HeterMoE: Efficient Training of Mixture-of-Experts Models on Heterogeneous GPUs

Yongji Wu[1,*] Xueshen Liu[2,*] Shuowei Jin[2] Ceyu Xu[4]

Feng Qian[3] Z. Morley Mao[2] Matthew Lentz[4] Danyang Zhuo[4] Ion Stoica[1]

[1]UC Berkeley [2]Univerisity of Michigan [3]University of Southern California [4]Duke Univerisity

Abstract

The Mixture-of-Experts (MoE) architecture has become increasingly popular as a method to scale up large language models (LLMs). To save costs, heterogeneity-aware training solutions have been proposed to utilize GPU clusters made up of both newer and older-generation GPUs. However, existing solutions are agnostic to the performance characteristics of different MoE model components (i.e., attention and expert) and do not fully utilize each GPU's compute capability.

In this paper, we introduce HeterMoE, a system to efficiently train MoE models on heterogeneous GPUs. Our key insight is that newer GPUs significantly outperform older generations on attention due to architectural advancements, while older GPUs are still relatively efficient for experts. HeterMoE disaggregates attention and expert computation, where older GPUs are only assigned with expert modules. Through the proposed zebra parallelism, HeterMoE overlaps the computation on different GPUs, in addition to employing an asymmetric expert assignment strategy for fine-grained load balancing to minimize GPU idle time. Our evaluation shows that HeterMoE achieves up to 2.3x speed-up compared to existing MoE training systems, and 1.4x compared to an optimally balanced heterogeneity-aware solution. HeterMoE efficiently utilizes older GPUs by maintaining 95% training throughput on average, even with half of the GPUs in a homogeneous A40 cluster replaced with V100.

1 Introduction

Over the past few years, large language models (LLMs) have demonstrated impressive capabilities in domains like conversation agents [3] and coding assistants [34, 39]. Recently, the sparsely-activated Mixture-of-Experts (MoE) architecture has gained popularity as the preferred way of scaling models to hundreds of billions of parameters [21, 46]. MoE models are often trained with expert parallelism [19] where the experts are distributed across GPUs, while token activations are exchanged between GPUs using all-to-all communication.

Meanwhile, GPU hardware is continuously evolving, with newer GPUs increasing performance but simultaneously cost. The demand for newer GPUs is high, with the latest Grok 3 [29] and upcoming Llama 4 [43] requiring over 100K H100 GPUs (and costing over 4 billion) [35]. Moreover, the latest GPUs face significant supply constraints, often with a backlog of several months [40, 41].

Therefore, we need to answer the following question: How can we effectively train MoE-based LLMs on clusters with multiple generations of GPUs? Due to the high cost and limited supplies of latest GPUs, many organizations often retain nodes with older GPUs (e.g., V100) while adding new nodes with the latest GPUs (e.g., H100) [1, 14] when upgrading infrastructure. Given the extremely high resource requirement for LLM training, we need to utilize all available GPUs.

Leveraging heterogeneous GPUs is challenging due to their varying hardware properties (e.g., memory size, computation capability). To split data, prior work has looked at scaling the batch size in data parallelism for each GPU [17, 24]. To split the model, prior work has explored unevenly distributing model layers in pipeline parallelism across GPUs [15, 42, 45]. However, the fundamental limitation of these solutions is that they are agnostic to the existence of heterogeneity *within* the model architecture itself.

Our insight is that different components of MoE models (i.e., attention and expert) exhibit distinct performance characteristics across GPU generations. Older GPUs remain highly efficient for expert computation. In contrast, attention performs significantly better on newer GPUs due to architecture-specific optimizations. For instance, FlashAttention v2 [4] exclusively supports Ampere and newer GPUs [9], while FlashAttention v3 [36] leverages Hopper-specific features like `wgmma` instructions and TMA [28]. As experts are already placed across GPUs in expert parallelism, we are presented with an opportunity to assign each GPU only components it can efficiently compute, without introducing additional com-

*Yongji Wu and Xueshen Liu contributed equally.

munication.

In this paper, we present HeterMoE to efficiently train MoE models with heterogeneous GPUs. HeterMoE disaggregates the attention and expert blocks of a transformer layer, assigning them to two different generations of GPUs (newer and older, respectively). The attention-expert disaggregation not only better harvests the compute power of older GPUs, but also alleviates the memory pressure on newer GPUs due to the dominant expert weights and their limited availability.

Still, there are two key challenges HeterMoE must address. First, how can we overlap the computation of different GPUs to reduce the idle wait time of GPUs? Naïve attention-expert disaggregation leaves GPUs spending most of their time waiting for each other due to data dependency. Second, to maximize the extent of overlapping, how can we balance the computation on each GPU at a fine granularity? Simply tuning the degree of parallelism leads to a narrow optimization space, limited by the number of valid configurations. For instance, in expert parallelism, the number of experts must be divisible by the number of GPUs to distribute them.

To address these challenges, we propose zebra parallelism, which divides an input batch to multiple micro-batches and overlaps the attention computation on the newer GPUs and expert computation on the older GPUs of different micro-batches. Zebra parallelism differs from pipeline parallelism, where the model is partitioned into multiple GPUs at the granularity of one or more layers. In pipeline parallelism, each sample is sequentially computed on each GPU with the corresponding layers, from the first GPU to the last one. In contrast, in zebra parallelism, the model is partitioned within a single layer, and each sample is computed in a zigzag pattern, passing back and forth between attention (newer) and expert (older) GPUs. To enable fine-grained load balancing of attention and expert GPUs, we propose an asymmetric expert assignment (Asym-EA) mechanism, where we place a part of the experts back to attention GPUs when expert computation is slow. Asym-EA can be selectively activated for a subset of layers and moves back a different number of experts for different layers. We further develop a "gather and squeeze" strategy for Asym-EA to optimize each layer's assignment and minimize the GPU idle time, i.e., bubbles.

We implemented HeterMoE in PyTorch, evaluated across MoE models of different scales, using both an on-premise testbed and EC2 instances under different heterogeneity settings. Our results show that HeterMoE outperforms existing MoE training systems by up to 2.3x and an optimally balanced heterogeneity-aware solution by 1.4x. In addition, HeterMoE achieves 95% training throughput on average compared to a homogeneous setting where all older GPUs are replaced by the newer ones.

We summarize our contributions as follows:

- We observe that the performance disparity between newer and older generation GPUs differs significantly for attention and expert blocks, motivating our solution to disaggregate the two blocks for training MoE models on heterogeneous clusters, with no extra communication.
- We propose zebra parallelism to overlap not only the computation of attention GPUs with the computation of expert GPUs, but also the computation and all-to-all communication on each GPU.
- We design an asymmetric expert assignment mechanism to selectively move different numbers of experts back to attention GPUs for different layers, enabling fine-grained load balancing for zebra parallelism to minimize bubbles.

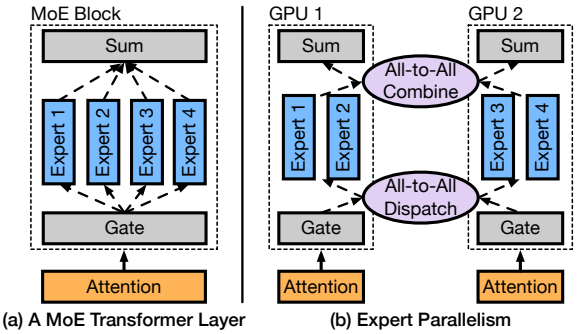


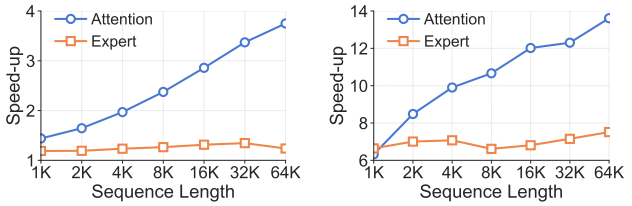
Figure 1: MoE architecture and expert parallelism.

