns a handle — a reference object that can be used later to wait for operation completion and retrieve results. We manage these handles using queues at both sender and receiver sides. Each cross-cluster asynchronous P2P operation involves two steps: (1) Wait for current operation to complete: Retrieve the handle from the head of the queue and wait for the ongoing operation to complete (sender waits for send completion, receiver waits for complete tensor reception); (2) Initiate next operation: Both sender and receiver launch the next asynchronous cross-cluster communication and enqueue its handle at the tail of their respective queues. This handle will be used for future synchronization when the operation

[1] For example, on NVIDIA A100 GPUs, we profile with at most 2 nodes or 16 GPUs, so configurations like CP=4, TP=8 that require more devices become unseen parallelism strategies.

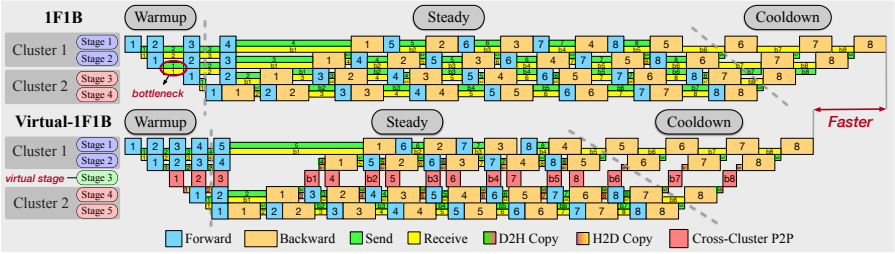


Figure 4. Comparison between 1F1B and Virtual-1F1B scheduling.

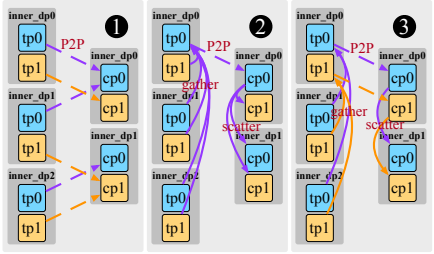


Figure 5. Tensor resharding strategies.

needs to complete. This queue-based mechanism enables non-blocking communication: when the pipeline scheduler needs to issue a cross-cluster communication command, it can immediately launch the asynchronous operation and continue with other tasks without waiting for completion. The actual synchronization (waiting for data) only occurs when the receiving stage needs the tensor for computation. This design allows cross-cluster data transfers to overlap with computation stages in both sender and receiver clusters, as illustrated in Fig. 4. Our cross-cluster P2P operation maintains the same interface as standard P2P operations, and our virtual stage abstraction is compatible with various pipeline scheduling designs including interleaved-1F1B [48], Chimera [39], and zero bubble pipeline parallelism [57].

5.2 Optimized Resharding

Since different stages may adopt different intra-parallelism strategies (i.e., different DP, CP, TP degrees), the tensor layout needs to be resharded across stages according to the intra-parallelism strategies of consecutive stages. Given DP, CP, TP degrees d_i, c_i, t_i of stage i, the expected input tensor layout for each device in stage i is $[\frac{b}{d_i}, \frac{s}{c_i t_i}, h]$, where b represents the micro-batch size (i.e., global batch size B divided by the number of micro-batches q), s is the sequence length (divided by c_i due to context parallelism and further divided by t_i due to the default use of sequence parallelism [35] to reduce memory consumption), and h is the hidden size. For stage $i+1$, the expected input tensor layout for each device is $[\frac{b}{d_{i+1}}, \frac{s}{c_{i+1} t_{i+1}}, h]$. Hence, tensors must be resharded along both batch and sequence dimensions. To optimize resharding efficiency, we decompose the DP degree of each stage into outer and inner components: $d_i = d^{outer} \cdot d_i^{inner}$, where $d^{outer} = \gcd(d_1, d_2, \ldots, d_p)$ represents the greatest common divisor of DP degrees across all p pipeline stages. This decomposition allows us to focus on each outer DP partition, as devices within the same outer partition across different stages process identical batch data samples. The expected tensor layout for each stage can be reformulated as $[\frac{b'}{d_i^{inner}}, \frac{s}{c_i t_i}, h]$, where $b' = \frac{b}{d^{outer}}$.

The naive resharding strategy (**Strategy ❶**) employs only P2P operations to transfer tensors [61, 77]. In the example in Fig. 5, for tensor resharding from $d_i^{inner} = 3, c_i = 1, t_i = 2$ in stage i to $d_{i+1}^{inner} = 2, c_{i+1} = 2, t_{i+1} = 1$ in stage $i+1$, strategy ❶ requires 6 individual P2P transfers: each of the 6 devices in stage i sends tensor slices to specific devices in stage $i+1$ based on the resharding requirements. While conceptually simple, this strategy results in numerous small P2P transfers that cause fragmented bandwidth usage and network congestion.

Another possible strategy (**Strategy ❷**) is to first gather all tensor slices to the first rank in stage i, and then perform a single P2P operation to transfer the complete tensor to the first rank of the next stage, followed by scattering the tensor to each device in stage $i+1$. While this strategy uses only a single P2P transfer, gathering all tensor slices to the first rank creates a communication bottleneck and can cause OOM issues more easily.

We propose an improved resharding strategy (**Strategy ❸**) which leverages devices with inner DP rank 0 as intermediaries. Inner DP rank 0 refers to devices in the first inner DP group in each stage — in our example, these are inner_dp0/tp0, inner_dp0/tp1 in stage i, and inner_dp0/cp0, inner_dp0/cp1 in stage $i+1$. The resharding process works as follows in our example: (1) Gather: devices with inner DP ranks 1 and 2 send their tensor slices to the corresponding device with inner DP rank 0; (2) P2P: Only devices with inner DP rank 0 perform cross-stage P2P transfers, where inner_dp0/tp0 sends to inner_dp0/cp0, and inner_dp0/tp1 sends to inner_dp0/cp1; (3) Scatter: Devices with inner DP rank 0 in stage i+1 distribute received tensors to other devices within their CP groups. From the tensor layout perspective, the resharding is carried out as:

$$[\frac{b'}{d_i^{inner}}, \frac{s}{c_i t_i}, h] \xrightarrow{\text{gather}} [b', \frac{s}{c_i t_i}, h] \xrightarrow{\text{P2P}} [b', \frac{s}{c_{i+1} t_{i+1}}, h] \xrightarrow{\text{scatter}} [\frac{b'}{d_{i+1}^{inner}}, \frac{s}{c_{i+1} t_{i+1}}, h] \quad (2)$$

This strategy strikes a balance between strategies ❶ and ❷, reducing cross-stage communication while avoiding OOM issues. Our resharding strategy integrates naturally with Virtual-1F1B. For consecutive stages deployed across clusters, cross-stage communication can be decomposed into: gather, device-to-host (D2H) copy, asynchronous cross-cluster P2P transfer, host-to-device (H2D) copy, and scatter operations.

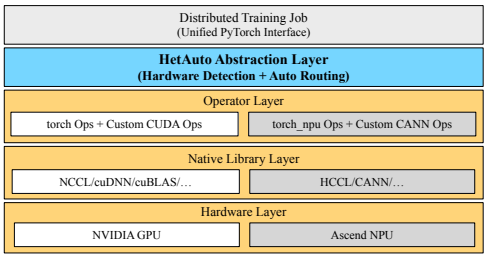


Figure 6. Unified hardware abstraction framework.

6 Unified Abstraction Layer

We design a unified hardware abstraction layer for HetAuto runtime, enabling transparent training execution across different hardware platforms with minimal user intervention.

