H2: Towards Efficient Large-Scale LLM Training on Hyper-Heterogeneous Cluster over 1,000 Chips

Ding Tang
Shanghai Artificial Intelligence Laboratory
Shanghai, China
tangding@pjlab.org.cn

Jiecheng Zhou
Shanghai Artificial Intelligence Laboratory
Shanghai, China
zhoujiecheng@pjlab.org.cn

Jiakai Hu
Shanghai Artificial Intelligence Laboratory
Shanghai, China
hujiakai@pjlab.org.cn

Shengwei Li
Shanghai Artificial Intelligence Laboratory
Shanghai, China
lucasleesw9@gmail.com

Huihuang Zheng
Shanghai Artificial Intelligence Laboratory
Shanghai, China
zhenghuihuang@pjlab.org.cn

Zhilin Pei
Shanghai Artificial Intelligence Laboratory
Shanghai, China
peizhilin@pjlab.org.cn

Hui Wang
Shanghai Artificial Intelligence Laboratory
Shanghai, China
wanghui@pjlab.org.cn

Xingcheng Zhang
Shanghai Artificial Intelligence Laboratory
Shanghai, China
zhangxingcheng@pjlab.org.cn

Abstract

Recent advancements in large language models (LLMs) necessitate extensive computational resources, prompting the use of diverse hardware accelerators from multiple vendors. However, traditional distributed training frameworks struggle to efficiently utilize hyper-heterogeneous clusters comprising thousands of chips due to significant disparities in software stacks, operator implementations, communication libraries, and hardware capabilities. To address these challenges, we propose H2, which stands for HyperHetero and is a systematic framework enabling efficient training of LLMs on clusters with over 1,000 heterogeneous chips. H2 incorporates DiTorch[1], a unified PyTorch-compatible interface ensuring program consistency across chips, and DiComm, a device-direct RDMA communication library optimized for heterogeneous environments. Furthermore, we introduce HeteroPP with HeteroAuto, an adaptive pipeline parallelism strategy that dynamically balances computational load, memory limitations, and communication overhead. Evaluations on a 100-billion-parameter LLM demonstrate that our approach consistently achieves a superlinear speedup, outperforming baseline homogeneous training solutions by up to 16.37% in our experiments. These findings validate the feasibility and efficiency of hyper-heterogeneous training at unprecedented scales.

1 Introduction

Recent advances in artificial intelligence have led to the fast development of large-scale models such as GPT-4[2], LLaMA[36], and DeepSeek[13, 21], which require substantial computational resources. The training of these models typically involves prolonged operation on expansive computing clusters to ensure convergence and optimal performance. Historically, such clusters have been built around single-chip servers[3]. However, as hardware requirements continue to escalate, numerous semiconductor companies have introduced their own distinctive chip designs[27], and each chip

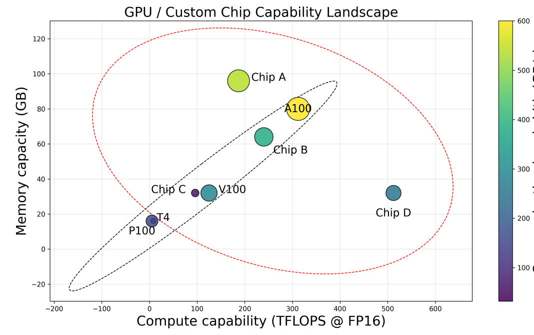

Figure 1. Comparison of chip specifications between capability-incremental and our hyper-heterogeneous scenario.In traditional heterogeneous scenarios, as indicated by the black dashed circles in the figure, chips show a trend of increasing capabilities in computation, communication, and memory. In contrast, in hyper-heterogeneous scenarios, as indicated by the red dashed circles in the figure, the capabilities of chips in these three aspects do not follow any specific pattern.

[1]https://github.com/DeepLink-org/ditorch

is developed according to proprietary design principles, resulting in both **software isolation and different hardware specifications**.

The real-world application of heterogeneous chips is steadily becoming a reality. First, relying solely on a single type of chip may be insufficient due to limitations in production capacity or the specific suitability for certain computational tasks[22]. Consequently, many organizations are now deploying heterogeneous clusters composed of multiple types of chips[38]. Furthermore, with the ongoing procurement of new chips over time[8, 10, 35], it is urgently important to integrate legacy chips with newer ones efficiently to reduce idle costs.

Hyper-heterogeneous computing refers to an emerging computing paradigm characterized by significant hardware and software diversity within large-scale computing clusters. Specifically, hyper-heterogeneous environments exhibit three key distinguishing features. First, the hardware specifications—including computational power, memory capacity, and communication bandwidth—of different chips can vary drastically without clear patterns or predictable relationships. Second, chips from different vendors typically exhibit pronounced software isolation, utilizing separate software stacks and distinct communication libraries, thereby complicating interoperability and software integration. Third, the quantities of different chip types within a single cluster may be highly imbalanced, further increasing complexity in workload scheduling and resource management. Unlike conventional heterogeneous computing systems, which usually involve fewer chip variants with more uniform programming models and communication protocols, hyper-heterogeneous systems are characterized by a significantly higher degree of complexity and integration challenges.

Existing heterogeneous approaches have predominantly concentrated on designing training algorithms for hardware setups that often involve different generations of chips from the same vendor[15, 37]. For example, Metis searches for heterogeneous parallelism strategies on Nvidia T4, P100, and V100, and Whale accelerates heterogeneous parallel training with Nvidia P100 and V100. This design choice leads to two notable implications. First, there is an inherent performance gap between pairs of chips; one type typically outperforms the other across a range of specifications such as computational power, memory capacity, and communication bandwidth as shown in Figure 1. This discrepancy can significantly impact workload distribution and overall system efficiency. Second, these setups benefit from a naturally unified software stack[25] and communication library[26] (e.g., CUDA and NCCL), which minimizes the need for adjustments that would otherwise be required to bridge isolated software environments within heterogeneous systems.

In addition, most heterogeneous training approaches have been designed for small-scale scenarios or validated only on clusters with a limited types of chips[40]. Consequently, their scalability and performance in larger, more diverse computing environments remain largely unproven.

In this work, we address the scenario of efficiently training extremely large models in hyper-heterogeneous computing environments. To uniformly leverage chip resources from different vendors while ensuring scalability, we highlight the necessity of developing new systems and algorithms specifically designed for hyper-heterogeneous scenarios. Such systems must effectively address the following issues:

First, the technical isolation among different types of chips are considered from three aspects: software, communication and operators. The **software isolation** comes from the fact that chips from different vendors are supported by distinct technology stacks and, in many cases, their own training frameworks. Although some chips offer compatibility via adaptations to popular interfaces[1, 39] (e.g., PyTorch), semantic inconsistencies between these solutions pose significant challenges for developers aiming for a unified development environment. Furthermore, the issue of **communication isolation** adds another layer of complexity. Different chips utilize diverse internal communication libraries, and the variations in server topologies—such as the number of network interface cards and respective bandwidths—make it difficult to develop a standardized communication library that can be uniformly applied[16]. Finally, **operator isolation** arises from the differing hardware design logics across chips. As each chip implements its own acceleration strategies and operator development methodologies, there often remain discrepancies in the support and precision of common operators. Although most chip manufacturers have implemented adaptations for widely-used large-scale models, gaps persist in both operator availability and computational accuracy alignment.

Second, heterogeneous training in **large-scale** scenarios brings several challenges. Driven by the rapid advancement of large-model techniques[9, 29], the computational resources necessary for training a state-of-the-art base model have grown extraordinarily demanding[24]. This scale of computational requirement naturally results in an extremely large space for parallel execution strategies, making the optimization process considerably complex. Therefore, it is imperative not only to develop efficient parallel algorithms, but also to ensure that the algorithmic search process itself remains computationally efficient to reduce the high cost of large-scale computing resources[18, 41].

