Chimera: Communication Fusion for Hybrid Parallelism in Large Language Models

Le Qin
The Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
lqin674@connect.hkust-gz.edu.cn

Junwei Cui
The Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
jcui382@connect.hkust-gz.edu.cn

Weilin Cai
The Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
wcai738@connect.hkust-gz.edu.cn

Jiayi Huang*
The Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
hjy@hkust-gz.edu.cn

Abstract

Large Language Models (LLMs), exemplified by ChatGPT, have emerged as a predominant workload in current machine learning systems. To achieve efficient training and inference within the constraints of limited single-NPU memory capacity, deploying LLMs on multi-NPU systems typically adopt a hybrid approach that combines various parallelism patterns. This hybrid parallelism within LLMs introduces a significant amount of diverse collective communications. However, these frequent blocking communications impose a substantial burden on the multi-NPU systems. Overcoming the communication bottleneck is crucial to unlocking the potential of multi-NPU systems for efficient and scalable LLM processing.

This paper introduces Chimera, a communication fusion mechanism for hybrid parallelism in LLMs. We comprehensively analyze the communication processes of each LLM parallelism pattern, identify the communication redundancy in hybrid parallelism and eliminate redundancy by fusing adjacent communication operators during parallelism transformation. By reordering operations and generating redundancy-free communication operator, Chimera effectively mitigates communication bottleneck in hybrid LLM parallelism. Our results show that Chimera achieves 1.23-7.06× network bandwidth speedup. Additionally, the end-to-end performance of LLM forward pass and backward pass on different typical multi-NPU systems achieves respective 1.32-1.58× and 1.16-1.36× speedups on average compared with those without communication fusion.

*Corresponding author.

ISCA '25, Tokyo, Japan

ACM ISBN 979-8-4007-1261-6/25/06
https://doi.org/10.1145/3695053.3731025

CCS Concepts

• **Networks** → **Network algorithms**; • **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Distributed architectures**.

Keywords

Collective Communication, Large Language Models, Distributed Training

ACM Reference Format:

Le Qin, Junwei Cui, Weilin Cai, and Jiayi Huang. 2025. Chimera: Communication Fusion for Hybrid Parallelism in Large Language Models. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3695053.3731025

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable potential in various tasks, such as reading comprehension [10], image classification [1], and code generation [9, 27, 68]. Prevailing LLMs, including BERT (110M-340M) [14], GPT-3 (175 B) [6], LlaMA (7B-65B) [67], and Mixtral 7Bx8 (7B) [25], have huge amounts of parameters and their sizes keep increasing at an extremely rapid pace [5]. Due to the slowing down of Moore's law, the memory capacity of one single NPU[1] cannot meet the requirement of storing the whole LLM model for training and inference [59]. Additionally, the immense computational demands of LLMs render the computation time on a single NPU intolerably lengthy. Therefore, parallel processing of LLM computations across multi-NPU clusters has emerged as a popular solution to accelerate the training and inference processes [2, 38, 56, 62, 71, 72].

Various parallelism patterns have been proposed by numerous proposals [32, 35, 53, 62], tailored to the characteristics of models and systems, such as data parallelism (DP) [62], tensor parallelism (TP) [62], pipeline parallelism (PP) [62], expert parallelism (EP) [35, 53], and sequence parallelism (SP[2]) [32], among others. Each of these parallelism patterns exhibits its own advantages and

[1]NPU indicates Neural Processing Unit, we use NPU to indicate GPU, TPU and other DNN dedicated accelerators.

[2]Since many studies [22, 32, 36, 37, 40] have proposed the design of "sequence parallelism", we use SP to refer to sequence parallelism in Megatron framework [32].