2 Background and Motivation

2.1 Mixture-of-Experts Models

The mixture-of-expert architecture, shown in Figure 1, has been rapidly adopted in recent LLMs and has demonstrated as a cost-efficient solution for parameter scaling. In MoE models, the dense feed-forward network (FFN) in a transformer layer is replaced by multiple parallel FFNs, which are called experts. For each token, a trainable gate network takes the output embeddings of the attention block and computes a confidence score for each expert. Based on the scores, the token is only routed to the top-k experts, whose outputs are weighted by their scores and summed up. The selective activation of experts enables MoE to scale model parameters with a sub-linear increase in computation costs. The attention blocks remain the same as dense LLMs. Unlike experts that mainly consist of GEMM operations, which are highly optimized on most GPUs, attention is less optimized. Only recently have implementations like FlashAttention [4, 5, 36] been proposed, which are specifically tuned to recent GPU architectures and can significantly accelerate attention.

Expert weights have become increasingly dominant in the overall model size as sparsity increases. In Mixtral 8x7B [16], 27.6% of the parameters are active, while the number has decreased to 5.5% in DeepSeek v3 [21]. Expert parallelism is hence proposed for training MoE models, which distributes the experts of a MoE block to multiple GPUs. For each MoE



(a) A40 over V100 (b) L40S over T4

Figure 2: Speed-up of newer generation GPUs over older ones on attention and expert modules from Mixtral 8x7B [16].

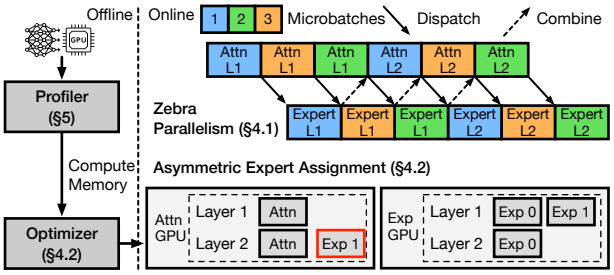


Figure 3: Major components of HeterMoE. Zebra parallelism overlaps the execution of attention and expert GPUs, while we introduce Asym-EA to minimize bubbles.

block, a dispatch all-to-all collective is performed to send the embeddings of input tokens to GPUs with target experts, followed by a combine all-to-all that retrieves the outputs.

2.2 Heterogeneous Training

There is an increasing demand for training LLMs on heterogeneous GPU clusters due to frequent GPU release cycles, high upgrade costs and a lack of supply. Traditional training systems typically assume a homogeneous cluster and split workloads evenly across all GPUs. Under heterogeneous settings, such systems will let faster GPUs idle and wait for slower ones, significantly under-utilizing faster GPUs.

To mitigate the bottlenecks caused by slower GPUs, existing heterogeneity-aware training systems like [15, 42, 45] unevenly split both data and models across different GPUs. They assign different batch sizes for different GPUs accordingly to balance the data load on each GPU, while to split the model, layers are partitioned according to each GPU's compute and memory capabilities.

However, these systems have several limitations for training MoE models. First, they do not differentiate different model components, failing to assign each component only to the GPUs that can efficiently execute it. Second, it is still infeasible to train large MoE models using only layer-based partitioning, as the weights of a single MoE block may exceed the GPU memory. Finally, since the model is partitioned at layer granularity, it can be challenging to find a partition that effectively balance the compute of different GPUs. Furthermore, balancing the compute often conflicts with memory capacities, hence limited by memory, faster GPUs may be assigned fewer layers required to fully utilize their compute capabilities, while slower GPUs may leave their memory underutilized due to their limited compute capabilities.

2.3 Opportunities

We find that the relative efficiency of different GPUs differs significantly for different model components. In Figure 2, we show the speed-up of two newer generation GPUs over two older ones. We measure each GPU's total forward and backward time for a single attention and expert module,using the model settings from Mixtral 8x7B [16]. We find that V100 GPUs are still quite capable for computing experts, achieving on average 80% of A40's performance, which is equipped with the fully enabled flagship GA102 chip of Ampere generation. The relative performance of V100 compared to A40 on experts also remains stable across different sequence lengths, as each token is independently computed.

V100 suffers from much worse performance on attention modules as it does not support FlashAttention [9] and hence greatly bound by memory bandwidth, although we still use an optimized attention implementation [18]. The performance gap between A40 and V100 also rapidly widens as sequence length increases, contributed by the quadratic complexity of self-attention with respect to the sequence length. For 64K sequences, A40 outperforms V100 by 3.7x. From Figure 2b, since T4 is two generations older with half the SMs of V100, L40S speeds-up MLP over T4 by 7.0x on average. Still, the speed-ups on attention is much more significant, with L40S outperforms T4 by 9.9x for 4K sequences and 13.6x for 64K sequences. We note that although T4 support FlashAttention v1 [5], due to limited SRAM sizes, it offers limited speeds-up and does not support larger attention head dimension [8] as used in [16, 21, 46].

Hence, to efficiently utilize older GPUs, we should avoid assigning them with attention operations. Existing heterogeneous training systems, however, are agnostic to such differences between attention and experts.

3 Overview

To efficiently train MoE models on heterogeneous GPU clusters, we propose HeterMoE, a training system that disaggregates attention and expert computation within a transformer layer. HeterMoE assigns only expert blocks to older generation GPUs. Since MoE models are already trained using expert parallelism with all-to-all on homogeneous clusters, our disaggregation introduces no extra communication, as the total amount of data exchanged between GPUs remains unchanged, given the same global batch size. Such disaggregation not only improves the utilization efficiency of older

GPUs, but also reduces memory pressure on newer GPUs by offloading expert weights to many older GPUs. It reduces the cost of MoE training by using fewer newer generation GPUs while having minimal performance degradation.

We present the major components of HeterMoE in Figure 3. We designate older generation GPUs that are only assigned with experts as expert GPUs, while newer GPUs are designated as attention GPUs. Naively computing the attention and expert blocks on separate GPUs results in attention GPUs being idle while waiting for expert GPUs during expert computation (and vice versa). To address this issue, we propose zebra parallelism (§4.1), which divides each transformer layer into two stages, allowing attention and expert GPUs to simultaneously work on different microbatches.

To minimize the idle time, we need to closely balance the execution time of attention and expert GPUs for each microbatch. We introduce asymmetric expert assignment (§4.2), or Asym-EA, to enable fine-grained control of attention and expert computation across GPUs. Asym-EA enables the migration of specific parts of the expert computation to (newer) attention GPUs; for instance, in Figure 3, Exp 1 for Layer 2 is moved from the Expert GPU to the Attention GPU to reduce bubbles in the compute stream. Given this underlying mechanism for migration, we develop a "gather and squeeze" algorithm, which is implemented by the Optimizer (§4.2), to determine the best expert assignment. The Optimizer depends on profiling information from the Profiler (§5), such as the computation time and memory consumption for attention and expert blocks on the different GPUs.

4 Design

In §4.1, we first discuss how zebra parallelism works, under the base case where attention GPUs are not assigned with any experts. Next, in §4.2, we discuss how Asym-EA can assign attention GPUs with some experts to reduce bubbles.

4.1 Zebra Parallelism

Zebra parallelism (ZP) takes the place of traditional expert parallelism (EP) under heterogeneous settings. In an EP group, the experts of each layer are evenly split across E GPUs, while attention blocks are replicated on each GPU. In contrast, in a ZP group, expert modules are distributed on N expert GPUs, while all other components, including attention blocks and input/output embedding layers are replicated on M attention GPUs. ZP differs from pipeline parallelism (PP) as PP assigns segments of consecutive layers to different GPUs, while ZP splits modules within each layer to different GPUs. ZP can be used in conjunction with PP, where we have ZP to split experts in each PP stage.

In ZP, computation on attention and expert GPUs are overlapped, as they process different microbatches at a time. Similar to EP. ZP still relies on all-to-all communication to exchange tokens. The communication in ZP is bipartite. During the forward phase, each of the M attention GPUs only sends tokens to the N expert GPUs during dispatch, and only receives from expert GPUs during combine. During the backward phase, the communication is reversed.