To address the two aforementioned issues, we have designed the H2 framework, which includes DiTorch, DiComm and HeteroPP. DiTorch extends PyTorch's interface to realize user-transparent support for heterogeneous computation, requiring only a one-line code modification to the original training program, and provides a suite of tools to ensure the alignment of computational precision among different chips. DiComm implements device-direct RDMA communication between heterogeneous GPUs and optimizes communication via hardware-topology-aware methods. HeteroPP is an efficient heterogeneous training framwork which employs pipeline parallelism to connect different types of chips, and introduces an efficient automatic search algorithm for parallelism strategy. In our evaluation of a 100B LLM training on a cluster comprising 1,024 chips of four distinct architectures, our H2 framework achieved a superlinear speedup in comparison to conventional homogeneous training approaches.

In summary, our contributions are as follows:

- We introduce a novel training paradigm termed hyper-heterogeneous training to address the evolving landscape of heterogeneous computing.
- We design and develop DiTorch and DiComm to break the technical isolation among different types of chips and achieve user-transparent heterogeneous computation and communication with only a one-line modification.
- We propose the HeteroPP framework, which implements highly efficient heterogeneous training algorithms, alongside HeteroAuto, a robust and efficient automatic strategy-search algorithm for HeteroPP framework under hyper-heterogeneous scenarios.
- We validate the efficiency and training stability of H2 system through experiments on training a 100B-parameter large model.

2 Background

In this section, we first present the software stack for LLM pre-training. In addition, we discuss about various parallelization strategies. Finally, we briefly introduce the characteristics of hyper-heterogeneous clusters.

2.1 LLM Pre-training Software Stack

Large-scale model pre-training often entails millions of iterations and involves over a hundred GPUs. As shown in Figure 2. first, users provide the training data and model to the pre-training framework, which efficiently implements data processing and parallelization strategies through its distributed design. Then, these distributed pre-training frameworks employ PyTorch's interfaces to realize the parallel strategies. Finally, PyTorch leverages the high-performance computing and communication libraries specific to each chip to execute the computation and communication functions.

2.2 Distributed Training

Our work focuses on 3D parallelism [32], including pipeline parallelism[11, 14, 23], tensor parallelism[6, 32], and data parallelism[20], which is the most commonly used parallel strategy in large model pre-training. ZeRO-1 [30] is enabled in default.

Pipeline Parallelism: A large-scale model is split layer-wise into multiple stages, with each stage comprising several

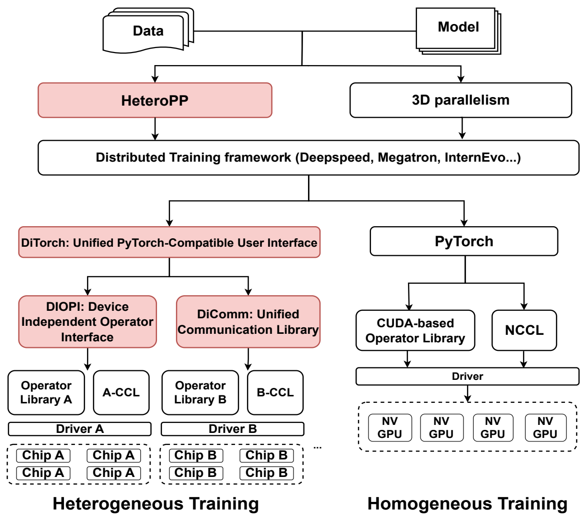
Data
Model
HeteroPP
3D parallelism
Distributed Training framework (Deepspeed, Megatron, InternEvo...)
DiTorch: Unified PyTorch-Compatible User Interface
PyTorch
DIOPI: Device Independent Operator Interface
DiComm: Unified Communication Library
CUDA-based Operator Library
NCCL
Operator Library A
A-CCL
Operator Library B
B-CCL
Driver
Driver A
Driver B
Chip A
Chip B
NV GPU
Heterogeneous Training
Homogeneous Training

Figure 2. LLM Pre-training Software Stack. Red parts represent our work.

contiguous layers. These stages are then assigned to different GPUs. Pipeline parallelism works by dividing the data into multiple micro-batches, allowing the stages to operate in a pipelined fashion and thus fully leveraging the computational resources of all GPUs.

Data Parallelism: In large model pre-training, training data can be distributed across multiple GPUs, with each GPU maintaining a complete copy of the model parameters to perform forward and backward computations. Once gradients are computed, an all-reduce operation is used to synchronize the parameters across GPUs.

Tensor Parallelism: For large Transformer-based models, tensor parallelism is an effective method to enhance training efficiency. In this approach, parameter tensors are partitioned horizontally among multiple GPUs, thereby distributing both the memory load and the computational workload for forward and backward passes across the GPUs. When tensor parallelism is applied, each Transformer block's forward pass requires two all-reduce operations to reconstruct the complete input values.

2.3 Hyper-Heterogeneity Characteristics

Training an extremely large model requires vast resources, and these resources are often needed concurrently during training. In industrial settings, computing clusters typically consist of various Chips from different vendors to meet both training and inference requirements. Utilizing heterogeneous chip resources can satisfy the demands of training tasks and experiments without having to wait for a homogeneous set of resources to become fully available, thereby accelerating

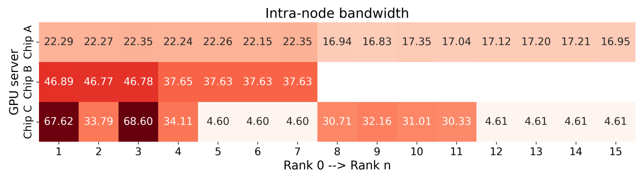

Figure 3. Intra-Node Bandwidth Performance in Different GPU Servers

model training and experimentation. Compared to homogeneous clusters and standard heterogeneous environments, Hyper-Heterogeneity has the following characteristics:

Distinct Software Stacks for Each Vendor's Chip: Due to the "isolation" between hardware design and the corresponding software stacks, the same operator implemented on different chips can produce varying degrees of precision. For example, in matrix multiplication, different vendors may employ unique data layouts and accumulation orders for operators, leading to discrepancies in the final results. Automatically detecting and resolving these precision issues is a fundamental challenge. Moreover, the software stack for large model pre-training is inherently complex, with each vendor typically focusing only on its proprietary libraries; breaking the technical "isolation" between these stacks is the cornerstone of heterogeneous parallel training.

Heterogeneity in Computational, Communication, and Storage Capabilities Across Different Chips: Machines from different vendors can exhibit unpredictable differences in computing, communication, and storage capacities. For instance, as shown in Figure 1, a Chip A might have up to 96GB of memory but only deliver 182 Tflops at FP16, whereas Chip C could achieve 512 Tflops at FP16 despite having only 32GB of memory. Such discrepancies lead to divergent choices in parallelization strategies and workload partitioning during training. Because different chips need different parallel strategies, selecting an optimal combination for large-scale, multi-vendor configurations becomes challenging. The rapid proliferation of GPU models and expanding cluster sizes call for a scalable parallel approach.

Complex Intra-node Topologies: Popular GPU servers often utilize high-speed intra-node interconnections[17] (e.g., NVLink) to achieve high and homogeneous bandwidth within a node. However, some GPU servers either lack or do not fully implement high-speed intra-node connections, resulting in varying communication capabilities among GPUs within the same node. This discrepancy an exponential search space, rendering the Metis algorithm extremely inefficient in hyper-heterogeneous scenarios.

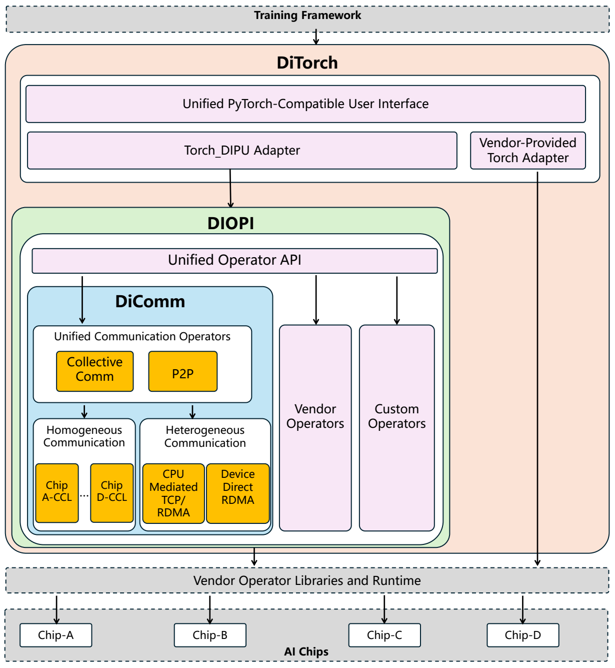

Figure 4. System Overview of DiTorch and DiComm.