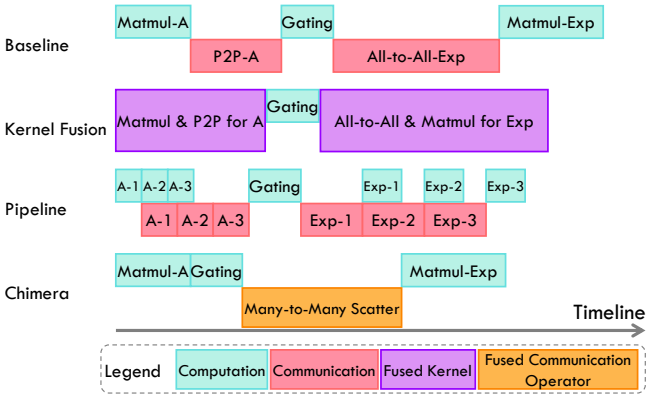


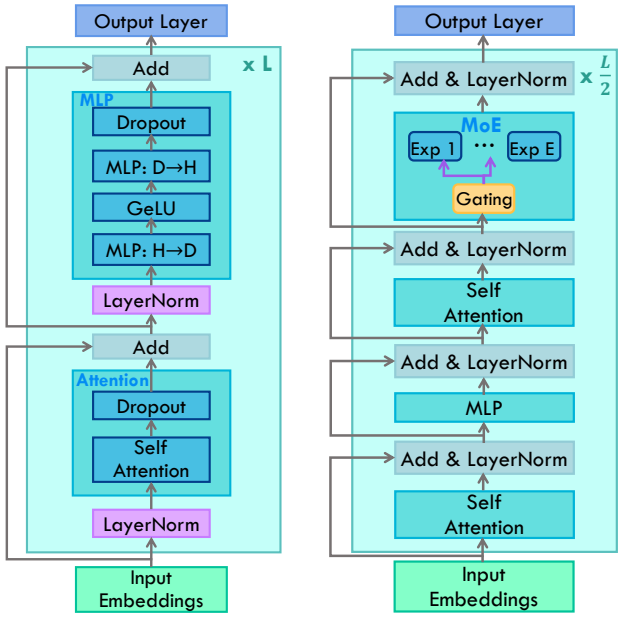
Figure 1: Different communication optimization strategies in hybrid pipeline parallelism and expert parallelism (PP+EP). Communication fusion can be combined with other strategies including kernel fusion and pipeline scheduling.

application scopes. For instance, tensor parallelism is designed for weight partitioning and parallel processing in self-attention and MLP layers, effectively reducing the memory requirement on a single NPU. As another example, sequence parallelism, targeting layer normalization and dropout layers, employs sequence partitioning and parallel processing to reduce activation recomputation. Therefore, the training and inference processes of LLMs tend to adopt different parallelism patterns at different layers according to the model's characteristics, which is called hybrid parallelism.

Communication in hybrid-parallel LLM faces two key challenges. First, different parallelism strategies introduce various communication patterns to ensure data synchronization between NPUs, leading to complex communication processes and heavy communication overhead. The blocking nature of these processes even exacerbates the bottleneck, as computations must wait for communications to complete before proceeding. Second, the diversity of hybrid parallelism types, network topologies, and hardware configurations makes developing general and flexible communication optimizations increasingly important yet challenging. The rapidly evolving landscape of hybrid parallel training/inference mechanisms [32, 35, 56, 62, 63] and LLM systems [15, 29, 30, 44, 49, 64] has led to a proliferation of customized communication solutions. However, these specialized optimizations often lose effectiveness when models or systems change significantly.

Numerous proposals have optimized the communication overhead of hybrid parallel LLMs [7, 8, 11, 16, 18, 21, 24, 26, 32, 37, 50, 54, 61, 70, 75]. Some studies adopt fine-grained kernel fusion to reduce memory accesses for adjacent computation and communication operators [18, 21, 24, 50]. Some other proposals focus on scheduling optimization for better overlap between computation and communication. For adjacent operations that have no dependency, their sequence can be flexibly scheduled without disrupting dependencies with other operators [8, 26]. For adjacent and dependent operations, they are typically scheduled through pipelining in split data chunks to achieve more fine-grained overlapping [8, 11, 37, 54, 69, 70].

However, these efforts focus on optimizing communication from the perspective of improving either memory access efficiency or computational resource utilization, lacking consideration of the