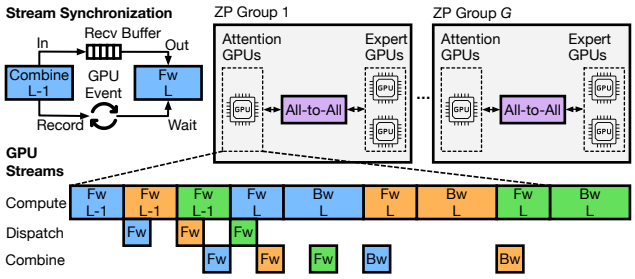


Figure 4: Zebra parallelism replaces expert parallelism in heterogeneous settings. It overlaps compute on attention and expert GPUs, as well as compute and communication within each GPU.

We consider network links between attention and expert GPUs have sufficient bandwidth so that dispatch and combine communication is faster than the attention and expert computation, similar to existing MoE training systems [12, 20]. We also assume a homogeneous network topology, otherwise, if the interconnect bandwidth within attention or expert GPUs is faster than the links between them, hierarchical all-to-all optimizations [12, 27] can be applied for HeterMoE to take advantage of. Still, all-to-all takes up to 30%-50% of the overall training time [12, 20, 49], hence HeterMoE also overlaps it with computation on each GPU.

To enable the overlapping, we use separate GPU streams for computation and communication. Since dispatch and combine all-to-all communicate in opposite directions and cause no bandwidth contention, they are scheduled on two independent communication streams and are overlapped. Hence, on each GPU, HeterMoE maintains two streams for communication and one stream for computation. To maintain the data dependency between computation and communication tasks, HeterMoE uses GPU events to synchronize different streams. For instance, to receive the input data, an all-to-all kernel is first launched (enqueued) on one of the communication stream, HeterMoE then records the corresponding event. The computation stream would wait for that event and block the execution of attention or expert computation until the all-to-all is finished and the data in the receive buffer is ready.

With zebra parallelism to overlap computation on different GPUs, as well as computation and communication on the same GPU, we still need to feed it with an execution schedule to minimize the per-iteration training time. We use $A^F_{i,j}, E^F_{i,j}, D^F_{i,j}, C^F_{i,j}$ and $A^B_{i,j}, E^B_{i,j}, D^B_{i,j}, C^B_{i,j}$ to denote the forward/backward attention computation (A), expert computation (E), dispatch (D) and combine (C) all-to-all tasks for the j-th

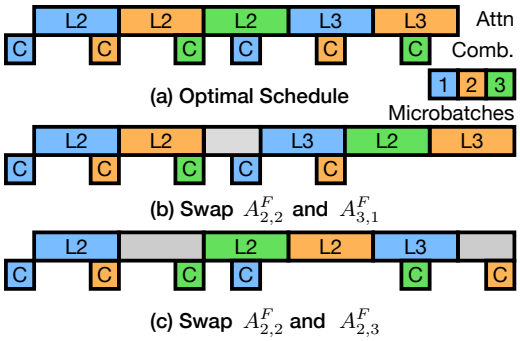


Figure 5: Demonstration of the optimal execution schedule of zebra parallelism, compared to swapping any of the two tasks. We show the forward attention computation and combine all-to-all of the second and third layers.

microbatch on the i-th layer. A task can start its execution as long as its dependent task is finished, and all previous tasks on the same stream finishes. For instance, $A^F_{i,j}$ can start as long as:

$$\begin{cases} t(A^F_{i,j}) \geq t(C^F_{i-1,j}) + T_C, & i \in (1,L], j \in [1,R] \\ \left| t(A^F_{i,j}) - t(A^F_{i',j'}) \right| \geq T_A, & \forall (i',j') \neq (i,j) \end{cases},$$

where $t(\cdot)$ is the start time of a task, L is the number of layers, R is the number of microbatches. T_A and T_C are the duration of attention computation and combine all-to-all communication for a microbatch. The first constraint is for data dependency, the attention computation of a microbatch cannot start before the previous layer's expert outputs are received. The second constraint enforces sequential execution of a stream, i.e., the attention GPU can only compute a single microbatch at a time. Intuitively, an optimal schedule should enable each task to start as soon as possible.

Theorem 1. *For a MoE model trained using HeterMoE's zebra parallelism, where all experts are placed on expert GPUs while the execution of tasks follow data dependency and stream sequential execution constraints, the following execution schedule on each GPU minimizes the total time of a training iteration:*
for computation on attention GPUs:

$$(A^F_{1,1} \cdots A^F_{1,R}) \cdots (A^F_{L,1} A^B_{L,1} \ldots A^F_{L,R} A^B_{L,R}) \cdots (A^B_{1,1} \cdots A^B_{1,R}),$$

for computation on expert GPUs:

$$(E^F_{1,1} \cdots E^F_{1,R}) \cdots (E^F_{L-1,1} \cdots E^F_{L-1,R})$$
$$(E^B_{L-1,1} \cdots E^B_{L-1,R}) \cdots (E^B_{1,1} E^B_{1,R}).$$

Following the above ordering of compute tasks, the dispatch and combine all-to-all tasks are scheduled to corresponding streams as soon as their dependent tasks are scheduled.

Proof. We prove that swapping any two tasks on the same stream will not reduce the total iteration time, regardless of how tasks on other streams are scheduled, as long as data dependencies are maintained. We show the proof for the computation stream on attention GPUs. The schedules for other streams can be similarly proved.

First, if we switch the order of $A^F_{i,j}$ and $A^F_{i+1,j'}$, where $1 \leq i < L$ and $1 \leq j' < j \leq R$, the earliest possible start time $t'(A^F_{i+1,j'})$ after swapping will not be earlier than $t(A^F_{i,j})$ when the two tasks are not swapped, as $A^F_{i+1,j'}$ needs to wait for the completion of $C^F_{i,j'}$ and $A^F_{i,j-1}$. If the compute tasks on expert GPUs are not swapped, we have $t'(C^F_{i,j'}) > t(C^F_{i-1,j})$, otherwise we have either $t'(C^F_{i,j'}) \geq t(C^F_{i-1,j})$ or $t'(A^F_{i,j-1}) \geq t(A^F_{i,j-1})$, depending on how the tasks are shuffled. Hence, the earliest start time of subsequent compute tasks will also be equal to or larger than their start time when $A^F_{i,j}$ is scheduled before $A^F_{i+1,j'}$. Figure 5(b) shows this case. Similarly, swapping $A^B_{i+1,j}$ with $A^B_{i,j'}$ leads to sub-optimal schedules.

Second, if we swap $A^F_{i,j}$ and $A^F_{i,j'}$, where $1 \leq i \leq L$ and $1 \leq j < j' \leq R$, the earliest possible start time $t'(A^F_{i,j'})$ will be no earlier than the earliest start time $t(A^F_{i,j})$ when the two tasks are not swapped, since either $t'(C^F_{i-1,j'}) \geq t(C^F_{i-1,j})$ or $t'(A^F_{i,j-1}) \geq t(A^F_{i,j-1})$ (for $j = 1$, it is $A^F_{i-1,R}$). Figure 5(c) shows this case. Similarly, we can prove the total execution time will not decrease if we swap $A^B_{i,j}$ and $A^B_{i,j'}$.

Finally, if we swap $A^B_{L,j}$ with $A^F_{L,j'}$ where $1 \leq j < j' \leq R$, the start time $t'(A^F_{L,j'})$ after swapping will not be earlier than $t(A^B_{L,j})$ when not swapped. All subsequent compute tasks will not start earlier. In particular, the execution of $A^B_{L,j}$ and $C^B_{L-1,j}$ will be deferred. □

With ZP to overlap the computation, we next explore how HeterMoE further balances the computation of each microbatch between the attention and expert GPUs in a fine-grained manner to minimize bubbles in a training iteration.