3 Breaking Software and Communication Barrier

To address the software and communication isolation challenges outlined previously, we introduce ***DiTorch*** and ***DiComm*** in this section, whose system overview are shown in Figure 4. DiTorch standardizes operator libraries and runtime environments and provides an unified PyTorch-compatible programming interface across heterogeneous chips. DiComm implements efficient RDMA communication among diverse chips through a consistent communication API. This design simplifies the integration of heterogeneous hardware into PyTorch-based frameworks, allowing algorithm developers to focus on higher-level optimizations rather than low-level hardware intricacies.

3.1 DiTorch

DiTorch is built upon two key principles. First, we employ PyTorch [4] as the unified programming interface layer, a decision justified by its widespread adoption as the de facto standard—owing to its ease of use, flexibility, and high efficiency. Given that PyTorch underpins numerous distributed training systems, such as Megatron[32], DeepSpeed[31] and InternLM[34], it serves as an ideal foundation for ensuring consistency across heterogeneous hardware platforms. Second, acknowledging that large-scale model computations are fundamentally comprised of numerous operators, we have developed a bottom-up precision alignment pipeline tailored for heterogeneous chips. This approach first enforces numerical consistency at the operator level across different chips and then achieves an end-to-end, model-level precision alignment, thereby ensuring the correctness and stability of training large models in hyper-heterogeneous environments.

3.1.1 Torch Adaptation.

Contemporary training frameworks predominantly rely on PyTorch; however, the runtime interfaces and operator libraries available for heterogeneous chips differ considerably. To address these disparities, DiTorch provides a unified programming interface, thereby simplifying the integration of multiple hardware operator libraries. In practice, merely appending a single line of code enables users to utilize heterogeneous chips in a manner fully consistent with the standard PyTorch distribution.

DiTorch Example for Heterogeneous Hardware

```
>>> import torch
>>> import ditorch
>>> #Unified device name & dispatch key
>>> x = torch.randn(4,4,device="device")
>>> #Precision-verified operator on all chips
>>> y = x + x
```

DiTorch incorporates two major strategies to enhance interoperability and efficiency across diverse AI hardware platforms. Initially, the Torch Adapter may be supplied by the chip vendor if available; otherwise, one can directly employ the ***Device-Independent Process Unit (DIPU)***, a Torch Adapter developed within DiTorch. DIPU harmonizes the underlying runtime systems from various vendors into unfied APIs, covering functionalities such as stream and memory management, device control, and kernel launching, thus achieving semantic uniformity for auxiliary (non-operator) operations at the Torch layer. Besides, the ***Device-Independent Operator Interface (DIOPI)***[2] is utilized to bridge the unified Torch operator API with vendor-tailored operator libraries, ensuring consistent semantic interpretation across operators. By establishing a formal computational standard, DIOPI decouples the upper framework layers from the underlying hardware, allowing independent evolution in both realms and promoting the reuse of operators across different systems. Currently, over 300 standardized operator interfaces have been implemented in DIOPI.

3.1.2 Precision Alignment and Performance Analysis.

DiTorch is furnished with a comprehensive set of operator analysis tools aimed at ensuring numerical accuracy and optimizing performance on distinct AI chip architectures. The suite encompasses modules for both offline and real-time precision evaluation, profilers for execution time analysis and optimization, and mechanisms designed to detect overflow issues in individual or all operators. In addition,

[2]https://github.com/DeepLink-org/DIOPI

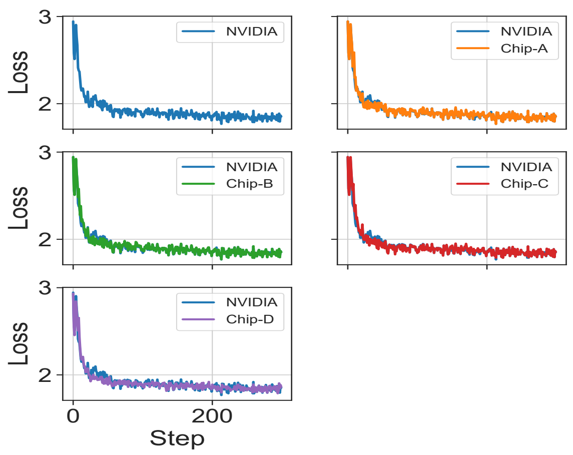

Figure 5. DiTorch's precision alignment across Chips A, B, C, and D compared to the NVIDIA A100.

Table 1. Observed Mean Relative Error of training loss for the assessed AI chips.

	Chip-A	Chip-B	Chip-C	Chip-D
MRE	0.391%	0.477%	0.584%	1.215%

DiTorch integrates input-output capture capabilities that extract authentic training data, greatly facilitating debugging and tuning processes. These tools are essential for achieving high-caliber training outcomes for large language models on the supported AI platforms.

By leveraging these instruments, DiTorch achieves precision congruity on Chips A, B, C, and D relative to the NVIDIA A100 during large language model training. As shown in Figure 5, we conducted training of a 20-billion parameter model for 300 iterations using the same dataset. The precision alignment criterion is satisfied if the Mean Relative Error (MRE) remains below 1.5%:

$$\frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i-\hat{y}_i}{y_i}\right| < 1.5\%,$$

where y_i represents the loss measured at iteration i on the NVIDIA A100 GPU, and $\hat{y}_i$ corresponds to the loss on the evaluated AI chip. In our experiments, we set $n = 300$. Table 1 lists the observed MRE values for Chips A, B, C, and D.

3.2 DiComm

In this work, we introduce DiComm, a unified communication library that overcomes the isolation and inefficiency challenges in complex heterogeneous environments. Leveraging the libibverbs library, DiComm facilitates RDMA communications across various chip architectures and supports both homogeneous and heterogeneous chip-to-chip interactions. DiComm is compatible with protocols including TCP

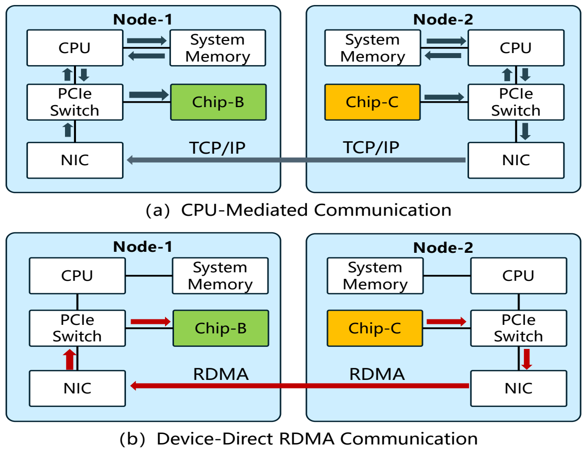
Node-1
CPU
System Memory
PCIe Switch
Chip-B
NIC
TCP/IP
Node-2
Chip-C
(a) CPU-Mediated Communication
RDMA
(b) Device-Direct RDMA Communication

Figure 6. Comparison of CPU-mediated and device-direct communication methods.

and RDMA, and implements two distinct communication paradigms: CPU-mediated and device-direct. As illustrated in Figure 6, we provide detailed comparisons of these protocols and their underlying techniques. Although its primary focus is high-performance peer-to-peer communication, DiComm also incorporates collective communication primitives (e.g., allreduce and broadcast) via a combination of send/receive operations and native communication operators.