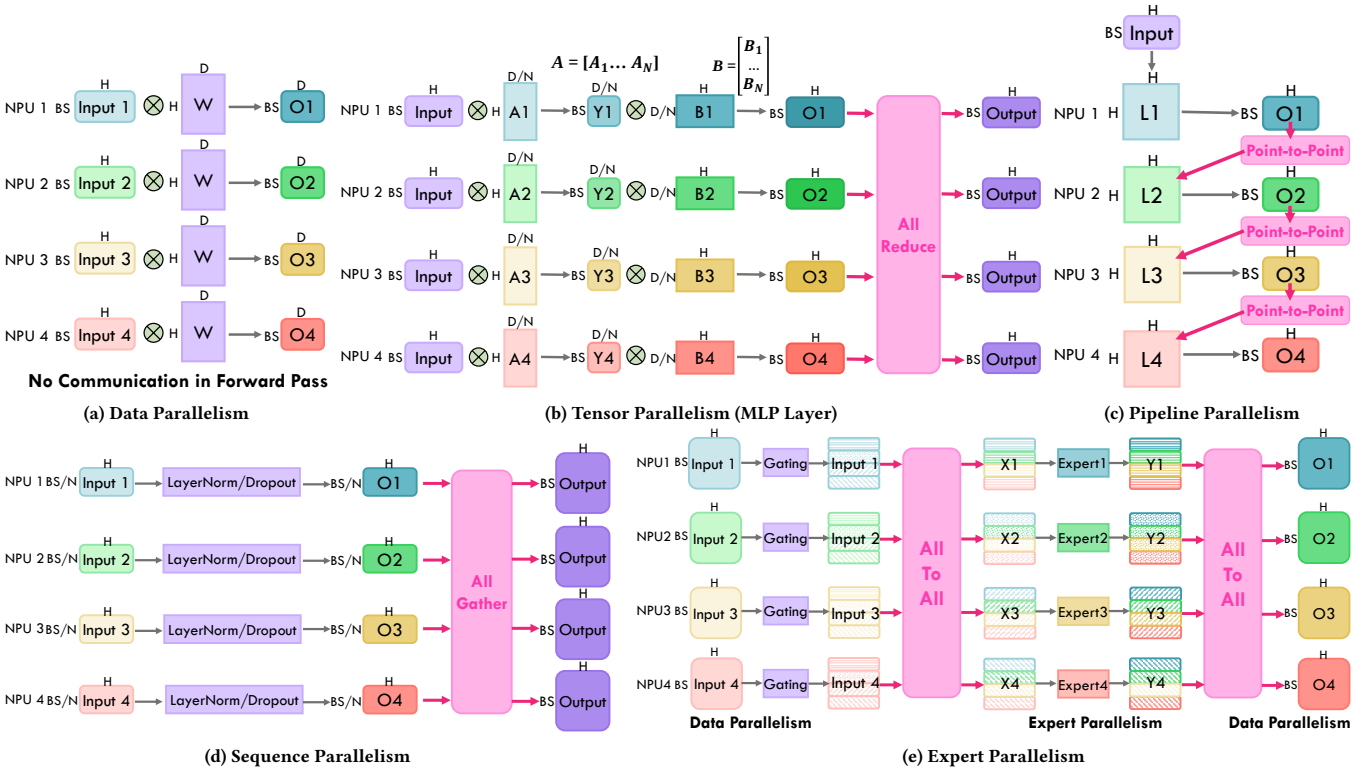
(a) Transformer Structure **(b) MoE Structure**

Figure 2: An overview of transformer and MoE architecture, omitting residual parts in subsequent discussions since they do not affect communication patterns in hybrid parallelism.

communication process itself in LLM hybrid parallelism. They are also unable to effectively address the two challenges mentioned previously. The continuously increasing data volume is squeezing the optimization space for kernel fusion and pipeline scheduling, with communication still resulting in significant time consumption. Furthermore, the ad-hoc nature of these methods leads to extensive manual customization, making them hard to generalize to evolving hybrid parallelism. Figure 1 illustrates the timeline for hybrid pipeline parallelism and expert parallelism, showing that both kernel fusion and pipeline scheduling struggle to provide significant improvements facing heavy communication overhead. Here we propose a novel optimization called communication fusion, which is different but compatible with all mentioned methods. By reordering operators and fusing adjacent communication operators, communication size could be effectively reduced, thus leading to considerable optimization.

In this work, we introduce Chimera, a communication fusion mechanism for hybrid parallelism in LLMs. We comprehensively study the characteristics and costs of communication patterns in all mainstream LLM parallelism modes. We have discovered extensive redundant communication during transitions in parallelism patterns. To this end, Chimera eliminates redundancy by fusing communication operators brought by adjacent parallelism patterns. For adjacent communication operators in the computational graph, we directly generate corresponding simplified collective communication operators. When there is computation operator between two communication operators, we reorder operators to create opportunities for adjacent communications, thereby enabling fusion. Chimera can be used in both distributed inference and training scenarios that adopt hybrid parallelism.