The abstraction layer sits above existing vendor frameworks, as illustrated in Fig. 6. It performs runtime hardware detection and automatic backend selection, leveraging existing vendor software stacks: PyTorch with CUDA/NCCL backend for NVIDIA GPUs [51, 54] and torch_npu/CANN/HCCL backend for Ascend NPUs [25–27]. Upon initialization, HetAuto detects available accelerator types and configures the appropriate software stack. The system automatically routes operations to the corresponding vendor-optimized implementations or custom operators, while maintaining API compatibility across different hardware platforms through a unified PyTorch interface. Custom operators are implemented using hardware-specific primitives (CUDA kernels for GPUs, CANN libraries for NPUs) to optimize performance-critical computations. This lightweight design eliminates conditional compilation and manual backend specification while exploiting proven vendor optimizations. Users write training code using standard PyTorch APIs, and HetAuto's abstraction layer ensures seamless execution across different hardware configurations.

Our abstraction layer operates at a different level from PyTorch's built-in operator dispatch mechanism [54, 55]. PyTorch dispatch selects kernel implementations (CUDA, ROCm, CPU) for individual operators, typically within a single-backend execution environment. In contrast, our abstraction layer addresses *distributed system-level heterogeneity*: it coordinates multiple collective communication backends (e.g., NCCL for intra-NVIDIA-cluster communication, HCCL for Ascend NPU clusters, and Gloo for cross-cluster host-based transfers), manages the routing of pipeline-parallel P2P operations across heterogeneous software stacks, and handles mixed-accelerator pipelines where adjacent stages reside in different vendor ecosystems. In this sense, our abstraction layer serves as system-level glue for heterogeneous distributed training, while relying on PyTorch's built-in dispatch for operator-level kernel execution. Leading accelerator platforms such as AWS Trainium [18, 21] adopt a similar philosophy of providing PyTorch-compatible frontends; our abstraction layer complements this trend by providing the system-level orchestration needed to unify diverse accelerators within a single training job.

Table 2. Experimental configurations for model training

Exp	# Layers	GBS	Clusters Setup
exp1	48	128	32 A100 + 32 Ascend A2
exp2	64	128	32 H20 + 32 A100 + 32 Ascend A2
exp3	96	512	32 H20 + 64 H800 + 128 A100 + 512 Ascend A2

GBS: Global batch size. Hidden size: 4096, attention heads: 32, sequence length: 8192.

7 Evaluation

7.1 Implementation

HetAuto is implemented in approximately 12.8k lines of Python code (LoC). We build the HetAuto runtime on top of Megatron-LM [35, 48, 61], a state-of-the-art framework for training large models in homogeneous environments. We extend Megatron-LM to support heterogeneous training by enabling different intra-parallelism strategies for each pipeline stage, incorporating our Virtual-1F1B scheduling and optimized resharding strategy, and integrating the unified abstraction layer for hardware detection and automatic operation routing. The extension to Megatron-LM comprises approximately 5.2k LoC. The principle-guided MCTS search algorithm is implemented in 4k LoC, and the cost model, implemented on top of `llm-analysis` [37], approximately 3.6k LoC. HetAuto adopts Gloo [17] as the communication backend for cross-cluster communication.

In implementing our principle-guided MCTS, we incorporate two optimizations to further reduce the search overhead. **Memoization.** We use memoization to cache prediction results for previously encountered inputs (i.e., intra-parallelism strategy and input shape), avoiding redundant computation. **Multiprocessing.** The children of the root node form independent sub-trees. Each sub-tree corresponds to a fixed number of pipeline stages p and optimizes the remaining parallelism decisions. Since different values of p lead to independent inner optimization problems with distinct solution spaces, we leverage multiprocessing to solve these subproblems in parallel, significantly accelerating the overall search process.

7.2 Experimental Setup

Testbed. We conduct our experiments across four heterogeneous clusters: 32 NVIDIA H20-141GB GPUs (4 nodes × 8 GPUs/node), 64 NVIDIA H800-80GB GPUs (8 nodes × 8 GPUs/node), 128 NVIDIA A100-80GB GPUs (16 nodes × 8 GPUs/node), and 512 Ascend A2-64GB NPUs (32 nodes × 16 devices/node). For NVIDIA GPU clusters, intra-cluster connectivity is 100 Gbps InfiniBand. The Ascend NPU cluster uses 100 Gbps RoCE for intra-cluster communication. Inter-cluster bandwidth between any two clusters is approximately 10 Gbps as measured by `iperf3` [16].

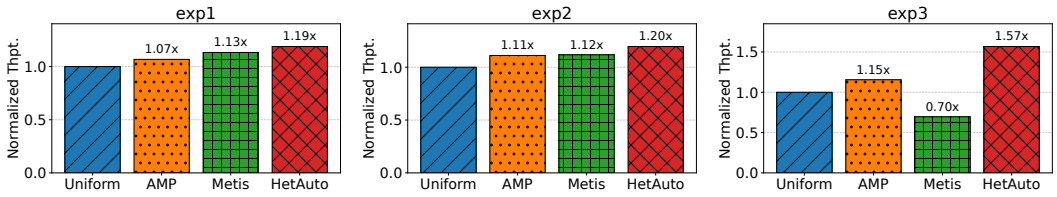


Figure 7. Training throughput, normalized by throughput of Uniform. Metis fails to complete its search within 12 hours for exp2 and exp3; we report its best results found within this time budget. Detailed configurations of exp 1-3 are given in Table 2.

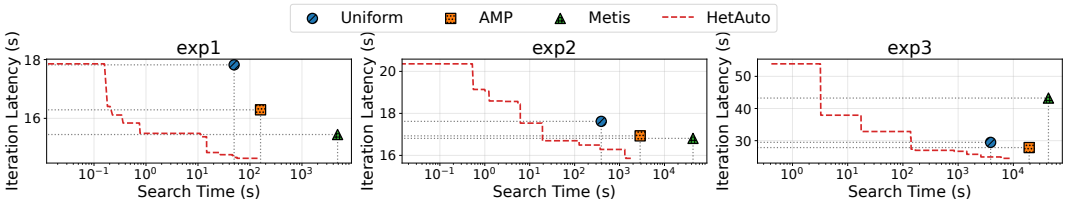


Figure 8. Parallelism strategy search time vs. training performance (iteration latency).

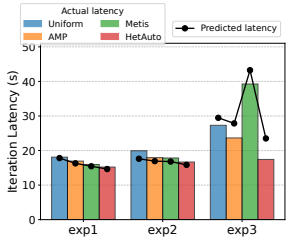


Figure 9. Actual vs. predicted training iteration latency.

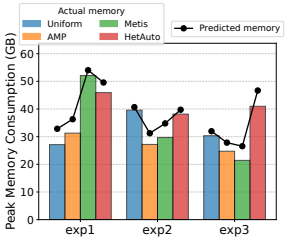


Figure 10. Actual vs. predicted peak memory usage.

We train Llama models [65] of different sizes, varying the number of layers and global batch sizes. The detailed configurations are given in Table 2.

Baselines. We compare HetAuto against the following baselines: **(1) Uniform:** A naive adaptation of Megatron-LM [61] to heterogeneous clusters. It requires each pipeline stage to adopt the same intra-parallelism strategy and contain the same number of model layers. For pipeline parallelism across clusters, we use the optimal solution found by enumerating all possible combinations of pipeline stage partition, micro-batch number, cluster permutations, and intra-parallelism strategies. **(2) AMP [38]:** An auto-parallelism system for heterogeneous clusters that assumes each pipeline stage adopts the same intra-parallelism strategy (to avoid combinatorial explosion in strategy search). It uses dynamic programming ($O(L^4)$ complexity where L is the number of model layers) to find optimal layer distribution among pipeline stages, to achieve inter-stage load balancing. **(3) Metis [66]:** An adaptation of the homogeneous auto-parallelism system Alpa [74] to heterogeneous environments. Metis assigns layers according to each stage's computing capacity. It faces combinatorial explosion when determining device allocation and intra-parallelism strategies, making it computationally intractable for large-scale systems.