CPU-mediated Strategy. As depicted in Figure 6, the CPU-mediated mechanism in DiComm initiates by transferring data from the source AI chip to the host's main memory. Thereafter, a CPU-focused communication framework—such as Gloo—is employed to relay this information to the destination node via TCP/IP or RDMA protocols. Following temporary storage in the host memory, the data is subsequently forwarded to the target AI chip. This operational structure ensures wide-ranging compatibility with diverse AI accelerators, making it both adaptable and broadly applicable.

Device-direct Strategy. In the device-direct method, each chip registers its local memory regions with an RDMA driver, thereby enabling an RDMA-enabled NIC to map physical addresses for remote access. An RDMA connection manager (e.g., `rdma_cm`) then coordinates the connection establishment by configuring queue pairs and exchanging memory region descriptors that incorporate the requisite keys and addresses. After the connection is set up, the NICs utilize their DMA controllers to execute direct read and write operations among device memories, thus bypassing the CPU and host memory entirely. This approach effectively shortens the data transmission path, curtails latency, and enhances overall communication efficiency. To further boost DiComm's performance, we deliberately select NICs that are particularly well-suited for inter-chip communications.

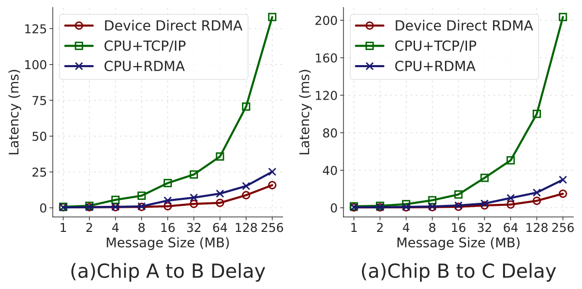

Figure 7. Comparison of cross Chip-Communication latency with different strategies.

3.2.1 DiComm Performance. DiComm successfully enables P2P communication between different chips using the device-direct strategy. As shown in Figure 7, device-direct RDMA reduces average latency by 9.94× compared to the conventional TCP/IP scheme, with speedups ranging from 1.79× to 16.0× depending on the message size.

Given that the GLOO backend in the PyTorch framework currently lacks support for RDMA and is limited to TCP, DiComm offers substantial performance enhancements for heterogeneous interconnects over the PyTorch GLOO approach.

4 Heterogeneous Parallelism

In this section, we delineate the key observations that underpin the design of ***HeteroPP***, a large-scale training algorithms for LLMs on heterogeneous chip architectures. Our analysis is motivated by three dimensions of hardware discrepancy—memory capacity, communication bandwidth, and computational power—coupled with the unbalanced distribution of chip counts and the expansive parallel and model scales intrinsic to modern language model training.

4.1 Key Observations for HeteroPP

In traditional homogeneous distributed training, we have the following observations and ideas how they can be applied to heterogeneous training:

Observation #1: *Both data parallelism and tensor parallelism require high-performance AllReduce and some other collective communication operators. In contrast, the point-to-point communication inherent in pipeline parallelism can be more readily overlapped with computation.*

In actual heterogeneous cluster environments, different heterogeneous chips are distributed across separate server nodes, resulting in inherently homogeneous intra-node configurations. Moreover, in our ultra-heterogeneous scenario, the underlying software stacks and hardware topologies vary for each type of chip, making the development of a unified high-performance collective communication operator extremely challenging. As the diversity of chip types increases, so do the engineering workload and complexity. Therefore, leveraging pipelined parallelism to connect different heterogeneous chips emerges as the most rational and efficient approach.

Observation #2: *Tensor parallelism is extremely sensitive to bandwidth and typically occurs within nodes with high bandwidth.*

Due to the varying internal designs and hardware topologies of heterogeneous chip nodes, there are differences not only in the number of chips but also in their intra-node interconnect configurations. In certain chip server environments, noticeable throughput degradation is observed when intra-node communication involves traversing NUMA boundaries or crossing PCIe switches. This variability in interconnect performance consequently influences the optimal range for configuring tensor parallelism dimensions.

Observation #3: *The efficiency of pipeline parallelism is closely related to the load balance of the tasks assigned to each pipeline stage.*

In homogeneous distributed training, pipeline load balancing is achieved through uniform layer sharding across each stage and consistent parallel dimensions. In heterogeneous mixed training, however, the computational and communication capabilities vary across different chip types. Consequently, non-uniform layer sharding and stage-specific adjustments of parallel dimensions become highly valuable. Additionally, configuration complexity can be reduced by applying identical settings to homogeneous chips while using distinct settings for heterogeneous chips.

Observation #4: *Within the pipeline parallelism, the early stages require a larger number of warmup micro-batches to ensure seamless computational flow to subsequent stages. This necessity imposes additional memory demands.*

Due to differences in memory specifications among heterogeneous chips, we can allocate chips with larger memory to the earlier stages of pipeline parallelism, while those with smaller memory can be assigned to the later stages. This not only reduces the need for recomputation but also expands the available parallel dimensional space for each chip type.

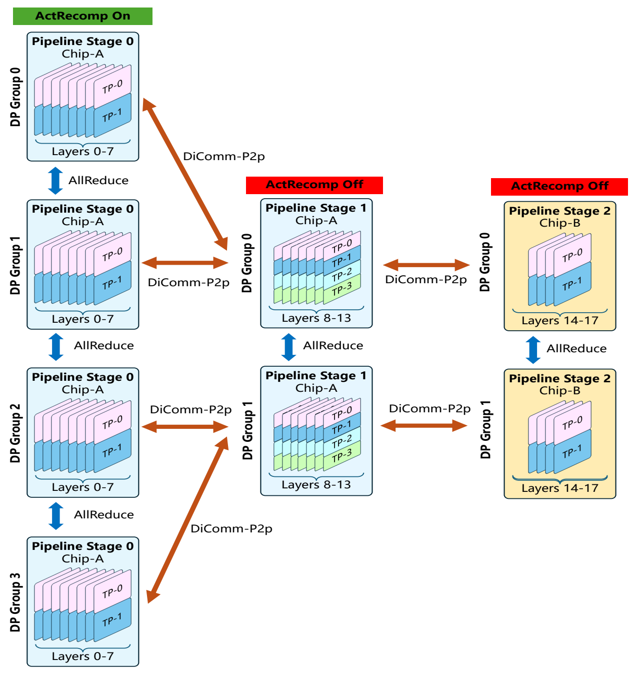

Figure 8. A simple example of HeteroPP. In this setup, two types of chips are used, and the model consisting of 18 layers is partitioned into three pipeline stages. At each pipeline stage, different configurations for data parallelism, tensor parallel dimensions, layer sharding and activation recomputation settings can be employed.

4.2 HeteroPP Design

In this work, we extend pipeline parallelism to heterogeneous AI Chip-Clusters for distributed LLM training. Figure 8 illustrates a simple example of the HeteroPP framework. In this example, two types of chips,A and B, are used to train an 18-layer model, with 16 chips A and 4 chips B. Each pipeline stage is composed entirely of a single chip type, and each chip type can be mapped to multiple pipeline stages. In our example, chip A occupies the first two stages, and chip B occupies the final stage. The mapping order corresponds to the chips' memory capacities in descending order. Within each pipeline stage, homogeneous chips are partitioned into multiple TP and DP groups, where TP sizes 2,4,2 and DP sizes 4, 2, 2 are used in three pipeline stages separately. Activation recomputation is enabled for the first pipeline stage to relieve memory stress and disabled for the last two pipeline stages to increase computation efficiency. 8, 6, and 4 layers are shared into three pipeline stages separately. P2P communication is facilitated by DiComm, which supports both a CPU-mediated strategy and a device-direct strategy. Gradient synchronization within DP groups is performed using AllReduce operations, constrained to AI chips of the same type.

As we can see, compared to homogeneous pipeline parallelism, HeteroPP introduces several distinct characteristics due to computational and memory capacity imbalances among different AI chips.