In summary, the contributions of this paper are as follows:



(a) Data Parallelism (b) Tensor Parallelism (MLP Layer) (c) Pipeline Parallelism

(d) Sequence Parallelism (e) Expert Parallelism

Figure 3: Computation and communication process in typical LLM parallelism patterns. The rectangles and rounded rectangles colored in blue, green, yellow, and red represent weights and activations deployed in each NPU respectively, where variations in shade denote changes within the same NPU. The purple blocks represent the same intermidiate results or layers across all NPUs. In this and following pictures, B indicates batch size, S indicates sequence length, H indicates hidden dimension, D indicates intermediate dimension in computation process, N indicates the number of NPUs involved in one parallelism.

- We scrutinize the communication patterns and identify the redundant data movement problem in hybrid parallelism of distributed LLMs, which reveals the optimization opportunities by adopting communication fusion when parallel pattern transitions.
- We propose Chimera, a communication fusion mechanism for hybrid parallel LLMs, effectively mitigating the communication bottleneck in LLM processing. To the best of our knowledge, Chimera is the first proposal that leverages fusion between communication operators for the optimization of comprehensive hybrid parallelism in LLMs.
- We conduct extensive evaluations for Chimera and demonstrate its effectiveness on common LLM models, multi-NPU systems, and hybrid parallelism modes. Results show that Chimera can bring 1.23-7.06× network bandwidth speedups for cascaded hybrid parallelism, 1.16-1.58× speedups for end-to-end forward and backward pass on average compared with no-fusion baseline, and 1.32×-3.09× speedups for communication during parallelism transitions in real machines.

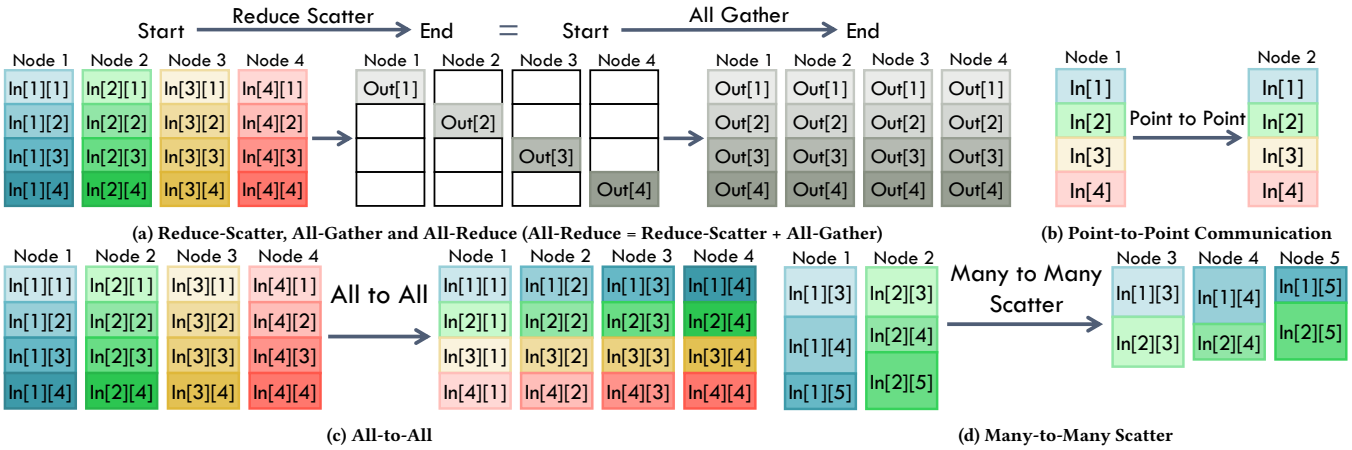
The rest of the paper is organized as follows: Sections 2 and 3 establish the background and motivation, respectively. Section 4 introduces Chimera, followed by the evaluation in Section 5. Sections 6 and 7 present discussions and related work comparisons, respectively. Finally, Section 8 concludes the paper.