4.2 Asymmetric Expert Assignment

In HeterMoE's zebra parallelism, the computation time on attention and expert GPUs for a single microbatch and a single layer is determined given the number of attention and expert GPUs in each ZP group, i.e., M and N respectively. Expert computation takes place on lower-end GPUs and typically dominates the overall computation, especially for shorter sequences. Therefore, attention GPUs may be left with a series of bubbles as a result of waiting for the next microbatch's expert computation (and combine all-to-all) to finish after completing their attention computation. In Figure 6, we analyze a concrete scenario in which computation on expert GPUs is 33% slower than on attention GPUs. Given three microbatches, we show the execution schedule for the forward

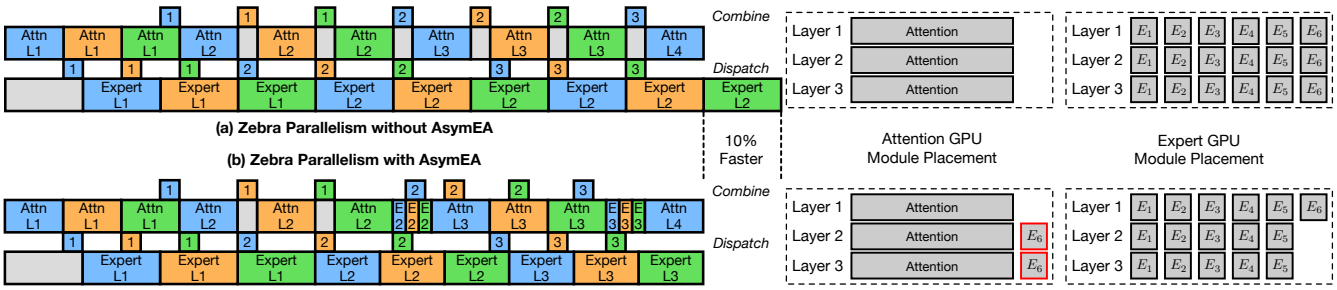


Figure 6: Asym-EA enables fine-grained balance of computation load on attention and expert GPUs to minimize bubbles.

pass of the first three layers. For default zebra parallelism (shown in Figure 6a), despite the fact that expert GPUs are fully occupied by computation tasks, attention GPUs only spend 75% time on effective computation.

To increase the utilization of attention GPUs, we propose an asymmetric expert assignment (Asym-EA) strategy where we offload some of the expert computation onto the (more powerful) attention GPUs. Given the ratio of experts to offload, HeterMoE splits the experts across attention and expert GPUs in a ZP group, During dispatch and combine, tokens are sent to and received from both attention and expert GPUs. On each attention GPU, HeterMoE separates attention and expert computation of each microbatch, as it must receives tokens from other attention GPUs before computing experts.

In Figure 6(b), we put one of the experts of the 2nd and 3rd layer to attention GPUs, while expert GPUs only handle five. For each layer, we schedule the expert computation on attention GPUs after attention computation of all microbatches. In this case, although the attention GPUs still have bubbles before the 2nd layer's attention computation, all subsequent computation becomes bubble-free. The 60% reduction in bubbles directly translates to 10% speed-up of the total forward time of the first 3 layers.

However, for a single layer, even if we only offload a single expert in this example, the computation on attention GPUs for each microbatch becomes longer than that on expert GPUs. If we simply offload the same number of experts for every layer, bubbles may shift to expert GPUs. For example, if we also offload an expert in the first layer in Figure 6(b), the total execution time does not decrease, as expert GPUs would then suffer from 6% bubbles. Therefore, we need to selectively offload specific layers.

To balance the total workloads on attention and expert GPUs, hence reducing the bubbles, we propose an algorithm based on a "gather and squeeze" strategy to determine the set of layers to offload experts, and the number of experts to offload. Our high-level insight is to "gather" enough bubbles on attention GPUs across several consecutive layers until we can offload at least one expert to "squeeze" the bubbles. We present this algorithm in Algorithm 1.

First, given the ZP group setup of M and N, we profile (§5) the forward compute time of a single layer's attention module for a microbatch on attention GPUs, T_A^{Attn}, and a single layer's expert module on expert GPUs, T_E^{Exp}. We note that T_E^{Exp} depends on the amount of tokens each expert GPU processes instead of the experts it is assigned. We also profile the compute time of a single expert (FFN) with the same batch size on attention GPUs as T_E^{Attn}. The profiled time is fed into Algorithm 1 as inputs.

Since each expert GPU and attention GPU must offload or acquire the same number of experts to ensure they have consistent workloads, and the total number of experts offloaded must equal that acquired by all attention GPUs, we define in line 1-2 the minimum number of experts n_1, that each atten-

Algorithm 1: Asym-EA offload optimization.

Input : n: Number of experts of each layer; L: Number of layers; M, N: Number of attention GPU and expert GPUs in a ZP group; $T_A^{\text{Attn}}, T_E^{\text{Attn}}, T_E^{\text{Exp}}$: Compute time of attention modules and expert modules on attention and expert GPUs.

Output : $O : \{o_1, o_2, \ldots, o_L\}$: Number of experts to offload from each expert GPU at each layer.

1 $n_1 \leftarrow \max(1, \frac{N}{M})$
// #experts each attention GPU acquires for a single chunk
2 $n_2 \leftarrow n_1 \cdot \frac{M}{N}$
// #experts each expert GPU offloads for a single chunk
3 $T_{\text{gather}} \leftarrow T_E^{\text{Exp}} - T_A^{\text{Attn}}$
// bubble at each layer without Asym-EA
4 $T_{\text{squeeze}} \leftarrow \frac{T_E^{\text{Exp}} N}{n} n_1 + \frac{T_E^{\text{Attn}} N}{n} n_2$
// bubble eliminated by offloading a chunk of n_2 experts
5 $t_{\text{bubble}} \leftarrow 0$ // total accumulated bubble
6 **for** $l \leftarrow 1$ *to* L **do**
7 $t_{\text{bubble}} \leftarrow t_{\text{bubble}} + T_{\text{gather}}$
8 **if** $t_{bubble} \geq T_{squeeze}$ **then**
 // gather enough bubbles until we can squeeze
9 $o_l \leftarrow \lfloor t_{\text{bubble}} / T_{\text{squeeze}} \rfloor$ // #chunks to offload
10 $t_{\text{bubble}} \leftarrow t_{\text{bubble}} - o_l \cdot T_{\text{squeeze}}$
11 $o_l \leftarrow o_l \cdot n_2$
12 **return** O

tion GPU must acquire, and the minimum number of experts n_2 each expert GPU must offload. n_1 and n_2 forms the minimal chunk (unit) for offloading. We can only offload one or multiple chunks at each layer. To make sure n_1 and n_2 are integers, we assume either M is a multiple of N, or N is a multiple of M. The limitation only applies when Asym-EA optimization is used, and is similar to the restriction of traditional EP that the number of GPUs in a EP group must divides the number of experts. We also assume we have enough microbatches to overlap the communication on expert GPUs and fully saturate their computation.

In line 3, we compute T_{gather}, the bubble formed at each layer for a microbatch when Asym-EA is not used. T_{squeeze} in line 4 is the bubble we can shrink by offloading a single chunk of n_2 experts from each expert GPU. $N \cdot T_E^{\text{Exp}}/n$ is the compute time reduced on each expert GPU by offloading a single expert, as it receives fewer tokens. $N \cdot T_E^{\text{Attn}}/n$ is the extra compute time added to each attention GPU, for each expert it acquires. We gather the bubbles in line 7 across multiple layers, until they are large enough to offload at least a single chunk. In line 9, we compute how many chunks we can offload, and multiply by n_2 in line 11, we get the number of experts to offload from each expert GPU.

We follow the forward pass to squeeze the bubbles and use profiled forward compute time as inputs. We note that backward time for each module scaled proportionally to the forward time, the assignment optimized from forward pass reduces both forward and backward time.