Based on Observation #1, the HeteroPP framework is architected so that each pipeline stage consists exclusively of homogeneous chip nodes, with heterogeneous nodes being strategically distributed across different stages. Additionally, drawing from Observation #4, we **map chips with larger memory capacities to the earlier stages of the pipeline**, while those with smaller memory are allocated to the later stages. This allocation strategy not only optimizes resource utilization but also enhances communication efficiency throughout the pipeline.

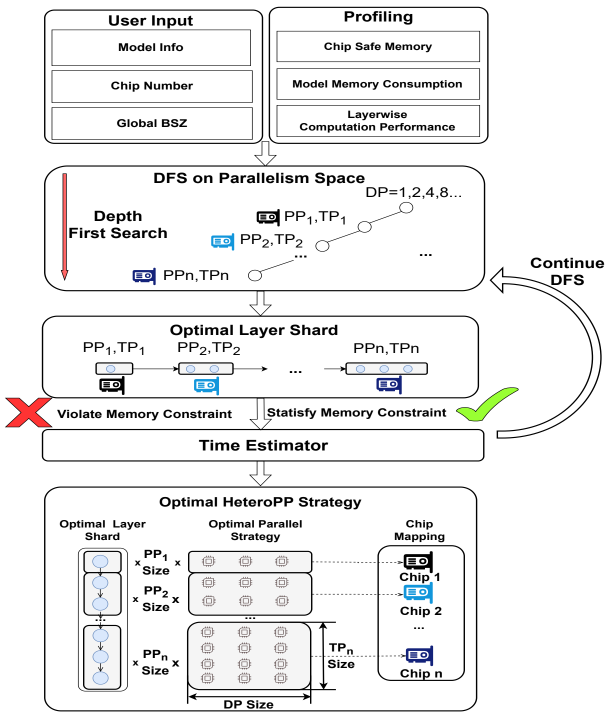

Figure 9. Overview of HeteroAuto.

Moreover, to achieve balanced pipeline workloads due to the Observation #3 and fully exploit the computing, communication, and memory capabilities of the chips, our system supports non-uniform task distribution in three key aspects. First, we enable **uneven partitioning of model layers**. This approach allows for the allocation of a greater number of layers to pipeline stages with stronger computational resources or to heterogeneous chips with larger memory capacities. Second, we allow for **flexible adjustments in the intra-stage parallel dimensions**, encompassing both data-parallel and tensor-parallel dimensions. This flexibility enables dynamic tuning of the number of chips per stage, as the total chip count is determined by the product of the data-parallel and tensor-parallel dimensions. Furthermore, **activation recomputation can be flexibly enabled or disabled** based on the memory requirements of each pipeline stage and the available chip memory.

Despite these differences, HeteroPP remains compatible with a variety of pipeline parallelism scheduling strategies, including Chimera [19], ZB-V [28] and ZeroPP [33].

4.3 Automatic Search for HeteroPP Strategy

We introduce ***HeteroAuto***, an automated search method for optimizing parallel strategy configurations in heterogeneous chip environments within the HeteroPP framework. Our approach tackles the dual challenges of **resource heterogeneity** and **communication overhead**. Specifically, HeteroAuto is designed to identify efficient heterogeneous training parallelization configurations for extremely large models on hyper-heterogeneous clusters. At the same time, we ensure that the search time remains within an acceptable range, balancing optimal performance with practical usability.

Besides the observations listed in previous section, our design is motivated by an extra factors: in training extremely large models, memory constraints often restrict the micro-batch size to 1, thereby limiting the potential for adjustments solely via data parallelism.

4.3.1 Search Space. Given an LLM with L Transformer layers, we train the model on a heterogeneous cluster comprising $\sum_{i=1}^{C} N_i$ AI chips, where C represents the number of different chip types, and N_i denotes the number of chips of type i. The global batch size is fixed at B. HeteroAuto explores the following decision variables to optimize overall training performance. It searches for $s_{pp,i}$, the number of pipeline stages assigned to chip type i determining the total pipeline parallelism degree as $s_{pp} = \sum_{i=1}^{C} s_{pp,i}$, and $s_{tp,i}$ which represents the tensor parallelism degree used by chip type i during training.

Given a total of L layers, HeteroAuto also determines the layer allocation strategy by searching for l_i, the number of layers assigned to chip type i. Consequently, the number of layers per pipeline stage on chip type i is given by $l_i/s_{pp,i}$. In this formulation, based on Observation #3, we assume that layers are ***non-uniformly*** distributed across different chip types and ***evenly*** distributed within the same chip type.

Additionally, HeteroAuto independently determines the recompute setting for each chip type. We denote this setting by r_i, where r_i can take the value of 0 or 1, corresponding to recompute being disabled or enabled, respectively. Furthermore, HeteroAuto determines s_{dp}, the data parallelism degree, which in turn defines the number of micro-batches used in pipeline parallelism as $b = B/s_{dp}$.

4.3.2 Cost Model. HeteroPP seeks an optimal solution that minimizes the estimated iteration time:

$$T = \min \left\{ \max_{1 \le i \le s_{pp}} \left(b \cdot T_i^{\text{comp}} + T_i^{\text{update}} + \alpha \cdot \sum_{j=1, j \ne i}^{s_{pp}} T_j^{\text{comp}} \right) \right\},$$

where T_i^{comp} denotes the per-pipeline-stage computation time of chip i for a single microbatch, including both forward, backward computation as well as recomputation overhead. Thus, T_i^{comp} can be further refined and expressed by the following equation, where $t_{s_{tp,i}}^{fwd}$, $t_{s_{tp,i}}^{bwd}$ and $t_{s_{tp,i}}^{recomp}$ represent layer-wise forward, backward and recomputation time with TP size $s_{tp,i}$ on chip i:

$$T_i^{\text{comp}} = ceil(\frac{l_i}{s_{pp,i}}) \cdot (t_{s_{tp,i}}^{fwd} + t_{s_{tp,i}}^{bwd} + r_i \cdot t_{s_{tp,i}}^{recomp})$$

Table 2. Notations used in HeteroAuto.

Notation	Description
C	Number of different chip types in the cluster.
N_i	Number of chips of type i, $N_i = s_{pp,i} \times s_{tp,i} \times s_{dp}$.
B	Global batch size.
L	Total number of Transformer layers in the model.
α	Bubble coefficient for pipeline schedule.
s_{dp}	Number of data parallelism degree.
b	Number of micro-batch, $b = B/s_{dp}$.
$s_{pp,i}$	Number of pipeline stages allocated to chip type i.
s_{pp}	Pipeline parallelism degree, $S_{pp} = \sum_{i=0}^{C-1} s_{pp,i}$.
$s_{tp,i}$	Tensor parallelism degree used by chip type i.
l_i	Number of layers assigned to chip type i.
r_i	Recomputation setting of chip type i.

T_i^{update} represents the per-pipeline-stage optimizer update time of chip i, which consists of the optimizer computation time and the gradient synchronization time that is not overlapped by the backward computation. T_i^{update} can be further refined and expressed by the following equation, where $t_{s_{dp},s_{tp,i}}^{update}$ represents layer-wise update time of chip i given tp size $s_{tp,i}$ and dp size s_{dp}:

$$T_i^{\text{update}} = ceil(\frac{l_i}{s_{pp,i}}) \cdot t_{s_{dp},s_{tp,i}}^{update}$$

In addition, there exist numerous pipeline parallelism strategies, each with distinct methods for computing the bubble time. Under large-scale training scenarios, the bubble time can be considered proportional to the computation time of a single microbatch. Therefore, we introduce a configurable bubble coefficient α. For instance, in zero-bubble pipeline parallelism approaches such as ZB-V[28], α is set to 0. In our work, since we employ the conventional 1F1B approach, we set α to be 1, and the computation formula for T the computation formula is equivalent to the one used in Metis[37]. Therefore, T represents the total iteration time, which accounts for the maximum computation time across all pipeline stages, the optimizer step time, and the pipeline bubble overhead.