2 Background

2.1 Parallelism Patterns in LLM

In the realm of Large Language Models (LLMs), the predominant architecture [6, 14, 52, 67] comprises a stack of transformer blocks, each consisting of one self-attention layer and one multilayer perceptron (MLP) layer, showing in Figure 2a. As shown in Figure 2b, recent advancements [35, 53] in the Mixture-of-Experts (MoE) model involve substituting some MLP layers in the transformer structure with an expert layer composed of numerous distinct MLPs, each termed an *expert*. When activations enter the expert layer, they first pass through a gating layer, which determines which experts to be selected. Typically, for each token, the selection of experts are based on the top-k output values of gating layer. Each of these selected k experts then computes for the token independently. After these computations, a weighted sum of the results from the selected experts is performed to produce the final output of the expert layer. Each input token may select a different set of k experts, and typically, not every expert is chosen by all tokens. This process enables dynamic routing of the tokens to the most relevant experts, optimizing the utilization of specialized computational resources and potentially enhancing the overall performance of the model.

To leverage multi-NPU systems for accelerating computations in LLMs with foundational Transformer and MoE structures, various parallelism patterns [32, 35, 62] have been proposed. Figure 3



(a) Reduce-Scatter, All-Gather and All-Reduce (All-Reduce = Reduce-Scatter + All-Gather)

(b) Point-to-Point Communication

(c) All-to-All

(d) Many-to-Many Scatter

Figure 4: Common communication patterns in LLM training and inference, $Out[j] = \sum_{i=1}^{4} In[i][j]$.

illustrates the computation and communication processes under five typical LLM parallelism patterns[3]:

(1) **Data Parallelism**: As shown in Figure 3a, data parallelism involves replicating the same model across different NPUs, speeding up the training process by dividing input tokens and assigning them to different NPUs for concurrent computation. Data Parallelism is adopted for the parallel processing of the whole model.

(2) **Tensor Parallelism**: In tensor parallelism, as depicted in Figure 3b, with each NPU processing the same input, only a portion of the model corresponding to the tensor-split is housed on each NPU, thus only part of the output is produced per NPU. For example, by column and row splitting MLP layers A and B respectively, partial results matching the correct output dimensions can be computed step-by-step. These partial results are summed through All-Reduce communication across NPUs, enabling each NPU to obtain the correct complete output. Tensor parallelism is typically applied in self-attention layers and MLP layers.

(3) **Pipeline Parallelism**: Figure 3c shows that pipeline parallelism involves deploying different layers on different NPUs, with the output from previous layers serving as the input to the following ones, transferred via point-to-point (P2P) communication for further computation. This layer processing in a pipeline manner completes the computation across the entire model. In the example of Figure 3c, each layer's dimensions are simplified to H×H, as neither self-attention nor MLP layers alter the dimensions of input activations, allowing each layer to be simplified as an H×H structure. Pipeline parallelism is generally used for the conversion of adjacent layers.

(4) **Sequence Parallelism**: As depicted in Figure 3d, sequence parallelism focuses on parallel computation of the layer normalization (LayerNorm) and Dropout layers, which are parts of self-attention and MLP layers, to reduce the recomputation cost of activations. By splitting input sequences, different sequence slices can undergo local LayerNorm or Dropout computations, and the results of these slices are gathered across NPUs via All-Gather to assemble the complete sequence output. Sequence parallelism is dedicated for LayerNorm and Dropout parts.

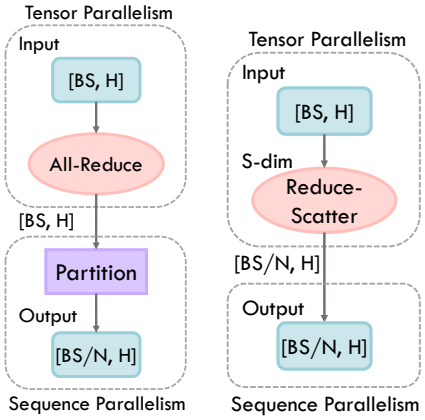
(5) **Expert Parallelism**: Figure 3e illustrates expert parallelism, which accelerates the computation process in the MoE model by placing experts across different NPUs and then directing activations to the selected experts via gating. This requires All-to-All communication to send tokens from each NPU to the NPUs hosting selected experts, and after expert computation, results are sent back to original NPUs for subsequent layer computations via another All-to-All. Notably, while expert parallelism initially denotes distributing experts across NPUs for parallel processing, as shown in the middle of Figure 3e, NPUs typically employ data parallelism before entering the expert layer in practical deployments for high throughput. Expert parallelism is suitable for the expert layers of MoE models. This paradigm has persisted since the introduction of expert parallelism in LLM training/inference [35] and has been universally adopted in subsequent studies [56, 63]. Therefore, we use expert parallelism to denote the parallelism pattern that conforms to this illustrated pattern of data parallelism + expert parallelism + data parallelism in this paper.