Addressing memory limitations. In practice, however, the number of experts to offload are limited by the memory capacities of both attention and expert GPUs. On the one hand, if the N expert GPUs cannot hold all experts across all layers, HeterMoE must offloads some experts, and the total of experts to offload, $\sum O$ will be lower bounded by $n_{\min}$. On the other hand, $\sum O$ will be upper bounded by $n_{\max}$, as attention GPUs cannot hold too many experts. To take account of such limitations, we modify line 7 as follows:

$$t_{\text{bubble}} \leftarrow t_{\text{bubble}} + \alpha \cdot \beta \cdot T_{\text{gather}}$$

where

$$\alpha = \min\left(\frac{\lfloor n_{\max}/n_2 \rfloor \cdot T_{\text{squeeze}}}{L \cdot T_{\text{gather}}}, 1\right)$$
$$\beta = \max\left(\frac{\lceil n_{\min}/n_2 \rceil \cdot T_{\text{squeeze}}}{L \cdot T_{\text{gather}}}, 1\right).$$

Without the coefficients α and β, the amount of bubble we can gather over all L layers is $L \cdot T_{\text{gather}}$, and we would offload $\lfloor \frac{L \cdot T_{\text{gather}}}{T_{\text{squeeze}}} \rfloor$ chunks. If the number of chunks exceeds $\lfloor n_{\max}/n_2 \rfloor$, i.e., the maximum allowed by the attention GPU's memory, we add a coefficient $\alpha < 1$ to T_{gather}. α enforces the upper bound, while line 6-11 spreads the offload chunks across layers. Similarly, β enforces the lower bound of $\lceil n_{\min}/n_2 \rceil$ chunks. Note that we have either $\alpha = 1$ or $\beta = 1$, since at most one of them is activated when $\lfloor \frac{L \cdot T_{\text{gather}}}{T_{\text{squeeze}}} \rfloor$ goes beyond the lower or upper bounds.

5 Implementation

We implement HeterMoE in 3K lines of Python based on PyTorch v2.2 [13], with components from DeepSpeed v0.14 [33].

Zebra parallelism engine. Given a user-defined MoE model, using the MoE block provided by HeterMoE and the ZP group setup, our zebra parallelism engine splits the model in each ZP group as in §4.1 and manages the training. During initialization, the ZP engine setups the three streams for compute and communication and allocates the receive buffers for each microbatch. It also establishes required collective communication groups. For all-to-all, HeterMoE creates separate groups for dispatch and combine to enable concurrent communication on separate streams, with each group containing all GPUs in the EP group. HeterMoE feeds unequal split sizes to PyTorch's NCCL all-to-all wrapper to distribute different number of tokens to different GPUs, based on the expert assignment. We implement ZP and data parallelism in our prototype, but HeterMoE can be extended to support ZP in conjunction with other parallelization strategies, e.g., heterogeneity-aware pipeline parallelism [42, 45].

We note that the gate network generates a confidence score for each of the top-k experts, which is used to compute weighted sum of the expert outputs. Such computation paradigm forms a "residual" connection. The backward pass from the MoE block's outputs is hence extended to two branches, one of them propagates to confidence scores and the gate network's weights. This branch would propagate all the way back to the first layer, as it does not depend on the gradients from expert GPUs. To prevent repetitive backward on the same weights, HeterMoE stops the backward propagation at attention outputs for each layer. The backward of previous layers would not start until the gradients of attention outputs (expert inputs) are received from expert GPUs and are accumulated with the gradients from the second branch.

Profiler. We implement a profiler for HeterMoE to obtain the compute time that is required by the Asym-EA optimizer as inputs. The profiler extracts a single expert FFN from a transformer layer. Given the global batch size, sequence length, the number of microbatchs and the ZP group setup, the profiler determines the number of tokens B each expert GPU computes for a single microbatch. It then generates random tensors with a batch size of B and feds it to the FFN to profile the expert compute time on both an attention (T_E^{Attn}) and an expert GPU (T_E^{Exp}). The remaining part of the transformer layer, including attention blocks and MoE gate, is extracted and profiled on an attention GPU to get T_A^{Attn}, where the input size is set to the size of a microbatch.

Our profiler also measures the memory usage of the GPU.

Table 1: Cluster settings used in the evaluation. O refers to our on-premise cluster and C refers to AWS.

Setup	O1	O2	O3	C1	C2
#GPUs	6xA	4xA	6xA	2xL	2xL
	6xV	8xV	3xV	6xT	8xT
GPU Model	A: A40 (48GB)			L: L40S (48GB)	
	V: V100 (16GB)			T: T4 (16GB)	
Network	100 Gbps			200 Gbps*	

Table 2: Configurations of models used in the evaluation.

Model	#Layers	Hidden Dim	#Experts	#Params
Mixtral-W1	4	2048	12	2.2B
Mixtral-W2	4	2048	24	4.3B
Mixtral-D1	8	1024	24	2.1B
Mixtral-D2	6	1024	18	1.2B
Mixtral-D3	8	1024	40	3.5B

On an expert GPU, it constructs a single expert FFN, and executes forward and backward passes with dummy inputs. It then measures the memory usage, which contains activations, weights, gradients, and optimizer states. Based on the remaining available memory, the profiler estimates how many experts in total can expert GPUs hold. Subtracting it from the total number of experts, we get n_{min}, the minimal number of experts HeterMoE needs to offload. Similarly, on an attention GPU, the profiler constructs the model by excluding all expert modules. The forward and backward pass is executed by replacing all-to-all with dummy operations. The profiler then estimates n_{max}, the maximum number of experts each attention GPU can hold. We only need to run the profiler once for each setup, and it only requires a single attention GPU and a single expert GPU.

6 Evaluation

6.1 Setups

Cluster setups. We evaluate HeterMoE on both an on-premise testbed and on AWS. On each testbed, we configure cluster setups with different numbers of GPUs of each type. Table 1 lists the setups we use. The GPUs in our on-premise testbed are connected with 100 Gbps Mellanox ConnectX-6 RoCE NICs. On AWS, we use g4dn.4xlarge and g6e.4xlarge instances, where they have 20 Gbps TCP networks. However, we find that under 20 Gbps links, communication takes up to 70% of the total training time. As a token's communication time is much longer than its compute time for both attention and experts, it is impossible to overlap compute and communication, resulting in all compared methods bottlenecked by communication. Therefore, to match L40S's compute speed, we simulate a network connection of 200 Gbps on AWS by reducing the amount of data transferred in all-to-all.

Models. We choose five MoE models of varying sizes based on the recent Mixtral architecture [16], listed in Table 2. We explore both deeper models with more layers and wider models with larger hidden sizes. We also vary the number of experts to fit clusters of different sizes, as the number of GPUs (to distribute experts) must divide it. [20], we use top-2 gating for all models as in [12, 20]. Following the standard practice [2, 25, 38], we apply activation checkpointing and FP16 mixed precision training. For each model and cluster setup, we set the global batch size to the maximum allowed by the GPU memory to run all baselines.

Baselines. As our zebra parallelism replaces traditional expert parallelism in heterogeneous environments, we mainly compared HeterMoE with EP-based training under a single ZP/EP group. We implement state-of-the-art system-side MoE training optimizations from Tutel [12] and Lina [20] on top of the widely adopted DeepSpeed MoE [32], which include optimized kernels and communication overlapping.