We use an auto-profiler to profile the layer-wise performance of each chip, including $t_{s_{tp,i}}^{fwd}$, $t_{s_{tp,i}}^{bwd}$, $t_{s_{tp,i}}^{recomp}$ and $t_{s_{dp},s_{tp,i}}^{update}$ under various DP and TP sizes. Additionally, we will profile layer-wise memory consumption with and without activation recomputation.

Based on our empirical findings, we enforce the following requirements in our search for heterogeneous parallel strategies:

1. For pipeline stages associated with the same chip category, a uniform tensor-parallel degree must be employed, and each stage should be allocated an identical number of Transformer layers.
2. For every chip indexed by $i \in [1, C]$, the tensor-parallel degree $s_{tp,i}$ is constrained to powers of two (e.g., 1, 2, 4, 8, etc.). Additionally, to optimize communication efficiency, $s_{tp,i}$ is generally capped at TP_MAX_i, a limit determined by the number of chips available within a single node, NUMA domain, or PCIe switch depending on the hardware configuration.
3. The overall memory usage must remain within the safe capacity profiled for each individual chip.

4.3.3 Search Algorithm.

To thoroughly explore the vast decision space, we adopt a depth-first search (DFS) methodology outlined by the following procedures:

1. **Depth-First Search for Parallelism Space.** Select candidate values for data parallelism from the allowed set, ensuring that each candidate evenly divides the global batch size B. For every chosen value s_{dp}, compute the associated micro-batch count as $b = B/s_{dp}$. For each candidate s_{dp} and for every chip type i, choose a tensor parallelism size $s_{tp,i}$ from the set $\{1, 2, \ldots, TP\_MAX_i\}$, and then determine the corresponding pipeline parallelism size $s_{pp,i}$ using the relation $N_i = s_{pp,i} \times s_{tp,i} \times s_{dp}$. A DFS procedure then systematically explores these options across different chip types, ordered by descending memory capacity.
2. **Optimal Layer Sharding.** For every configuration represented as $(s_{dp}, b, \{(s_{pp,i}, s_{tp,i}, r_i) \mid i \in [1, C]\})$, perform a heuristic search to derive the best layer distribution. An initial assignment is computed by equalizing the compute time among chip groups. If this allocation exceeds the total number of layers L, an iterative refinement readjusts the per-group assignments while ensuring that memory usage remains within safe limits.
3. **Cost Estimation and Strategy Selection.** Among the configurations that meet memory constraints, estimate the execution time per iteration. The computation time for processing a single layer with one micro-batch on the i-th chip (given a specific tensor parallel configuration) is pre-profiled. The configuration that minimizes the total iteration time is then adopted as the optimal parallel strategy.

A **two-stage** procedure further refines the search for parallelism strategies, enhancing output efficiency with only a minimal increase in search overhead. In the first stage, HeteroAuto identifies a suitable data parallelism size exclusively. In the subsequent stage, a sizable group of homogeneous chips is partitioned into smaller subgroups, each treated as a distinct heterogeneous entity. These groups, combined with the DP size established earlier, are input into the algorithm once again. Additionally, an extra constraint can be enforced among groups of the same chip type: if group a precedes group b, then it must hold that $s_{tp,a} \geq s_{tp,b}$, thereby pruning

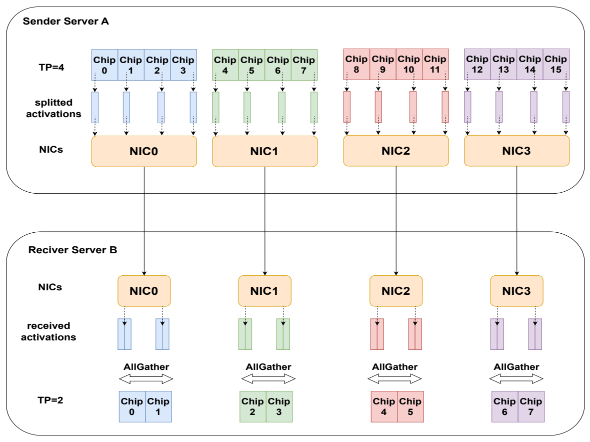

Figure 10. An example of topology-aware send/recv + all-gather activation resharding with TP size 4 on Chip-A server and TP size 2 on Chip-B server.

less promising candidates. This strategy allows for a more nuanced exploration of heterogeneous parallelism, ultimately leading to enhanced efficiency in parallel training.

5 Optimization

To enhance training efficiency in heterogeneous settings, we study activation resharding in parallel architectures. Here, activation resharding denotes transferring activation values between consecutive pipeline stages which can be mapped to different types of chips severs.

Under the assumption that each GPU server is equipped with only a single NIC and that the NIC's bandwidth is the primary bottleneck for cross-node activation resharding, previous work [42] has proposed a broadcast-based method for inter-node activation remapping. This approach minimizes the volume of data transferred between nodes and allows for overlapping data transfers among different GPUs within the same node. However, hyper-heterogeneous environments face additional challenges:

1. GPU servers of different chips have multiple NICs with varying counts and affinities.
2. PCIe links between PCIe switches and chips can limit bandwidth, requiring concurrent transmissions of multiple chips to saturate a single NIC.

To address these issues, we propose a topology-aware activation resharding strategy as shown in Figure 10.

First, we configure each chip server to assign a dedicated communication NIC to each chip based on its NIC affinity. This setup not only balances the load across the NICs but also minimizes the overall communication path length. The performance enhancement achieved by utilizing affinity NICs is evident in Table 3.

Table 3. Throughput comparison between NIC affinity and non-affinity on two heterogeneous servers using 8 chips for concurrent communication with message size 64MB.

Chips	Non-affinity NIC (GB/s)	Affinity NIC (GB/s)	Improvement
Chip A → B	5.51 ×8	9.56 ×8	73.5%
Chip B → D	5.23 ×8	9.91 ×8	89.5%

Table 4. Model configuration used in this work.

Parameter	Value	Parameter	Value
# Layers	96	Hidden Size	8192
# Attention Heads	64	Intermediate Size	36864
# Queries per Head	8	Vocabulary Size	92544
Max Sequence Length	4096		

Second, we employ a communication mode that combines send/recv with an all-gather operation. This strategy optimizes the system by minimizing cross-node data transfers. Additionally, since multiple chips share a single NIC during communication, the available bandwidth on each NIC is maximized. Given that our tensor parallel communications predominantly occur within the node, the communication overhead incurred by the all-gather operation within the node is negligible compared to inter-node communication delays.

To further reduce communication overhead in pipeline parallelism, we have implemented a fine-grained overlap between point-to-point (P2P) communication and computation at the framework level. Drawing inspiration from previous works [7, 28], we decompose the traditional forward-backward computation into four distinct phases: forward computation, backward recomputation, backward input gradient calculation, and backward weight gradient calculation. This decomposition enables a more precise interleaving of computation with P2P communication. Consequently, our approach achieves near-lossless P2P communication performance, substantially minimizing the communication overhead in pipeline-parallel training.

6 Main Evaluations

6.1 Experimental Setup

6.1.1 Model Configuration. The architectural configuration of the 100B-parameter model[5], as presented in Table 4, adheres to the structural design principles established by LLaMA [12]. To further enhance memory efficiency during inference, the model incorporates Group Query Attention, a technique that effectively reduces memory usage while maintaining performance.

6.1.2 Chip Specification. We tested our framework with four different types of AI chips. A detailed comparison of

Table 5. Performance comparison of AI chips relative to the A100 used in this study.

Chip	FP16 (TFLOPS)	Memory (GB)	#Chips per Node
A	>0.5, <1.0 × A100	96	16
B	>0.5, <1.0 × A100	64	8
C	>0.0, <0.5 × A100	32	16
D	>1.5, <2.0 × A100	32	8

Table 6. Training throughput (tokens per chip per second, TGS) comparison and hybrid parallelism configurations for homogeneous training among Chips A, B, C and D on 256 chips.