2.2 Collective Communication

Collective communication frequently occurs in various parallelism patterns of LLMs, and we discuss six common communication patterns, including five collective communications and one point-to-point (P2P) communication.

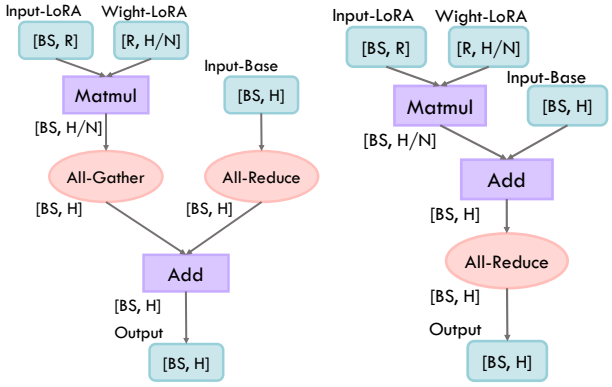
(1) **Reduce-Scatter, All-Gather and All-Reduce**: As shown in Figure 4a, data of each NPU can be divided into data chunks equal to the number of NPUs, where $In[i][j]$ represents the j_{th} data chunk on the i_{th} NPU. Reduce-Scatter enables each NPU to obtain the aggregated value of the data chunk corresponding to its NPU ID, All-Gather involves sending data from each NPU to all other NPUs, and All-Reduce ensures

[3]In this paper, we use DP, TP, PP, SP, and EP to represent data parallelism, tensor parallelism, pipeline parallelism, sequence parallelism, and expert parallelism, respectively.



(a) TP+SP Baseline (b) TP+SP Optimization

Figure 5: Computational graph of hybrid TP and SP before and after natural communication optimization for each NPU.



(a) S-LoRA Baseline (b) S-LoRA Optimization

Figure 6: Computational graph of S-LoRA before and after natural communication optimization for each NPU. Both weight of base model and LoRA adapters are partitioned and deployed across NPUs. *Input-Base* is the partial result, thus leading to All-Reduce for synchronization.

that each NPU obtains the complete aggregated data value. Theoretically, All-Reduce can be considered as a combination of Reduce-Scatter followed by All-Gather.

(2) **Point-to-Point Communication**: Figure 4b depicts point-to-point communication, where data from one NPU is sent to the other NPU.

(3) **All-to-All**: As shown in Figure 4c, All-to-All sends different data chunks from each NPU to all other NPUs, ensuring that each NPU acquires all the data chunks with the corresponding NPU ID.

(4) **Many-to-Many Scatter**: Figure 4d demonstrates Many-to-Many Scatter (M2MS), where each NPU in the first group (the first *Many*) sends different data to NPUs in another group (the second *Many*). Here, $In[i][j]$ represents the data that NPU i_{th} needs to send to NPU j_{th}, with varying data sizes for different $In[i][j]$. The similar communication pattern, termed *many-to-many multicast*, was proposed to optimize the communication in hybrid tensor and pipeline parallelism [75]. This pattern consists of a cross-mesh scatter followed by a local all-gather. In our work, we refer to cross-mesh scatter as Many-to-Many Scatter.

3 Motivation

3.1 Understanding Communication Redundancy

To provide universally effective optimization for communication in hybrid parallelism, we explore optimization possibilities from the perspective of communication volume. Unlike optimizations specific to a single collective communication algorithm or a particular network topology, optimizing communication volume can be applied to all systems, models, and collective algorithms. Parallelism transitions introduce the communication patterns required by both the preceding and subsequent parallelisms. Sequentially executing these communication patterns results in additional communication overhead. Some existing studies have made natural attempts to eliminate this communication redundancy [32, 61].