Neither AMP nor Metis consider context parallelism, limiting their ability to comprehensively explore intra-parallelism strategies. Critically, to isolate the contribution of each system's search algorithm, all systems — including Uniform, AMP, and Metis — use the same runtime optimizations during evaluation: our proposed Virtual-1F1B scheduling, optimized resharding, and random forest-enhanced cost model. Therefore, the throughput differences reported in Fig. 7 are solely attributable to the quality of parallelization strategies discovered by each system's search algorithm. For HetAuto, we use a default exploration weight $\lambda = 10$ in Eqn. (1) and set the search time budget according to problem scale: 2 minutes for exp1, 25 minutes for exp2, and 150 minutes for exp3.

7.3 Training Throughput

We evaluate training throughput across different systems and settings, by running respective model training for 30 iterations (with the first 5 iterations as warmup) and presenting the average throughput in Fig. 7. Due to its overly-simplified search space, AMP achieves only around 10% throughput improvement over Uniform. Metis suffers from scalability issues due to its exhaustive search — it failed to complete the search process within a reasonable time budget (12 hours) in exp2 and exp3, limiting its applicability to large-scale settings. We report the best results found by Metis within the 12-hour time budget for these two experiments. Notably, in exp3, Metis even achieves 30% lower throughput than Uniform, as it can only explore a small fraction of the vast search space within the allotted time. Additionally, both AMP and Metis do not consider context parallelism, affecting their performance in today's popular long-context training scenarios. HetAuto achieves significant training throughput improvement across all settings. The improvement is particularly notable in exp3 (up to 1.57×), where heterogeneity is amplified by increasing both device types and cluster size imbalance. In this scenario, the cluster with the fewest devices (H20 cluster with 32 devices) severely constrains the search

space for Uniform and AMP: since they require each pipeline stage to use identical intra-parallelism strategies (i.e., the same number of devices), no pipeline stage can contain more than 32 devices. HetAuto removes this restriction and can identify better parallelism strategies. The optimal parallelism strategies given by Uniform and AMP both have 23 pipeline stages, with each stage containing 32 devices. In contrast, the optimal solution given by HetAuto has 18 pipeline stages, where each cluster adopts a distinct intra-parallelism strategy and layer number. We give the exact solution in exp3 produced by HetAuto in Appendix D.

7.4 Search Overhead

Fig. 8 shows the parallelism strategy search time and corresponding training iteration latency estimated by our cost models, under different methods. AMP and Metis adopt dynamic programming and exhaustive search, respectively; their search process runs till obtaining the final solution, so does Uniform. The discrete points in Fig. 8 give the search completion time and training iteration latency incurred using the solution found under the three baselines, respectively. HetAuto's principle-guided MCTS progressively converges toward the exhaustive search solution: with a larger time budget, HetAuto generates better solutions, as illustrated by the continuous lines in Fig. 8. Under the same time budget as the solution time of baselines, HetAuto always identifies parallelism strategies achieving better training performance, and attains solutions of the same performance using much shorter search time.

7.5 Cost Model Accuracy

We evaluate the prediction accuracy of our cost models. Fig. 9 compares the actual training iteration latency and cost-model predicted latency across different parallelization strategies found by various methods. Note that all methods use the same cost model (our proposed random forest-enhanced model) for fair comparison; the different data points represent various parallelization configurations discovered by each method, providing diverse samples to evaluate cost model accuracy. Our latency cost model achieves satisfactory prediction accuracy (with prediction errors ranging from 1.5% to 34.9%), especially in smaller-scale settings (exp1 and exp2). Although the accuracy decreases as the problem scale increases, the cost model can still effectively discriminate which solution performs better.

Fig. 10 shows actual peak memory consumption in the first pipeline stage and memory usage predicted by our memory cost model. Our memory cost model deliberately overestimates consumption (ranging from 2.8% to 21.2%) to prevent OOM issues, following a 'safety first' approach.

7.6 Profiling Overhead

For each cluster, we use at most two nodes to profile single Transformer layer forward latency under different CP and TP degrees with varying input shapes. For NVIDIA GPUs (up to 16 devices), we test CP-TP combinations $(c, t) = (1, 1), (1, 2), (2, 1), (1, 4), (2, 2), \ldots, (8, 2), (16, 1)$. For each CP-TP combination, we vary the input shape: hidden size from 1024 to 4096, batch size from 1 to 16, sequence length from 1024 to 8192, and attention heads from 8 to 32. For each specific CP-TP degree and input shape, we profile single layer execution latency for 10 iterations (with 5 warmup iterations) and use the average value. For Ascend A2 NPU cluster, we follow the same process but use at most 32 devices. This profiling is cluster-dependent — for each cluster, we profile once to build the cost model, then reuse it across exp1-3. We skip invalid combinations during profiling (e.g., CP=1, TP=16 with fewer than 16 attention heads). Table 3 shows the profiling time. We use 80% of the profiled data to train a random forest model for single layer latency prediction, with 20% as test data. The random forest model's prediction accuracy is detailed in Appendix C.3.

Table 3. Profiling time for different device types

Device	H20-141GB	H800-80GB	A100-80GB	Ascend A2-64GB
Time (min)	11.10	7.88	10.53	19.98

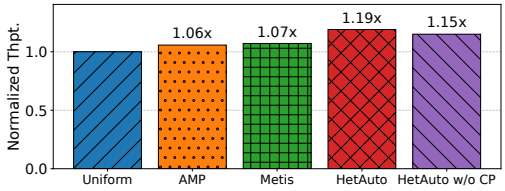


Figure 11. Training throughput, normalized by the throughput of Uniform. We use the exp2 training configuration with sequence length 32,768.

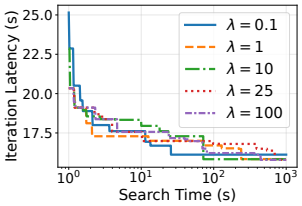


Figure 12. Strategy search time vs. iteration latency under different exploration weights.

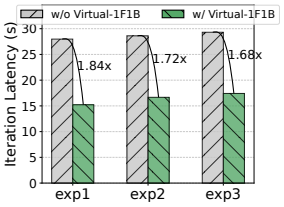


Figure 13. Training iteration latency of HetAuto with and without Virtual-1F1B.

7.7 Ablation Study

We next explore the impact of hyperparameters and core design modules in the overall performance of HetAuto.

Context Parallelism. As shown in Fig. 11, to evaluate the impact of including context parallelism in the search space, we conduct additional cost-model-based evaluations with sequence length 32,768 under the exp2 cluster configuration (Metis again fails to complete its search within 12 hours, so we report its best-found result). At this longer sequence length, HetAuto with CP enabled achieves higher throughput than HetAuto without CP, demonstrating that CP is

beneficial for long-context training. In contrast, at sequence length 8,192 (exp1/exp2), MCTS automatically selects CP=1, confirming that including CP in the search space does not degrade performance when it is unnecessary. Together, these results show that MCTS can automatically exploit CP when longer sequences justify its use, without introducing overhead otherwise. Even with CP disabled, HetAuto still outperforms other baselines by up to 15%, further confirming its effectiveness.

Exploration Weight. The exploration weight λ in UCB (Eqn. (1)) of the MCTS algorithm controls the trade-off between exploration and exploitation. Under the exp2 setting, Fig. 12 shows that $\lambda = 0.1$ leads to premature convergence to a suboptimal solution due to insufficient exploration, while moderate λ values (1, 10, 25, 100) achieve better and comparable performance. Since determining the optimal λ for a specific training job and time budget is challenging in advance, we have used $\lambda = 10$ by default in all our experiments as a balanced compromise.