Chip	PP	DP	TP	Extra	TGS
Chip-A	16	4	4	N/A	136.9
Chip-B	16	4	4	Activation Recompute	143.7
Chip-C	32	2	4	Activation Recompute	46.2
Chip-D	8	4	8	CPU Offload	99.5

the AI accelerators used in this study is presented in Table 5. Notably, Chip-D outperform the NVIDIA A100 GPU in computational power, although they are constrained by limited memory capacity. Among the accelerators deployed, only Chip-A offers a larger memory capacity than the NVIDIA A100 GPU.

Figure 6 presents the training throughput of the 100B-parameter model, measured in tokens per chip per second (TGS[3]), on homogeneous clusters using Chips A, B, C and D. Tables 6 shows the used hybrid parallelism configurations. Each chip's training efficiency is evaluated on a 256-chip setup with a global batch size of 2M tokens. Chip-B achieves the highest throughput at 143.7 TGS. In contrast, Chip-C demonstrates the lowest performance, reaching only 46.2 TGS, primarily due to its lower FP16 TFLOPS, as detailed in Table 5. Interestingly, despite Chip-D having the highest theoretical peak performance, its actual throughput is limited to 99.5 TGS due to memory constraints and communication bottlenecks. To mitigate these limitations, we employ CPU offloading and tensor parallelism for Chip-D, though this significantly impacts its overall training efficiency.

6.1.3 Cluster Information. In this study, all AI chips within each of the nine clusters are interconnected via a multi-rail RDMA network utilizing RoCE-v2. The network is configured with a 1:1 oversubscription ratio to ensure optimal data transmission efficiency.

[3]TGS is typically used to denote tokens per GPU per second; however, we use it here for consistency with previous studies.

Table 7. Chip-Configurations and corresponding global batch sizes for HeteroPP and HeteroAuto.

Index	Chip-Configuration	GBS (Tokens)
Exp-A-1	Chip-A (256) + B (256) + C (256)	2M
Exp-A-2	Chip-A (256) + B (256) + C (256)	6M
Exp-B-1	Chip-A (256) + B (256) + C (256) + D (256)	2M
Exp-B-2	Chip-A (256) + B (256) + C (256) + D (256)	8M
Exp-C-1	Chip-A (384) + B (1024)	4M
Exp-C-2	Chip-A (384) + B (1024)	8M
Exp-D	Chip-A (384) + B (2048)	8M

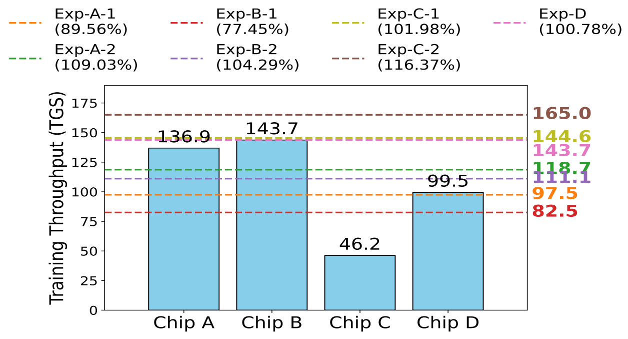

Figure 11. Training throughput for individual homogeneous and heterogeneous training setups, and HeteroSpeedupRatio for heterogeneous training setups.

6.2 Evaluations on HeteroPP and HeteroAuto

To validate the performance and efficiency of the proposed heterogeneous parallelism strategy, we conducted a series of extensive experiments focusing on HeteroPP and HeteroAuto. Various chip configurations, along with their corresponding global batch sizes (GBS), were evaluated. These configurations are summarized in Table 7.

We use the *HeteroSpeedupRatio* to evaluate heterogeneous training performance, defined as the ratio of training throughput on the heterogeneous cluster to the sum of the baseline training throughput of each AI chip type in Table 6:

$$HeteroSpeedupRatio = \frac{N \cdot TGS}{\sum_{i=1}^{C} N_i \cdot TGS_i}.$$

6.2.1 Effectiveness. In the cases of Exp-A to Exp-D, which utilized Chips A, B, C and D, we compared the throughput of each chip type with that achieved through their combined heterogeneous training configurations. In Exp-A and Exp-B, Two heterogeneous training configurations were evaluated. In the first configuration, the global batch size was the same as that used for individual chip-group training. In the second configuration, the global batch size equaled the sum of the batch sizes employed in individual chip-group training. The results, as shown in Figure 11, demonstrate that the heterogeneous training approach delivers competitive, and in some instances, superior efficiency compared to the baseline throughput of homogeneous training.

For the configuration where GBS is set to the sum, DiTrain achieves, a 109.03% *HeteroSpeedupRatio* with 768 chips in Experiment A, involving 3 chip types, and 104.29% with 1,024 chips in Experiment B, incorporating 4 chip types. For constant GBS, *HeteroSpeedupRatio* can still achieve 89.56% and 77.45% respectively.

Although the observed superlinear performance improvement may appear counterintuitive, it is explainable. The conventional 3D parallel training tends to overlook the imbalanced resource requirements among various computational tasks, while the HeteroPP framework with HeteroAuto capitalizes on these imbalances by intelligently allocating chip tasks and fine-tuning training hyperparameters based on the specific resource demands. Taking Exp-C as example, Chip B has better computational power but is limited by memory, requiring a TP size of at least 8 or activation recomputation enabled, which reduces efficiency due to communication overhead or extra computation tasks. Chip A, while less powerful, has more global memory. In the heterogeneous setup, Chip A handles the earlier pipeline stages, which requires more memory, while Chip B is used for later stages where a smaller TP size is sufficient and activation recomputation can be turned off. This configuration allows Chip B to fully utilize its computational power and chip A to fully utilize its memory capacity, improving overall efficiency.

In practical production environments, lower-spec chips usually feature significantly lower pricing and reduced power consumption compared to their high-spec chips. By leveraging the HeteroPP framework for heterogeneous training, which integrates both lower-spec and high-spec chips, we can achieve training performance comparable to or even exceeding that of homogeneous high-spec setups, which preserves high performance and reduces overall training costs.

6.2.2 Searching Overheads. Beyond performance and scalability, we also evaluated the computational overhead of the strategy search process. Due to the diverse set of heterogeneous chips used in our training, directly comparing with other heterogeneous strategy search algorithms (e.g., Metis[37] and Alpa[41]) requires significant adaptation efforts. However, for reference, Metis takes 600 seconds to complete a search for only 64 chips and two chip types, while Alpa takes 240 minutes for the same task. Table 8 summarizes the overheads for our search strategy in three configurations. The search is implemented as a single-threaded Python script running on an Intel® Xeon® Platinum 8460Y+ CPU model, and the two-stage search algorithm described in Section 4 is applied. In the second stage, every set of 128 homogeneous chips is treated as a single group of heterogeneous chips.

Table 8. Overhead of the strategy search process.

	Exp-A	Exp-B	Exp-C
Time (s)	0.62	5.48	12.29

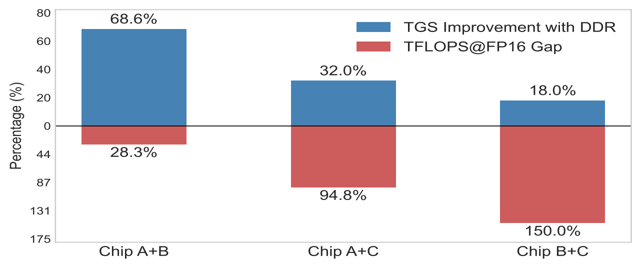

Figure 12. For end-to-end training of a small-scale 8-decoder-layer model, experiments were conducted with and without DDR. Uniform 1F1B schedule was employed, and the parallelism configuration was set to TP=4, PP=2, and DP=2. Two heterogeneous servers, with eight chips on each server, were utilized.

6.3 Ablation Study

Figure 7 provides a detailed comparison of the latency across various message sizes for different communication methods. To more intuitively demonstrate the impact of DiComm, we conducted an end-to-end performance comparison between CPU-mediated TCP and device-direct RDMA communications. This evaluation encompasses both large-scale and small-scale models as well as various chip configurations. The experimental results for small-scale end-to-end training are presented in Figure 12.