Megatron is an open-sourced PyTorch-based library for distributed LLM training which integrates DP, TP, PP, and SP for hybrid parallelism. [32, 43, 46, 62]. Figure 5 illustrates communication patterns during transitions between hybrid TP and SP in Megatron. When TP concludes, an All-Reduce is required to synchronize output activation values, as shown in Figure 5a. However, subsequent SP necessitates partitioning inputs along the sequence dimension, making the synchronization at the end of TP redundant. Consequently, Megatron naturally simplifies the communication from All-Reduce to Reduce-Scatter, as shown in Figure 5b, thereby eliminating the redundancy [32].

S-LoRA is a scalable serving system that enables the fine-tuning with thousands of LoRA (Low-Rank Adaptation) adapters. As depicted in Figure 6a, S-LoRA partitions and deploys LoRA adapters and base model across devices, getting the final result via accumulating output from both parts. The output from LoRA adapters is a partitioned slice and the output from base model is a partial sum, necessitating the introduction of All-Gather and All-Reduce to obtain respective outputs before accumulation. Figure 6b shows the computational graph with communication optimization in S-LoRA, which can be considered as a simplification of communication via reordering the Add operator before the two communication operators [61]. After reordering, All-Gather as part of All-Reduce, can be naturally omitted, thereby reducing communication overhead.

We have observed that both proposals remove the redundant communication in distributed LLM computing. However, they are based on human observations without systematically defining redundancy or providing an universal method to eliminate it. Hybrid parallelism introduces respective communication patterns for data synchronization during parallel transitions. However, the real goal of communication is to provide the correct inputs for subsequent parallelism, rather than producing correct outputs for previous parallelism. Thus, we find that communication redundancy is introduced by the synchronization of output activation values in the transitions of parallel patterns.

3.2 Modeling Communication Overhead

To have a quantitative understanding of communication overhead in each parallel pattern, we model communication patterns corresponding to TP, PP, SP, EP shown in Figure 3 as well as the Reduce-Scatter and Many-to-Many Scatter (M2MS) collectives. As shown in Table 1, *Overhead* represents the amount of data sent/received in each device, k represents the top-k value for expert selection in MoE models, and the remaining parameters are consistent with those in Figure 3. Based on the common ring algorithm [23, 74], All-Gather requires $N-1$ steps, with each step involving transmitting $1/N$ of local data size BSH for every NPU. Reduce-Scatter follows a similar pattern, thus both involve data transfers of $(N-1)\times BSH/N$, and All-Reduce is the sum of them. We assume that All-to-All in MoE models is uniform, so each token has a $(N-1)/N$ probability of being transferred to each expert, and $1/N$ probability of staying local. Each token should choose top-k experts. Therefore, the overhead for All-to-All is $(N-1)\times BSHk/N$. P2P involves sending all local data to the destination. Although M2MS scatters data to different destinations, it still needs sending all local data, hence both P2P and M2MS have the overhead of BSH.

Table 1: Communication Overhead Model

Parallelism	Communication	Overhead
TP	All-Reduce	$2(N-1)\times\frac{BSH}{N}$
PP	P2P	BSH
SP	All-Gather	$(N-1)\times\frac{BSH}{N}$
EP	All-to-All	$(N-1)\times\frac{BSHk}{N}$
/	Reduce-Scatter	$(N-1)\times\frac{BSH}{N}$
/	Many-to-Many Scatter (M2MS)	BSH

Based on the above definition of redundancy and analysis of each communication pattern, we further model the communication costs and real demands of common hybrid parallelism patterns to assess the extent of redundancy, as summarized in Table 2. *Communication Size 1 and 2* refer to the data volumes sent/received by each NPU in the earlier and later parallelism pattern, respectively. *Real Demand* represents the communication overhead after removing the synchronization of output activation values from the previous parallel pattern. TP+PP is a special case where we adopt M2MS+All-Gather to replace P2P for communication in PP, which has been proved to be a more efficient communication scheme in heterogeneous system [75]. Communication redundancy in TP+SP has been removed in prior work [32] which is not first proposed by us. It is evident that redundancy occupies a significant portion of the communication volume, and we explain the specific methods for eliminating redundancy carefully in Section IV. It is worth noting that while TP+PP, TP+EP, and PP+EP have been widely applied in hybrid parallelism, no existing work has identified or defined the communication redundancy within these combinations. Although SP+PP and SP+EP have not yet been explicitly used in practice, analyzing and optimizing the communication overhead during their parallelism transitions can still contribute to more diverse and flexible deployments in the future.