Virtual-1F1B and resharding. To evaluate the effect of our pipeline scheduling design, Fig. 13 illustrates HetAuto's training iteration latency with and without Virtual-1F1B (we use 1F1B in the latter). With Virtual-1F1B, the iteration latency is significantly reduced by 1.68× to 1.84×, as compared to standard 1F1B. This demonstrates that optimizing cross-cluster communication in cross-cluster distributed training is crucial. See Appendix E for resharding comparisons.

7.8 Contribution Decomposition

The performance improvements of HetAuto stem from two orthogonal sources. **(1) MCTS-based strategy search.** As shown in Fig. 7, where all systems share the same runtime infrastructure (including Virtual-1F1B and optimized resharding), HetAuto's MCTS achieves 1.19–1.57× throughput improvement purely by discovering superior parallelization strategies. **(2) Optimized cross-cluster communication.** As shown in Fig. 13, Virtual-1F1B yields an additional 1.68–1.84× speedup over standard 1F1B by overlapping cross-cluster communication with computation. Furthermore, optimized resharding (Fig. 16 in Appendix E) achieves the lowest average latency while avoiding severe fluctuations. As these two sources target different aspects of the system, their improvements are multiplicative.

8 Discussion

Scope and Limitations. HetAuto addresses *system-level* heterogeneity (accelerator types, cluster sizes, interconnect topologies, and software stacks). Numerical precision differences across accelerator types are an *operator-level* concern orthogonal to parallelization strategy design; we assume vendor-provided operators meet the precision requirements for the specified data format, a standard prerequisite for production deployment [18, 21]. Investigating the impact of numerical heterogeneity on training convergence is an interesting direction for future work. Our current design also assumes stable inter-cluster bandwidth, measured once during profiling. This assumption holds in our deployment with dedicated inter-datacenter links that exhibit minimal bandwidth fluctuation during training runs. For environments with time-varying bandwidth (e.g., shared WAN links [62]), HetAuto's cost model and MCTS search can be re-invoked periodically to adapt. Fully online adaptive re-optimization is a promising direction for future work.

Extension to MoE Models. Our evaluation focuses on dense Transformer architectures (e.g., Llama [65]). For Mixture-of-Experts (MoE) models [11, 20, 31, 36, 60], HetAuto's MCTS framework can incorporate expert parallelism (EP) as an additional decision dimension with minimal structural changes. The main challenge lies in cost modeling: MoE per-layer latency is input-dependent due to dynamic token routing, requiring finer-grained profiling of expert utilization distributions. Extending HetAuto's cost model to support MoE workloads is an important direction for future work.

Privacy, Security, and Fault Tolerance. Our target scenario involves clusters owned by the *same organization*, connected via dedicated or encrypted links [23, 32]; data privacy is managed through existing organizational infrastructure, as is standard in industry-scale deployments [29]. Multi-tenant or cross-organization settings (e.g., federated learning [43]) are beyond our scope. Regarding fault tolerance, large-scale training across multiple clusters increases the surface area for hardware failures. While HetAuto does not implement cluster-aware fault tolerance, it is compatible with standard checkpoint-based recovery [12, 46, 56, 69, 71]. Because MCTS search completes in minutes and the cost model is pre-built, re-planning for reduced resources upon failure can be performed quickly.

9 Conclusion

We present HetAuto, an automatic parallelization system for cross-cluster heterogeneous distributed training. HetAuto combines principle-guided MCTS with random forest-enhanced cost models and introduces Virtual-1F1B with optimized resharding to mitigate cross-cluster bottlenecks. A unified abstraction layer enables seamless deployment across heterogeneous hardware. Evaluation across four heterogeneous clusters with up to 736 devices demonstrates up to 1.57× training throughput improvements over baselines.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd Pamela Delgado for their valuable feedback. This work was supported in part by a Meituan Research Collaboration Project, and grants from Hong Kong RGC under the contracts HKU 17204423, 17204625, C7004-22G (CRF), and CRS_PolyU501/23.

References

[1] Advanced Micro Devices. 2024. RCCL: ROCm Communication Collectives Library. https://github.com/ROCmSoftwarePlatform/rccl.

[2] Advanced Micro Devices. 2024. ROCm: Radeon Open Compute Platform. https://rocm.docs.amd.com/.

[3] Reed Albergotti. 2024. Microsoft Azure CTO: US data centers will soon hit size limits. Semafor. Technology.

[4] Anthropic. 2025. Claude Sonnet 4. https://www.anthropic.com/claude Large language model.

[5] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[6] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

[7] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[8] Tiancheng Chen, Ales Kubicek, Langwen Huang, and Torsten Hoefler. 2025. CrossPipe: Towards Optimal Pipeline Schedules for Cross-Datacenter Training. In *Proceedings of the 2025 USENIX Annual Technical Conference, USENIX ATC 2025, Boston, MA, USA, July 7-9, 2025*, Deniz Altinbüken and Ryan Stutsman (Eds.). USENIX Association, 1089–1108. https://www.usenix.org/conference/atc25/presentation/chen-tiancheng

[9] Runxiang Cheng, Chris Cai, Selman Yilmaz, Rahul Mitra, Malay Bag, Mrinmoy Ghosh, and Tianyin Xu. 2023. Towards GPU Memory Efficiency for Distributed Training at Scale. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '23)*. Association for Computing Machinery, New York, NY, USA, 281–297. https://doi.org/10.1145/3620678.3624661

[10] Tapan Chugh, Srikanth Kandula, Arvind Krishnamurthy, Ratul Mahajan, and Ishai Menache. 2023. Anticipatory Resource Allocation for ML Training. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '23)*. Association for Computing Machinery, New York, NY, USA, 410–426. https://doi.org/10.1145/3620678.3624669

[11] Damai Dai, Chengqi Deng, Chenggang Zhao, R.X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y.K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 1280–1297. https://doi.org/10.18653/v1/2024.acl-long.70

[12] John T. Daly. 2006. A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. *Future Generation Computer Systems* 22, 3 (2006), 303–312. https://doi.org/10.1016/j.future.2004.11.016

[13] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *NeurIPS*.

[14] DeepSeek-AI. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[15] Jiangfei Duan, Xiuhong Li, Ping Xu, Xingcheng Zhang, Shengen Yan, Yun Liang, and Dahua Lin. 2024. Proteus: Simulating the Performance of Distributed DNN Training. *IEEE Trans. Parallel Distrib. Syst.* 35, 10 (Oct. 2024), 1867–1878. https://doi.org/10.1109/TPDS.2024.3443255

[16] Energy Sciences Network (ESnet). 2014. *iPerf3: A TCP, UDP, and SCTP network bandwidth measurement tool.* https://iperf.fr/ Network performance measurement tool.

[17] Facebook. 2017. Gloo: A Collective Communications Library. https://github.com/facebookincubator/gloo.

[18] Haozheng Fan, Hao Zhou, Guangtai Huang, Parameswaran Raman, Xinwei Fu, Gaurav Gupta, Dhananjay Ram, Yida Wang, and Jun Huan. 2024. HLAT: High-quality Large Language Model Pre-trained on AWS Trainium. In *2024 IEEE International Conference on Big Data (BigData)*. IEEE Computer Society, 2100–2109. https://doi.org/10.1109/BigData62323.2024.10825098

[19] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, Lansong Diao, Xiaoyong Liu, and Wei Lin. 2021. 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* (Virtual Event, Republic of Korea) *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 431–445. https://doi.org/10.1145/3437801.3441593

[20] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39. http://jmlr.org/papers/v23/21-0998.html

[21] Xinwei Fu, Zhen Zhang, Haozheng Fan, Guangtai Huang, Mohammad El-Shabani, Randy Huang, Rahul Solanki, Fei Wu, Ron Diamant, and Yida Wang. 2024. Distributed Training of Large Language Models on AWS Trainium. In *Proceedings of the 2024 ACM Symposium on Cloud Computing (SoCC '24)*. ACM, Redmond, WA, USA, 1–16. https://doi.org/10.1145/3698038.3698535

[22] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2018. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677

[23] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu Bollineni Narayanan, Theophilus Benson, et al. 2018. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. ACM, 74–87.