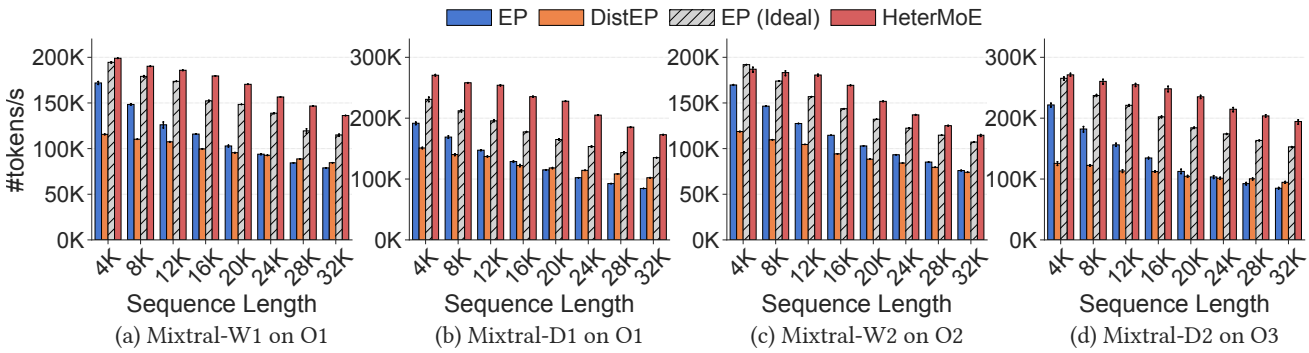
Based on the optimized DeepSpeed MoE, we compare two solutions: run it directly on the heterogeneous cluster using all GPUs, denoted as **EP**; naively disaggregate attention and expert modules without ZP to overlap the computation on different GPUs, denoted as **DistEP**. Since there is no heterogeneity-aware EP solution that achieves optimal load balancing, we also compare with an ideal case by running DeepSpeed MoE separately for each GPU model using all GPUs of that model, and then summing up their throughput. We denote it as **EP (Ideal)**. For instance, for O1 setup in Table 1, we independently run DeepSpeed MoE on 6xA100 and on 6xV100, then we simply add the throughput together. EP (Ideal) assumes prefect load balancing and does not take account of communication overheads across different GPU models. It represents the theoretical maximum throughput a MoE training system without attention-expert disaggregation can achieve. We tune both the number of microbatches R for HeterMoE and the all-to-all partitioning degree for all compared methods to maximize performance.

In addition, even though ZP/EP is orthogonal to data, tensor, pipeline parallelism and can be used in combination with them, we also compare HeterMoE with pure heterogeneity-aware pipeline parallelism in §6.3 to show PP's limitations.

6.2 Overall Performance

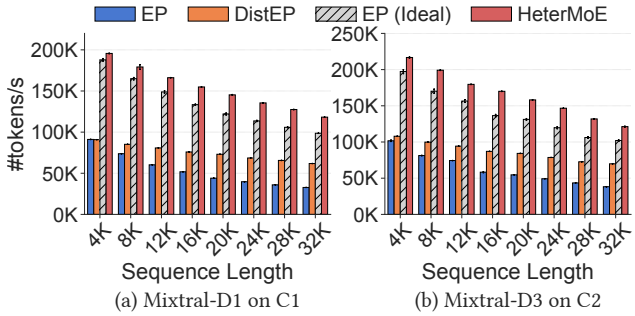
We report the overall training throughput in Figure 7 for our on-premise testbed, with sequence lengths from 4K to 32K. We note that long-context reasoning capabilities have been increasingly sought after, and recent LLMs are trained with context lengths of 32K or even up to 128K [6, 10, 21, 46].

Since attention has quadratic computation complexity to the sequence lengths, the training throughput of all compared



(a) Mixtral-W1 on O1 (b) Mixtral-D1 on O1 (c) Mixtral-W2 on O2 (d) Mixtral-D2 on O3

Figure 7: **[Overall Performance]:** Overall training throughput under different sequence lengths on the on-premise cluster of A40 and V100 GPUs, with different models and GPU setups. Error bars represent 95% confidence intervals.



(a) Mixtral-D1 on C1 (b) Mixtral-D3 on C2

Figure 8: **[Overall Performance]:** Overall training throughput on AWS with L40S and T4 GPUs.

methods drops as sequence length increases. We observe that at shorter sequence lengths, EP still achieves decent performance, since A40 offers limited speed-ups on both attention and experts as shown in Figure 2a. For instance, for 4K sequences, EP maintains 82% of HeterMoE's performance on average, and it even reaches 91% for Mixtral-W2 on `O2`. Since EP is not heterogeneity aware and does not balance the compute loads on A40 and V100, it suffers poor performance on longer sequences. For 16K sequences, HeterMoE outperforms EP by 1.67x on average, and it further increases to 1.89x at 32K. In particular, for deeper but narrower Mixtral-D1 and D2, where attention is more dominant and attention-expert disaggregation is more essential, HeterMoE's speed-ups over EP reach 2.05x and 2.29x for 32K sequences.

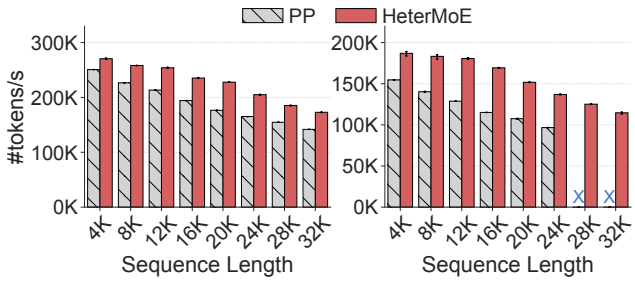
We also find that naively assigning experts to older GPUs without compute overlapping yields poor performance. DistEP achieves only 55% of HeterMoE' throughput on average across all settings. However, DistEP works relatively better for longer sequences when A40 has significant speed-ups on attention. On average, at 4K, DistEP's throughput is only 56% that of HeterMoE. It performs even worse than EP by 32%. At 32K, DistEP increases to 59% of HeterMoE's performance. It slightly outperforms EP by 8% on average, and up to 21%.

Comparing HeterMoE with the ideal case, EP (Ideal), HeterMoE still achieves an average speed-up of 1.18x. The speed-up is up to 1.38x for Mixtral-D1 on `O1` for 20K sequences, where HeterMoE can match A40's attention compute time with V100's expert compute time. For shorter sequences, attention is faster and HeterMoE balances the computation on A40 and V100 with Asym-EA, which we provide a detailed breakdown in §6.4.3. Still, for O2 on Mixtral-W2, HeterMoE suffers 3% performance drop compared with EP (Ideal). This is due to the the limited speed-up of attention on A40 at 4K lengths, while expert computation is more intensive for the wider model variants. The limited depth and number of experts per GPU for Mixtral-W2 also reduce the search space for Asym-EA. In shallower models, the penalty of ZP from the 1st microbatch's first forward and first backward is more pronounced, where V100 has to wait for A40.

We show the results on AWS in Figure 8. HeterMoE still consistently outperforms all compared methods. Compared with EP (Ideal), HeterMoE achieves an average speed-up of 1.17x and up to 1.25x. Since the performance gap between L40S and T4 is much larger than that between A40 and V100, EP and DistEP experience severe performance degradation as L40S remains idle most of the time. On average, HeterMoE outperforms EP by 2.89x and DistEP by 1.96x.

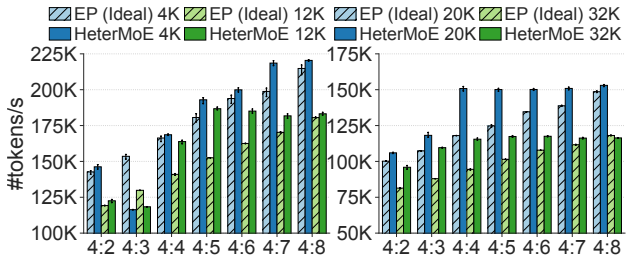
Since we use a simulated network connection of 200 Gbps on AWS, unless otherwise specified, we perform subsequent evaluations on our on-premise testbed.

In general, we find that the benefit of HeterMoE over baselines is more significant for deeper models and models with more experts, matching recent model designs [21,46] that use many smaller experts instead of a few large ones. Although we only evaluate sequence lengths up to 32K due to limited GPU resources and memory constraints, we observe that the speed-up of HeterMoE over EP also grows with increasing sequence lengths, which greatly benefits the training of long-context models.



(a) Mixtral-D1 on O1 (b) Mixtral-W2 on O2

Figure 9: **[Pipeline Parallelism]:** Performance comparison of HeterMoE's zebra parallelism with heterogeneity-aware pipeline parallelism on the on-premise testbed.



(a) 4K and 12K (b) 20K and 32K

Figure 10: **[Ablation Study]:** Impacts of GPU ratios in the setup of a ZP group to HeterMoE's effectiveness.

6.3 Comparing with Pipeline Parallelism

In this section, we study how our zebra parallelism compares to existing heterogeneity-aware training techniques [15, 42, 45]. They are mainly based on pipeline parallelism, where different pipelines stages are assigned to different GPU models. Each stage is also assigned with a different number of layers to balance the compute time, where faster GPUs are assigned with more layers.