The results clearly show that device-direct RDMA mode consistently yields substantial performance gains over CPU-mediated TCP across all chip combinations. However, the degree of improvement varies: the performance gap is relatively small between Chip A and B, whereas Chip C exhibits a markedly larger discrepancy compared to the others. Consequently, under the current parallel strategy, Chip C becomes the computational bottleneck, thereby limiting the overall benefits derived from optimizing pipeline parallel P2P communication. These observations suggest that the refinement of heterogeneous parallel strategy is essential to fully take advantages of DiComm and achieve enhanced training efficiency.

In large-scale training scenarios, the increased computational load occupies a larger portion of the overall workload,

Table 9. Ablation study variants for large-scale heterogeneous training using Exp-C-1 configuration. SR&AG resharding refers to the send/recv and all-gather activation resharding mentioned in section 5.

DiComm Mode	Strategy Search	PP Method	Optimization	Relative Iteration Time
DDR	HeteroAuto	HeteroPP 1F1B	Full	100%
TCP	HeteroAuto	HeteroPP 1F1B	Full	110.1%
DDR	HeteroAuto	Uniform 1F1B	Full	126.4%
DDR	HeteroAuto	HeteroPP 1F1B	w/o SR&AG resharding	104.8%
DDR	HeteroAuto	HeteroPP 1F1B	w/o fine-grained overlap	101.8%

effectively masking more substantial communication latencies. Consequently, the performance gap between device-direct RDMA and CPU-mediated TCP communication methods narrows, and HeteroPP brings obvious throughput improvement as demonstrated in Table 9.

7 Conclusion

In this paper, we propose a unified framework H2 for efficient large language model training on hyper-heter-ogeneous clusters comprising over 1,000 chips with diverse architectures and software stacks. By integrating DiTorch and DiComm to unify the programming interface and communication library, our approach overcomes the challenges related to operator and communication isolation. In addition, the HeteroPP framework, together with the HeteroAuto automatic strategy search, dynamically optimizes parallelism configurations based on rigorous cost modeling and hardware profiling. Moreover, our topology-aware activation resharding strategy and fine-grained computation-communication overlapping minimizes cross-node data transfer delays, further improving overall performance. Experimental results on a 100B-parameter model demonstrate that our solution achieves a superlinear speedup compared to traditional homogeneous methods by effectively balancing computational loads, memory disparities, and communication overhead.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 265–283.

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

[3] Wei An, Xiao Bi, Guanting Chen, Shanhuang Chen, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Wenjun Gao, Kang Guan, Jianzhong Guo, Yongqiang Guo, Zhe Fu, Ying He, Panpan Huang, Jiashi Li, Wenfeng Liang, Xiaodong Liu, Xin Liu, Yiyuan Liu, Yuxuan Liu, Shanghao Lu, Xuan Lu, Xiaotao Nie, Tian Pei, Junjie Qiu, Hui Qu, Zehui Ren, Zhangli Sha, Xuecheng Su, Xiaowen Sun, Yixuan Tan, Minghui Tang, Shiyu Wang, Yaohui Wang, Yongji Wang, Ziwei Xie, Yiliang Xiong, Yanhong Xu, Shengfeng Ye, Shuiping Yu, Yukun Zha, Liyue Zhang, Haowei Zhang, Mingchuan Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, and Yuheng Zou. 2024. Fire-Flyer AI-HPC: A Cost-Effective Software-Hardware Co-Design for Deep Learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis* (Atlanta, GA, USA) *(SC '24)*. IEEE Press, Article 83, 23 pages. doi:10.1109/SC41406.2024.00089

[4] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM. doi:10.1145/3620665.3640366

[5] Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Chao Xu, Ruiliang Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fengzhe Zhou, Zaida Zhou, Jingming Zhuo, Yicheng Zou, Xipeng Qiu, Yu Qiao, and Dahua Lin. 2024. InternLM2 Technical Report. arXiv:2403.17297 [cs.CL]

[6] Li-Wen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Ziheng Jiang, Haibin Lin, Xin Jin, and Xin Liu. 2024. FLUX: Fast Software-based Communication Overlap On GPUs Through Kernel Fusion. arXiv:2406.06858 [cs.LG]

[7] Ping Chen, Wenjie Zhang, Shuibing He, Weijian Chen, Siling Yang, Kexin Huang, Yanlong Yin, Xuan Zhan, Yingjie Gu, Zhuwei Peng, et al. 2024. Optimizing large model training through overlapped activation recomputation. *arXiv preprint arXiv:2406.08756* (2024).

[8] Jack Choquette, Edward Lee, Ronny Krashinsky, Vishnu Balan, and Brucek Khailany. 2021. 3.2 the a100 datacenter gpu and ampere architecture. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 48–50.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[10] Anne C Elster and Tor A Haugdahl. 2022. Nvidia hopper gpu and grace cpu highlights. *Computing in Science & Engineering* 24, 2 (2022), 95–100.

[11] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. 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*. 431–445.

[12] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[13] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

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

[15] Xianyan Jia, Le Jiang, Ang Wang, Wencong Xiao, Ziji Shi, Jie Zhang, Xinyuan Li, Langshi Chen, Yong Li, Zhen Zheng, et al. 2022. Whale: Efficient giant model training over heterogeneous {GPUs}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 673–688.

[16] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R Tallent, and Kevin J Barker. 2019. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems* 31, 1 (2019), 94–110.

[17] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R Tallent, and Kevin J Barker. 2019. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems* 31, 1 (2019), 94–110.

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

[19] 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. doi:10.1145/3458817.3476145

[20] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[21] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[22] 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.

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

[24] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 1–15.

[25] NVIDIA. 2025. *CUTLASS: CUDA Templates for Linear Algebra Subroutines*. https://github.com/NVIDIA/cutlass Accessed: 2025-05-11.

[26] NVIDIA. 2025. *NCCL: Optimized primitives for collective multi-GPU communication*. https://github.com/NVIDIA/nccl Accessed: 2025-05-11.

[27] Nathan Otterness and J Anderson. 2020. AMD GPUs as an alternative to NVIDIA for supporting real-time workloads. In *Proceedings of the 32nd Euromicro Conference on Real-Time Systems*.

[28] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. 2024. Zero Bubble (Almost) Pipeline Parallelism. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=tuzTN0eIO5

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

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

[31] 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*. 3505–3506.

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

[33] Ding Tang, Lijuan Jiang, Jiecheng Zhou, Minxi Jin, Hengjie Li, Xingcheng Zhang, Zhilin Pei, and Jidong Zhai. 2024. ZeroPP: Unleashing Exceptional Parallelism Efficiency through Tensor-Parallelism-Free Methodology. arXiv:2402.03791 [cs.DC] https://arxiv.org/abs/2402.03791

[34] InternLM Team. 2023. InternLM: A Multilingual Language Model with Progressively Enhanced Capabilities. https://github.com/InternLM/InternLM.

[35] Ajay Tirumala and Raymond Wong. 2024. Nvidia blackwell platform: Advancing generative ai and accelerated computing. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 1–33.

[36] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

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

[38] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 945–960. https://www.usenix.org/conference/nsdi22/presentation/weng

[39] Peng Wu. 2023. Pytorch 2.0: The journey to bringing compiler technologies to the core of pytorch (keynote). In *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*. 1–1.

[40] Si Xu, Zixiao Huang, Yan Zeng, Shengen Yan, Xuefei Ning, Quanlu Zhang, Haolin Ye, Sipei Gu, Chunsheng Shui, Zhezheng Lin, et al. 2024. HETHUB: A Distributed Training System with Heterogeneous Cluster for Large-Scale Models. *arXiv preprint arXiv:2405.16256* (2024).

[41] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. 2022. Alpa: Automating inter-and {Intra-Operator}

parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 559–578.

[42] Yonghao Zhuang, Lianmin Zheng, Zhuohan Li, Eric Xing, Qirong Ho, Joseph Gonzalez, Ion Stoica, Hao Zhang, and Hexu Zhao. 2023. On optimizing the communication of model parallelism. *Proceedings of Machine Learning and Systems* 5 (2023), 526–540.