[24] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. *GPipe: efficient training of giant neural networks using pipeline parallelism.* Curran Associates Inc., Red Hook, NY, USA.

[25] Huawei. 2023. CANN: Compute Architecture for Neural Networks. https://www.hiascend.com/software/cann.

[26] Huawei. 2023. HCCL: Huawei Collective Communication Library. https://www.hiascend.com/cann/hccl.

[27] Huawei. 2023. torch_npu: PyTorch extension for Huawei Ascend NPU. https://gitee.com/ascend/pytorch.

[28] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. 2023. DeepSpeed Ulysses: System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models. arXiv:2309.14509 [cs.LG] https://arxiv.org/abs/2309.14509

[29] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference (ATC)*. 947–960.

[30] Xianyan Jia, Le Jiang, Ang Wang, Wencong Xiao, Ziji Shi, Jie Zhang, Xinyuan Li, Langshi Chen, Yong Li, Zhen Zheng, Xiaoyong Liu, and Wei Lin. 2022. Whale: Efficient Giant Model Training over Heterogeneous GPUs. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 673–688. https://www.usenix.org/conference/atc22/presentation/jia-xianyan

[31] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las

Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of Experts. *arXiv preprint arXiv:2401.04088* (2024).

[32] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. 2014. Calendaring for wide area networks. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 515–526.

[33] Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[34] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006 (Lecture Notes in Computer Science, Vol. 4212)*, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (Eds.). Springer, Berlin, Heidelberg, 282–293. https://doi.org/10.1007/11871842_29

[35] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2022. Reducing Activation Recomputation in Large Transformer Models. arXiv:2205.05198 [cs.LG] https://arxiv.org/abs/2205.05198

[36] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net.

[37] Cheng Li. 2023. LLM-Analysis: Latency and Memory Analysis of Transformer Models for Training and Inference. https://github.com/cli99/llm-analysis.

[38] Dacheng Li, Hongyi Wang, Eric Xing, and Hao Zhang. 2022. AMP: automatically finding model parallel strategies with heterogeneity awareness. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 480, 10 pages.

[39] 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* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 27, 14 pages. https://doi.org/10.1145/3458817.3476145

[40] Andy Liaw and Matthew Wiener. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[41] Guodong Liu, Youshan Miao, Zhiqi Lin, Xiaoxiang Shi, Saeed Maleki, Fan Yang, Yungang Bao, and Sa Wang. 2024. Aceso: Efficient Parallel DNN Training through Iterative Bottleneck Alleviation. In *Proceedings of the Nineteenth European Conference on Computer Systems* (Athens, Greece) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 163–181. https://doi.org/10.1145/3627703.3629554

[42] Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring Attention with Blockwise Transformers for Near-Infinite Context. arXiv:2310.01889 [cs.CL] https://arxiv.org/abs/2310.01889

[43] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) (Proceedings of Machine Learning Research, Vol. 54)*. PMLR, 1273–1282.

[44] Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. 2022. Galvatron: Efficient Transformer Training over Multiple GPUs Using Automatic Parallelism. *Proc. VLDB Endow.* 16, 3 (Nov. 2022), 470–479. https://doi.org/10.14778/3570690.3570697

[45] Dejan Milojicic, Paolo Faraboschi, Nicolas Dube, and Duncan Roweth. 2021. Future of HPC: Diversifying heterogeneity. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 276–281.

[46] Jayashree Mohan, Amar Phanishayee, and Vijay Chidambaram. 2021. CheckFreq: Frequent, Fine-Grained DNN Checkpointing. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, 203–216. https://www.usenix.org/conference/fast21/presentation/mohan

[47] 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* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3341301.3359646

[48] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 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* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 58, 15 pages. https://doi.org/10.1145/3458817.3476209

[49] NVIDIA. 2020. *NVIDIA A100 Tensor Core GPU Architecture*. Whitepaper. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

[50] NVIDIA. 2022. *NVIDIA H100 Tensor Core GPU Architecture*. Whitepaper. https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

[51] NVIDIA. 2023. NCCL: NVIDIA Collective Communications Library. https://developer.nvidia.com/nccl.

[52] NVIDIA Corporation. 2024. *CUDA C++ Programming Guide*. https://docs.nvidia.com/cuda/cuda-c-programming-guide/ Version 12.3.

[53] OpenAI. 2023. GPT-4. https://openai.com/gpt-4.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. 8024–8035.

[55] PyTorch Team. 2023. PyTorch Dispatcher: A Technical Deep Dive. https://pytorch.org/blog/pytorch-dispatcher/.

[56] Guicheng Qi, Zongpeng Li, Chuan Wu, Zhuwei Peng, and Yi Zheng. 2025. ECCheck: Enhancing In-Memory Checkpoint with Erasure Coding in Distributed DNN Training. In *2025 IEEE 45th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Glasgow, Scotland, 253–263. https://doi.org/10.1109/ICDCS63083.2025.00033

[57] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. 2023. Zero Bubble Pipeline Parallelism. arXiv:2401.10241 [cs.DC] https://arxiv.org/abs/2401.10241

[58] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 20, 16 pages.

[59] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 3505–3506. https://doi.org/10.1145/3394486.3406703

[60] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.

In *5th International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=B1ckMDqlg

[61] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053

[62] Foteini Strati, Paul Elvinger, Tolga Kerimoglu, and Ana Klimovic. 2024. ML Training with Cloud GPU Shortages: Is Cross-Region the Answer?. In *Proceedings of the 4th Workshop on Machine Learning and Systems* (Athens, Greece) *(EuroMLSys '24)*. Association for Computing Machinery, New York, NY, USA, 107–116. https://doi.org/10.1145/3642970.3655843

[63] Foteini Strati, Zhendong Zhang, George Manos, Ixeia Sánchez Périz, Qinghao Hu, Tiancheng Chen, Berk Buzcu, Song Han, Pamela Delgado, and Ana Klimovic. 2025. Sailor: Automating Distributed Training over Dynamic, Heterogeneous, and Geo-distributed Clusters. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles* (Lotte Hotel World, Seoul, Republic of Korea) *(SOSP '25)*. Association for Computing Machinery, New York, NY, USA, 204–220. https://doi.org/10.1145/3731569.3764839

[64] Jakub Tarnawski, Deepak Narayanan, and Amar Phanishayee. 2021. Piper: multidimensional planner for DNN parallelization. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 1902, 12 pages.

[65] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[66] 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 *Proceedings of the 2024 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA, USA) *(USENIX ATC'24)*. USENIX Association, USA, Article 35, 16 pages.