In Figure 9, we present the training throughput of HeterMoE and heterogeneity-aware pipeline parallelism (PP) on our on-premise testbed. For setup `O1`, we spin up 6 pipeline instances, with each containing a single A40 and a single V100. Similarly, for setup `O2`, we use 4 pipeline instances, each with 1x A40 and 2x V100. Training data is distributed across pipeline instances. Each GPU in a pipeline instance corresponds to a pipeline stage. We tune the layers assigned to each stage to balance the load and maximize the throughput, under the memory limitation posed by each GPU. We perform stage balance tuning independently for each model and each sequence length.

We find that HeterMoE consistently outperforms PP across all sequence lengths, achieving an average speed-up of 1.28x and up to 1.47x. HeterMoE even outperforms PP by 7%-21% for 4K sequences, where attention-expert disaggregation provides marginal gains. PP's poor performance is due to several limitations it faces. First, PP does not distinguish between attention and expert modules. Second, the granularity of PP's load balancing is restricted to a single layer, while HeterMoE enables fine-grained load balancing for ZP with Asym-EA. Finally, limited by the GPU memory, PP may fail to achieve an optimal layer assignment that balances the compute of each stage. As PP splits model by layers, a GPU may not fit even a single MoE block. In Figure 9(b), PP cannot fit even a single layer and the activations of a single 28K or 32K sequence into V100's memory. Hence, using PP alone to split a large MoE model is often not enough, it must be used in conjunction with expert parallelism or HeterMoE's zebra parallelism to split experts within a layer.

6.4 Ablation Study

6.4.1 Impacts of GPU ratios in a ZP group

Next, we study how the GPU ratio in a ZP group impacts HeterMoE's performance. We change the ratio of A40 and V100 GPUs used in a single ZP group to study the best GPU ratio for HeterMoE under different sequence lengths. The amount of compute and communication on each GPU depends solely on the GPU ratio, not their absolute number. Hence, to control the ratio, we fix the number of A40 GPUs to 4 while changing the number of V100 GPUs. We compare HeterMoE's throughput with EP (Ideal) at different sequence lengths under each setup. We use Mixtral-D1 model architecture, but we scale the total number of experts linearly with the number of V100 to ensure that we can evenly distribute experts to GPUs for HeterMoE as well as EP (Ideal). We present the results in Figure 10.

We find that the optimal A40 to V100 ratio varies for different sequence lengths. For example, HeterMoE's effectiveness peaks at 4:5 for 12K sequences with a speed-up of 1.22x over EP (Ideal), while the peak for 4K sequences is at 4:7 with a speed-up of 1.10x. For longer sequences of 20K and 32K, HeterMoE's throughput under 4:4 even reaches that of EP (Ideal) under 4:8 with 2x the number of V100s used, with a difference within 2%. At a fixed sequence length, HeterMoE's throughput does not necessarily increase by simply increasing the relative proportion of expert GPUs (V100) in a single ZP group, as attention may instead dominate the compute time. We also note that Asym-EA is only effective at 4:2, 4:4 and 4:8, due to the divisibility requirement in §4.2. This leads to a significant 24% performance drop compared to EP (Ideal) under 4:3 for 4K sequences, while HeterMoE is 3% faster under 4:2. For a target sequence length, a ZP group should be configured using the optimal GPU ratio with enough expert GPUs to hold all experts, while setting up multiple such ZP groups to utilize all available GPUs. We have also implemented a simulator to estimate the training throughput under different ZP group setups, where in addition to compute time, we also profile the communication time of NCCL send/recv

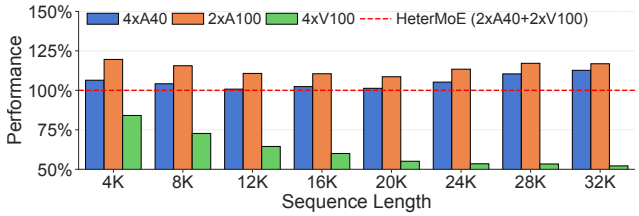
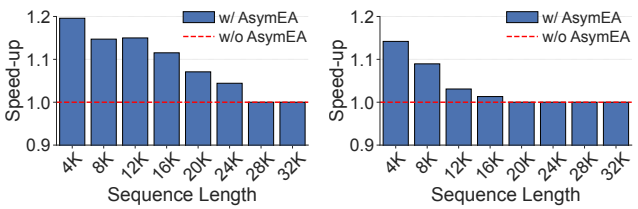


Figure 11: **[Ablation Study]:** HeterMoE's performance comparison with EP on fully homogeneous setups.



(a) Mixtral-W1 on O1 (b) Mixtral-D1 on O1

Figure 12: **[Ablation Study]:** Speed-up provided by HeterMoE's Asym-EA in terms of training throughput, compared to HeterMoE without Asym-EA.

under different message sizes.[2]

6.4.2 Comparing with fully homogeneous setups

We also study how HeterMoE compares to running EP, i.e., DeepSpeed MoE on fully homogeneous cluster setups. We compare the performance of HeterMoE using 2xA40 and 2xV100, to that of EP on 4xA40, 4xV100 and 2xA100 (80 GB). The 2xA100 are connected with PCIe Gen4. We use Mixtral-D1 model, but we set the total number of experts to 8 to match the number of GPUs. In Figure 11, we report the relative training throughput of EP compared to HeterMoE, under different sequence lengths.

We note that an A100 delivers 2.1x FP16 tensor TFLOPS than an A40 while having 2.8x memory bandwidth [30, 31]. Still, 2xA100 achieves only up to 1.20x speed-up over HeterMoE, and is only 1.14x on average. Since V100 has performance similar to A40 for computing experts according to Figure 2a, HeterMoE can efficiently harvest V100's compute. With half of the A40 replaced by V100, HeterMoE still achieves 95% the performance of 4xA40 on average. The performance gap is as little as 1%-2% for 12K-20K sequences, where HeterMoE realizes decent load balancing. V100's inefficiency on attention leads to the poor performance of 4xV100. With 2xA40 and 2xV100, HeterMoE achieves 1.66x speed-up over 4xV100 on average. Although for 4K sequences, 4xV100 still reaches 84% the performance of HeterMoE, it drops to 52% for 32K sequences.

6.4.3 Effects of asymmetric expert assignment

We study the effectiveness of Asym-EA in Figure 12 on `O1` setup, where we compare the speed-up brought by Asym-EA for two settings under different sequence lengths. Asym-EA is most effective for shorter sequences where attention's compute time T_A^{Attn} on A40 is significantly faster than the compute time T_E^{Exp} of experts on V100. For 4K sequences, Asym-EA provides 1.20x speed-up on Mixtral-W1 and 1.14x on Mixtral-D1. As the gap between T_A^{Attn} and T_E^{Exp} closes with increasing sequence lengths, the additional contribution of Asym-EA gradually reduces. For instance, the speed-up of Asym-EA decreases to 1.04x on Mixtral-W1 at 24K. For sequences longer than 20K on Mixtral-D1 and sequences longer than 28K on Mixtral-W1, Asym-EA is no longer required, as we have $T_A^{\text{Attn}} \geq T_E^{\text{Exp}}$. Instead, we should increase the number of A40 in a ZP group to decrease T_A^{Attn}.

Table 3: **[Ablation Study]:** Impacts of HeterMoE's ZP (w/o Asym-EA) and Asym-EA to GPU utilization (percentage of time spent on effective compute) for Mixtral-D1 on `O1` setup. We also include the utilization improvement against DistEP in the parentheses.

Seq. Len	Asym-EA	A40 Util.	V100 Util.
8K	✗	55% (1.69x)	89% (1.73x)
	✓	83% (2.51x)	78% (1.52x)
16K	✗	77% (1.90x)	89% (1.97x)
	✓	86% (2.12x)	84% (1.87x)

We also provide the breakdown of how HeterMoE improves GPU utilization, with and without Asym-EA. We show GPU utilization and the improvement compared to DistEP in Table 3. We find that with ZP's ability to overlap computation on attention and expert GPUs, both A40 and V100's utilization are greatly improved, as DistEP without such overlapping spends 30%-70% of training time on idle waiting. Since Asym-EA moves some expert computation to A40, A40's utilization is significantly increased, at the expense of a small decrease in V100's utilization.