[67] Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, Xi Luo, Dheevatsa Mudigere, Jongsoo Park, Misha Smelyanskiy, and Alex Aiken. 2022. Unity: Accelerating DNN Training Through Joint Optimization of Algebraic Transformations and Parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 267–284. https://www.usenix.org/conference/osdi22/presentation/unger

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[69] Borui Wan, Mingji Han, Yiyao Sheng, Yanghua Peng, Haibin Lin, Mofan Zhang, Zhichao Lai, Menghan Yu, Junda Zhang, Zuquan Song, Xin Liu, and Chuan Wu. 2025. ByteCheckpoint: A Unified Checkpointing System for Large Foundation Model Development. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 559–578. https://www.usenix.org/conference/nsdi25/presentation/wan-borui

[70] Peiran Wang, Haibing Li, Fu Haohan, Shiyong Li, Yanpeng Wang, and Dou Shen. 2025. Astra: Efficient and Money-saving Automatic Parallel Strategies Search on Heterogeneous GPUs. arXiv:2502.13480 [cs.DC] https://arxiv.org/abs/2502.13480

[71] Zhuang Wang, Zhen Jia, Shuai Zheng, Zhen Zhang, Xinwei Fu, T. S. Eugene Ng, and Yida Wang. 2023. GEMINI: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23)*. ACM, Koblenz, Germany. https://doi.org/10.1145/3600006.3613145

[72] Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. 2021. Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 503–521. https://www.usenix.org/conference/atc21/presentation/yu

[73] 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* (Athens, Greece) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 524–541. https://doi.org/10.1145/3627703.3629580

[74] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 559–578. https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin

[75] Hongyu Zhu, Mohamed Akrout, Boyuan Zheng, Andrew Pelegris, Anand Jayarajan, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko. 2020. Daydream: Accurately Estimating the Efficacy of Optimizations for DNN Training. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 337–352.

[76] Zhanda Zhu, Christina Giannoula, Muralidhar Andoorveedu, Qidong Su, Karttikeya Mangalam, Bojian Zheng, and Gennady Pekhimenko. 2025. Mist: Efficient Distributed Training of Large Language Models via Memory-Parallelism Co-Optimization. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 1298–1316. https://doi.org/10.1145/3689031.3717461

[77] 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. In *Proceedings of Machine Learning and Systems*, D. Song, M. Carbin, and T. Chen (Eds.), Vol. 5. Curan, 526–540. https://proceedings.mlsys.org/paper_files/paper/2023/file/a42cbafcabb6dc7ce77bfe2e80f5c772-Paper-mlsys2023.pdf

[78] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2022. Monte Carlo Tree Search: a review of recent modifications and applications. *Artificial Intelligence Review* 56, 3 (July 2022), 2497–2562. https://doi.org/10.1007/s10462-022-10228-y

A Detailed Complexity Analysis

We provide a detailed analysis of the exponential search space complexity using a realistic training scenario to illustrate the scale of the optimization challenge. Consider a concrete example: training a 70B parameter Llama-2 model [65] across 3 heterogeneous clusters. The model contains 80 transformer layers, with each layer comprising approximately 15 operators (including attention computation, feed-forward networks, layer normalization), totaling around 1,200 operators in the computational graph. As a domain-specific simplification, we partition at layer boundaries rather than at the operator level. The clusters contain 256 NVIDIA A100 GPUs (32 nodes × 8 GPUs/node), 256 NVIDIA H20 GPUs (32 nodes × 8 GPUs/node), and 512 Ascend A2 NPUs (32 nodes × 16 devices/node) respectively. For this scenario, the system faces a multi-dimensional optimization problem:

• **Pipeline stage number selection**: The number of pipeline stages p can range from 1 to the number of layers (80), introducing an initial factor of 80 possibilities.

• **Graph partitioning**: For each choice of p stages, there are $\binom{79}{p-1}$ ways to partition at layer boundaries. Summing across all possible stage numbers: $\sum_{p=1}^{80}\binom{79}{p-1} = 2^{79} \approx 6 \times 10^{23}$ total partitioning options.

• **Device type combinations**: Each stage can utilize devices from any non-empty subset of the 3 clusters (7 possibilities per stage). For p stages, this yields 7^p combinations. Averaging across different stage numbers (assuming typical $p \in [8, 32]$), this contributes approximately $7^{20} \approx 8 \times 10^{16}$ combinations.

• **Mesh shape selection**: Single-cluster configurations offer approximately 35, 35, and 36 shapes for A100, H20, and Ascend A2 clusters respectively. Cross-cluster configurations multiply these possibilities. Considering all device type combinations, each stage faces an average of approximately 10^2 mesh shape options, resulting in $(10^2)^{20} = 10^{40}$ total mesh shape combinations across all stages.

• **Intra-parallelism strategies**: For each mesh configuration, the system must select DP, TP, and CP degrees such that $DP \times TP \times CP$ equals the mesh size. The number of valid combinations follows $d_3(n)$ (three-factor divisor function), typically yielding 20-50 valid combinations per mesh. Using an average of 30 combinations per stage contributes $(30)^{20} \approx 3 \times 10^{29}$.

The complete search space contains approximately:

$$6 \times 10^{23} \times 8 \times 10^{16} \times 10^{40} \times 3 \times 10^{29} \approx 1.4 \times 10^{110}$$

This astronomical number of configurations makes exhaustive exploration computationally impossible.

B Complete Problem Formulation

This appendix provides the complete mathematical formulation of the cross-cluster auto-parallelism optimization problem discussed in § 4.

B.1 Formal Optimization Problem

Table 4. Complete notations for problem formulation

Notation	Description
B	Global batch size
s	Sequence length
h	Hidden size ($h = h_n \times h_d$)
h_n	Number of attention heads
h_d	Head dimension
(N_j, M_j)	Total mesh dimensions of cluster j (nodes × devices per node), $1 \le j \le C$
X_j	Memory capacity for each device in cluster j, $1 \le j \le C$
C	Number of clusters
$\mathcal{O}$	Set of operators in the computation graph
$\mathcal{S}$	Scheduling strategy
$\mathcal{R}$	Recomputation level
p	Number of pipeline stages
q	Number of micro-batches
$(n_{i,j}, m_{i,j})$	Device mesh from cluster j assigned to stage i (where $n_{i,j} \le N_j, m_{i,j} \le M_j$)
$d_{i,j}$	Data parallelism degree for device mesh from cluster j in stage i
$c_{i,j}$	Context parallelism degree for device mesh from cluster j in stage i
$t_{i,j}$	Tensor parallelism degree for device mesh from cluster j in stage i

Table 4 presents the complete set of notations used in our problem formulation. We formulate the cross-cluster auto-parallelism problem as the following nested optimization:

$$\min_{\mathcal{P}_1,\mathcal{P}_2} \quad \mathcal{T}(\mathcal{P}_2 \circ \mathcal{P}_1(\theta)) \tag{3}$$

$$\text{s.t.} \quad \sum_{i=1}^{p} n_{i,j} m_{i,j} \le N_j M_j, \quad \forall j \in \{1, \dots, C\} \tag{4}$$

$$\sum_{j=1}^{C} n_{i,j} m_{i,j} \ge 1, \quad \forall i \in \{1, \dots, p\} \tag{5}$$

$$d_{i,j} c_{i,j} t_{i,j} = n_{i,j} m_{i,j}, \quad \forall i \in \{1, \dots, p\}, \ \forall j \in \{1, \dots, C\} \tag{6}$$

$$\mathcal{X}(\mathcal{P}_2 \circ \mathcal{P}_1(\theta)) \preceq (X_j)_{j=1}^{C} \tag{7}$$

$$p, q \ge 1, \quad p, q \in \mathbb{Z}_+ \tag{8}$$

$$n_{i,j}, m_{i,j}, d_{i,j}, c_{i,j}, t_{i,j} \in \mathbb{Z}, \quad \forall i \in \{1, \dots, p\}, \ \forall j \in \{1, \dots, C\} \tag{9}$$

where:

• $\theta = \left\{B, s, h, \mathcal{O}, \mathcal{S}, \mathcal{R}, \{(N_j, M_j)\}_{j=1}^{C}, \{X_j\}_{j=1}^{C}\right\}$ represents the model configuration, cluster information, and training settings.

• $\mathcal{P}_1 : \Theta \to \Pi_1$ is a mapping that partitions the operator set $\mathcal{O}$ into p stages and determines the number of micro-batches q. Each stage i contains operators $(o_{l_i}, \dots, o_{r_i})$:

$$\mathcal{P}_1(\theta) = \left\{p, q, \{(o_{l_i}, \dots, o_{r_i})\}_{i=1}^{p}\right\} \tag{10}$$

• $\mathcal{P}_2 : \Pi_1 \to \Pi$ is a mapping that augments the pipeline partitioning with device allocation and intra-parallelism strategies. For each stage i, it assigns device meshes $(n_{i,j}, m_{i,j})$ from cluster j and specifies the intra-parallelism degrees $(d_{i,j}, c_{i,j}, t_{i,j})$:

$$\mathcal{P}_2 \circ \mathcal{P}_1(\theta) = \Big\{p, q, \{(o_{l_i}, \ldots, o_{r_i})\}_{i=1}^{p}, \{\{(n_{i,j}, m_{i,j}), (d_{i,j}, c_{i,j}, t_{i,j})\}_{j=1}^{C}\}_{i=1}^{p}\Big\} \quad (11)$$

• $\mathcal{T} : \Pi \to \mathbb{R}_+$ is the latency cost model that maps a complete parallelism strategy to training iteration latency. $\mathcal{X} : \Pi \to \mathbb{R}^C$ is the memory cost model that outputs the maximum memory consumption for any device from each cluster.

B.2 Constraint Interpretation

• Constraint (4) ensures device allocation does not exceed cluster capacity, reflecting the fundamental resource limitation in training environments.
• Constraint (5) requires each stage to have at least one device, ensuring computational feasibility for each pipeline stage.
• Constraint (6) enforces consistency between device mesh size and intra-parallelism degrees, maintaining the mathematical coherence of tensor partitioning where the product of parallelism degrees must equal the total number of devices allocated to a stage.
• Constraint (7) prevents out-of-memory issues, where $\preceq$ denotes element-wise inequality.
• Constraints (8) and (9) specify that all variables are non-negative integers.

B.3 Principle-Guided Search Space Pruning

Table 5. Principle-guided constraints on the search space

Principle	Constraint
P1	$l_i \geq 1, \forall i \in [1, p]$ (layer-level granularity)
P2	$\forall i \in [1, p], \exists! j \in [1, C] : n_{i,j} m_{i,j} > 0$
P3	Consecutive stage allocation (see Eq. 12)
P4	Uniform intra-parallelism (see Eq. 13)

Based on the four principles described in § 4.1, we significantly reduce the search space of the original optimization problem. Following **Principle #1**, we replace the operator-level stage definition $\{(o_{l_i}, \ldots, o_{r_i})\}_{i=1}^{p}$ with layer-level granularity $\{l_i\}_{i=1}^{p}$, where l_i represents the number of complete Transformer layers in stage i. The principle-constrained search space $\mathcal{P}_{\text{pruned}}(\theta) \subset \mathcal{P}_2 \circ \mathcal{P}_1(\theta)$ is then defined by the constraints in Table 5.

Table 6. Basic and advanced features based on domain knowledge

Symbol	Description
	Basic Features
s	Sequence length
h	Hidden size ($h = h_n \times h_d$)
h_n	Number of attention heads
h_d	Head dimension
c, t	CP, TP degrees
$b = \frac{B}{q \cdot d}$	Local micro-batch size per device (global batch size divided by micro-batches and DP degree)
e	Bytes per element (e.g., 1 for FP8, 2 for BF16)
	Derived Advanced Features
u	Input tensor elements per local micro-batch: $u = b \cdot s \cdot h$
F_{layer}	FLOPS per device per local micro-batch per layer: $F_{layer} = \frac{24uh+4us}{c \cdot t}$
M_{layer}	Peak memory per device per local micro-batch per layer: $M_{layer} = \frac{(12h^2+8u+4bsh_n) \cdot e}{c \cdot t}$
C_{tp}	TP communication volume per device per local micro-batch per layer: $C_{tp} = 2(t-1)\frac{u \cdot e}{c \cdot t}$
C_{cp}	CP communication volume per device per local micro-batch per layer: $C_{cp} = \frac{6(c-1)}{c}\frac{u \cdot e}{c \cdot t}$
R_{cm}	Compute-to-memory ratio: $R_{cm} = \frac{F_{layer}}{M_{layer}}$
R_{cc}	Compute-to-communication ratio: $R_{cc} = \frac{F_{layer}}{C_{tp}+C_{cp}}$

The formal definitions of P3 and P4 are:

P3: $\exists \sigma \in S_C, \exists 0 = e_0 < e_1 < \cdots < e_C = p$ s.t.
$$\forall k \in [1, C], \forall i \in (e_{k-1}, e_k] : n_{i,\sigma(k)} m_{i,\sigma(k)} > 0 \quad (12)$$
P4: $\forall k \in [1, C], \forall i_1, i_2 \in (e_{k-1}, e_k]$:
$$(d_{i_1,\sigma(k)}, c_{i_1,\sigma(k)}, t_{i_1,\sigma(k)}) = (d_{i_2,\sigma(k)}, c_{i_2,\sigma(k)}, t_{i_2,\sigma(k)}) \quad (13)$$

where S_C denotes the set of all permutations of $\{1, 2, \ldots, C\}$, σ represents a cluster assignment permutation, and e_k defines the ending stage index for cluster k under this assignment.

C Supplementary Explanation for Cost Model

C.1 Basic and Advanced Features for Random Forest Model Training

To further improve prediction accuracy, we incorporate domain knowledge to engineer informative features beyond the basic features and derive advanced features that capture key performance characteristics, including computational intensity, memory requirements, communication overhead, and their trade-offs. Representative basic and derived features are listed in Table 6.

C.2 Memory Cost Model

We build our memory cost model through theoretical analysis following previous work [37, 41, 74]. Our approach extends existing homogeneous cluster memory models to support heterogeneous cross-cluster deployments and each pipeline stage can use different intra-parallelism strategies.

The memory estimation framework accounts for four main components: weight memory, optimizer states, gradients, and activations. For weight memory, we calculate requirements for attention matrices, MLP projections, layer normalization parameters, and embedding tables, considering tensor parallelism sharding across TP groups and DeepSpeed ZeRO Stage 3 weight distribution [59] across data parallel groups when enabled. Optimizer state and gradient memory calculations assume the Adam optimizer [33] with master weights, momentum, and variance terms. We support different ZeRO [58] stages where Stage 1 shards optimizer states, Stage 2 additionally shards gradients, and Stage 3 shards weights across data parallel groups. Activation memory estimation considers intermediate computations during forward and backward passes, accounting for different activation recomputation strategies that trade computation for memory savings. We incorporate Flash Attention [13] optimizations and context parallelism where sequence dimensions are sharded across devices. For pipeline parallelism, each stage stores activations for multiple micro-batches according to the scheduling strategy used.

For heterogeneous deployments, we consider different memory capacities across cluster types. The framework validates configurations against each cluster's hardware constraints. When configurations exceed memory limits, the model returns infinite cost to indicate infeasibility, guiding the search algorithm toward viable parallelization strategies.

C.3 Random Forest Model Accuracy

For each device type (i.e., each cluster), we train a random forest model to predict single Transformer layer forward execution latency. We use 80% of the profiling data as training data and 20% as test data. The prediction results are illustrated in Fig. 14.