7 Related Work

MoE training systems. Extensive literature has been proposed to specifically optimize MoE training with expert parallelism. MegaBlocks [7] proposes a grouped GEMM kernel to accelerate expert computation. A series of work [11,26,44,47] optimize expert placement to handle the dynamic loads on experts. All-to-all communication is also optimized, with both collective implementation optimizations [12,27,37,49,50]and compute-communication overlapping [20, 37, 49]. In partic-

[2]PyTorch's all-to-all is implemented using NCCL send/recv operations.

ular, DeepSeek [21] introduces DualPipe to overlap computation and communication within a pair of forward and backward chunks, while optimized kernels with tuned SM allocations are used for cross-node communication. HeterMoE, on the other hand, disaggregates attention and experts by taking advantage of their differences in performance characteristics. Most of these optimizations are orthogonal and can be incorporated into HeterMoE.

Heterogeneity-aware training systems. Training LLMs on heterogeneous clusters requires partitioning both data and model. Whale [15] balances the data load within a pipeline stage by considering heterogeneous compute and memory, but it only addresses uneven memory demands across pipeline stages by considering device assignment. Metis [42] and FlashFlex [45] further balance layers across stages. SDPipe [23] targets dynamic heterogeneity where GPUs suffer from uncontrollable performance variations. HAP [48] unevenly distributes workloads in both data and tensor parallelism. However, tensor parallelism requires frequent synchronization and HAP cannot overlap communication. Cephalo [1] targets FSDP and only balances compute by adjusting batch sizes. Recently, HEXA-MoE [22] is proposed for MoE training on heterogeneous GPUs. However, similar to Cephalo, it only balances compute by unevenly splitting data. These approaches are complementary to HeterMoE and can be combined with zebra parallelism.

8 Conclusion

This paper presents HeterMoE, a system for efficient Mixture-of-Experts (MoE) models training on heterogeneous GPUs. HeterMoE disaggregates attention and expert modules to fully utilize each GPU's capability. HeterMoE introduces zebra parallelism (ZP), along with asymmetric expert assignment (Asym-EA), to enable computation overlapping and fine-grained load balancing. Our evaluations show that HeterMoE consistently outperforms existing techniques, achieving up to 2.3x speedup over existing MoE training systems, while maintaining an average 95% throughput with half GPUs of newer generation in a homogeneous cluster replaced by older ones. We will open source HeterMoE.

References

[1] Runsheng Benson Guo, Utkarsh Anand, Arthur Chen, and Khuzaima Daudjee. Cephalo: Harnessing heterogeneous gpu clusters for training transformer models. *arXiv e-prints*, pages arXiv–2411, 2024.

[2] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[3] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, and Chaoning Zhang. A complete survey on llm-based ai chatbots. *arXiv preprint arXiv:2406.16937*, 2024.

[4] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[5] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[6] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[7] Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems*, 5:288–304, 2023.

[8] GitHub. Flashattention support on t4, 2023.

[9] GitHub. Flashattention support on v100, 2024.

[10] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

[11] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 120–134, 2022.

[12] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems*, 5:269–287, 2023.

[13] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. *Programming with TensorFlow: Solution for Edge Computing Applications*, pages 87–104, 2021.

[14] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R Ganger. Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 642–657, 2023.

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

[16] 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, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[17] Kyeonglok Kim, Hyeonsu Lee, Seungmin Oh, and Euiseong Seo. Scale-train: A scalable dnn training framework for a heterogeneous gpu cloud. *IEEE Access*, 10:68468–68481, 2022.

[18] Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, et al. xformers: A modular and hackable transformer modelling library, 2022.

[19] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[20] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed {MoE} training and inference with lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 945–959, 2023.

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

[22] Shuqing Luo, Jie Peng, Pingzhi Li, and Tianlong Chen. Hexa-moe: Efficient and heterogeneous-aware moe acceleration with zero computation redundancy. *arXiv preprint arXiv:2411.01288*, 2024.

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

[24] Sergio Moreno-Alvarez, Juan M Haut, Mercedes E Paoletti, Juan A Rico-Gallego, Juan C Diaz-Martin, and Javier Plaza. Training deep neural networks: a static load balancing approach. *The Journal of Supercomputing*, 76:9739–9754, 2020.

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

[26] Xiaonan Nie, Xupeng Miao, Zilong Wang, Zichao Yang, Jilong Xue, Lingxiao Ma, Gang Cao, and Bin Cui. Flexmoe: Scaling large-scale sparse pre-trained model training via dynamic device placement. *Proceedings of the ACM on Management of Data*, 1(1):1–19, 2023.

[27] Xiaonan Nie, Pinxue Zhao, Xupeng Miao, Tong Zhao, and Bin Cui. Hetumoe: An efficient trillion-scale mixture-of-expert distributed training system. *arXiv preprint arXiv:2203.14685*, 2022.

[28] NVIDIA. Nvidia hopper architecture in-depth, 2022.

[29] NVIDIA. Nvidia ethernet networking accelerates world's largest ai supercomputer, built by xai, 2024.

[30] NVIDIA. A100 datasheet, 2025.

[31] NVIDIA. A40 datasheet, 2025.

[32] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pages 18332–18346. PMLR, 2022.

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

[34] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama:

Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[35] SemiAnalysis. 100,000 h100 clusters: Power, network topology, ethernet vs infiniband, reliability, failures, checkpointing, 2024.

[36] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08608*, 2024.

[37] Shaohuai Shi, Xinglin Pan, Qiang Wang, Chengjian Liu, Xiaozhe Ren, Zhongzhe Hu, Yu Yang, Bo Li, and Xiaowen Chu. Schemoe: An extensible mixture-of-experts distributed training system with tasks scheduling. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 236–249, 2024.

[38] Siddharth Singh, Olatunji Ruwase, Ammar Ahmad Awan, Samyam Rajbhandari, Yuxiong He, and Abhinav Bhatele. A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training. In *Proceedings of the 37th International Conference on Supercomputing*, pages 203–214, 2023.

[39] CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A Choquette-Choo, Jingyue Shen, Joe Kelley, et al. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*, 2024.

[40] DataNami Team. How aws plans to cope with genai's insatiable desire for compute, 12 2023. Accessed: 2025-03-10.

[41] TechPowerUp. Nvidia "blackwell" gpus are sold out for 12 months, customers ordering in 100k gpu quantities, 2024.

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

[43] Wired. Meta's next llama ai models are training on a gpu cluster 'bigger than anything' else, 2024.

[44] Yongji Wu, Wenjie Qu, Tianyang Tao, Zhuang Wang, Wei Bai, Zhuohao Li, Yuan Tian, Jiaheng Zhang, Matthew Lentz, and Danyang Zhuo. Lazarus: Resilient and elastic training of mixture-of-experts models with adaptive expert placement. *arXiv preprint arXiv:2407.04656*, 2024.

[45] Ran Yan, Youhe Jiang, Wangcheng Tao, Xiaonan Nie, Bin Cui, and Binhang Yuan. Flashflex: Accommodating large language model training over heterogeneous environment. *arXiv preprint arXiv:2409.01143*, 2024.

[46] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[47] Mingshu Zhai, Jiaao He, Zixuan Ma, Zan Zong, Runqing Zhang, and Jidong Zhai. {SmartMoE}: Efficiently training {Sparsely-Activated} models through combining offline and online parallelization. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 961–975, 2023.

[48] Shiwei Zhang, Lansong Diao, Chuan Wu, Zongyan Cao, Siyu Wang, and Wei Lin. Hap: Spmd dnn training on heterogeneous gpu clusters with automated program synthesis. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 524–541, 2024.

[49] Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, et al. Comet: Fine-grained computation-communication overlapping for mixture-of-experts. *arXiv preprint arXiv:2502.19811*, 2025.

[50] Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. Deepep: an efficient expert-parallel communication library. https://github.com/deepseek-ai/DeepEP, 2025.