D Solution for exp3 Given by HetAuto

The exact solution for exp3 given by HetAuto is illustrated in Fig. 15. The pipeline parallelism degree is 18. Since each cluster adopts the same data parallelism degree, the outer DP is 8 and inner DP is 1 for each stage. We can see that different clusters adopt different intra-parallelism strategies. For the H20 cluster, it adopts intra-parallelism $(d, c, t) = (8, 1, 1)$, so each stage contains 8 H20 GPUs. For the H800 cluster, it adopts intra-parallelism $(d, c, t) = (8, 1, 2)$, so each stage contains 16 H800 GPUs. For the A100 cluster, it adopts intra-parallelism $(d, c, t) = (8, 2, 4)$, so each stage contains 64 A100 GPUs. For the Ascend A2 cluster, it adopts intra-parallelism $(d, c, t) = (8, 1, 8)$, so each stage contains 64 Ascend A2. Although the number of devices per stage varies significantly, load balance is achieved through layer distribution to ensure uniform latency across all pipeline stages. The cluster permutation follows a memory capacity descending order due to the descending memory consumption pattern in the 1F1B pipeline. Additionally, the last pipeline stage contains 2 fewer model layers because it needs to compute the loss, which incurs additional overhead. This solution demonstrates HetAuto's ability to effectively handle complex heterogeneous environments.

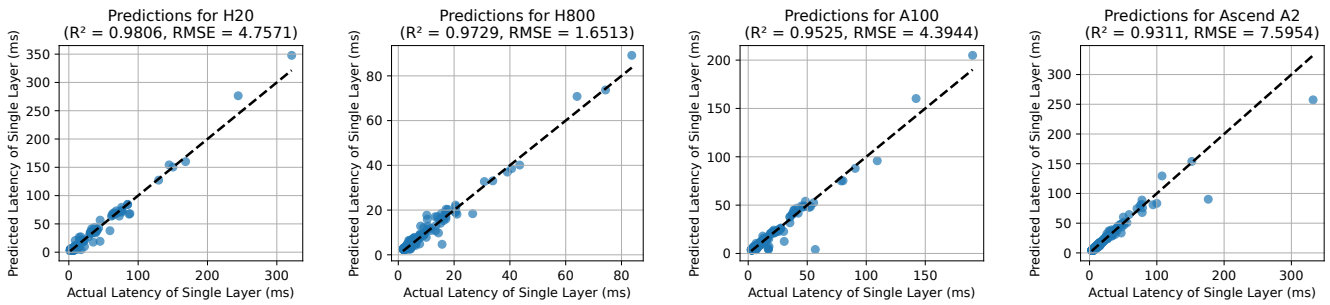
E Resharding Strategies Comparison

We compare the three resharding strategies introduced in § 5.2. In our scenario, resharding only occurs among adjacent stages across clusters, as the intra-parallelism strategy is identical within each cluster. We test resharding strategies across two clusters, each containing 16 NVIDIA A100-80GB GPUs (2 nodes × 8 GPUs/node), representing one pipeline stage. Intra-cluster bandwidth is either 100 Gbps InfiniBand (inter-node) or 600 GB/s NVLink (intra-node). Inter-cluster bandwidth is approximately 10 Gbps measured by `iperf3`.

Assume outer DP degree $d^{outer} = 1$, which means in each pipeline stage (cluster), $d^{inner} \cdot c \cdot t = 16$, where d^{inner} is inner DP degree, c is context parallelism degree, t is tensor parallelism degree. As illustrated in § 5.2, the tensor layout on each device is $[\frac{b'}{d^{inner}}, \frac{s}{ct}, h]$, where $b' = \frac{b}{d^{outer}}$. Hence, in the resharding scenario, we only care about the value of d^{inner} and the ct product. In our experiment, we vary all possible $(d_1^{inner}, (ct)_1, d_2^{inner}, (ct)_2)$ combinations (subject to $d_1^{inner} \cdot (ct)_1 = 16$, $d_2^{inner} \cdot (ct)_2 = 16$, and $d_1^{inner} \neq d_2^{inner}$ otherwise no need for resharding), resulting in 20 different parallelization combinations for comprehensive evaluation. We set the global batch size $B = 16$ and number of micro-batches $q = 1$, that is to say, the micro-batch size $b = \frac{B}{q \cdot d^{outer} \cdot d^{inner}} = \frac{16}{d^{inner}}$ varies between 1 and 16. We set sequence length $s = 8192$, hidden size $h = 4096$, and use bfloat16 tensor format. For each resharding operation, we test 10 iterations (with the first 5 iterations as warmup) and record the average latency. The results are illustrated in Fig. 16 as box plots showing statistical distributions across all 20 parallelization combinations. In the box plots, the box represents the interquartile range (IQR) from the 25th to 75th percentiles, and the whiskers extend to the minimum and maximum values.

For strategy 1, its performance is unstable. As we analyzed, this strategy results in numerous small P2P transfers that cause fragmented bandwidth usage and network congestion, especially when the tensor layouts between consecutive stages vary significantly. The slowest latency (2107.84 ms) comes from the resharding requirement from $(d_1^{inner}, (ct)_1) = (16, 1)$ to $(d_2^{inner}, (ct)_2) = (1, 16)$, which requires 256 P2P transfers.

For strategy 2, its performance is stable but slow on average. The stable performance comes from the unchanged communication pattern: the first stage gathers all tensor slices on the first rank in stage 1, then uses a single P2P to transfer the complete tensor to the first rank in stage 2, then the first rank scatters tensor slices to other ranks in stage 2.



(a) NVIDIA H20-141GB **(b)** NVIDIA H800-80GB **(c)** NVIDIA A100-80GB **(d)** Ascend A2-64GB

Figure 14. Random forest model prediction accuracy for different device types.

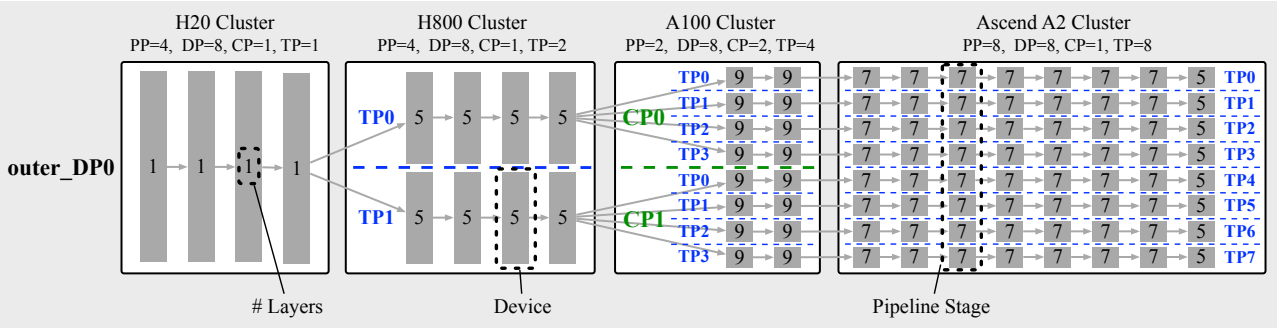


Figure 15. The exact solution for exp3 given by HetAuto. The number of micro-batches is 64. We only plot the pipeline for outer DP 0.

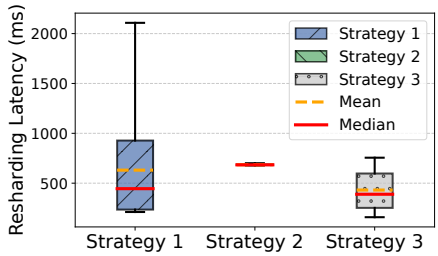


Figure 16. Resharding latency for different strategies across 20 parallelization combinations.

The pattern does not change with intra-parallelism strategy. This strategy creates a bottleneck on the first rank, which degrades its performance. Moreover, it is more likely to incur OOM issues than the other two resharding strategies.

Strategy 3 achieves a good balance between strategies 1 and 2, obtaining the fastest average resharding latency and avoiding severe performance fluctuations.