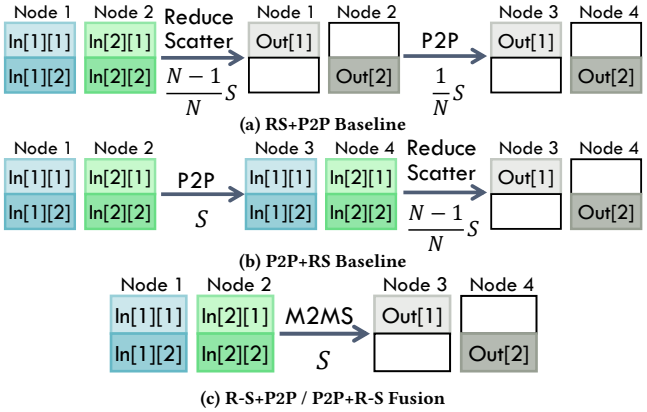


Figure 7: Communication overhead for adjacent P2P and Reduce-Scatter before and after fusion, $Out[j]=\sum_{i=1}^{2} In[i][j]$.

4 Chimera

In this section, we first introduce our method to remove communication redundancy by adopting communication fusion in the computational graph. Then we carefully illustrate how to apply the method in hybrid parallelism of LLM with two case studies.

4.1 Communication Fusion

To fully eliminate communication redundancy in hybrid parallelism, we fuse the communication operators introduced by consecutive parallel patterns to produce the simplest necessary communication operators. The process can be divided into three steps. First, we split all All-Reduce operators into Reduce-Scatter and All-Gather operators to allow for finer-grained fusion. After splitting, our five basic communication operators are Reduce-Scatter, All-Gather, All-to-All, P2P, and M2MS. Then, we perform reordering of the computation operators. In the computational graph of hybrid parallelism, there are instances where computational operators separate adjacent communication operators, preventing them from being adjacent. To further create opportunities for fusion, we reorder the communication operators. For existing parallel patterns, reordering is only needed for the Gating operators in MoE models. Gating operators are replicated and deployed in every NPU and solely responsible for calculating the score of each input activation corresponding to each expert, thereby determining the expert selection result without altering the activation values themselves. Consequently, the reordering won't introduce inequivalence problem for the computational graph. Finally, we replace adjacent communication operators with fused operators to complete the fusion. The fused operator achieves the final communication objectives, the real demand, of the original two communication operators and eliminates unnecessary output synchronization from the previous parallel pattern, thereby saving communication volume. Fused operators can also be included in the same five basic operators, which reduces the implementation cost of dedicated operators.

Table 3 summarizes all possible combinations of the five basic communication operators and their corresponding fused communications. For more complex communication patterns, they can be broken down into these five operators for step-by-step fusion. In terms of communication volume, we evaluate the changes before

Table 2: Analysis of communication overhead in all common cascaded parallelism. Ratio represents the proportion of actual required communication volume to current size, and it changes with parameters such as k and N. Note that N_1 and N_2 indicate the respective number of involved NPUs for former and latter parallelism. For PP+EP and SP+PP, we use N as N_1 equals to N_2.

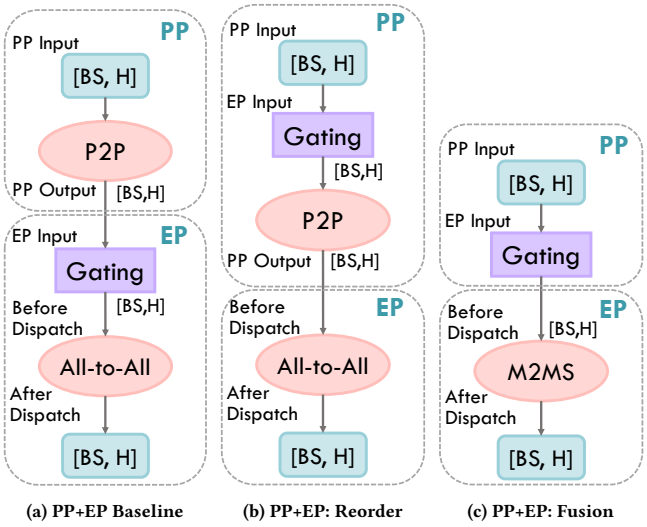
Hybrid Parallelism	Communication Size 1	Communication Size 2	Real Demand	Ratio
TP + SP [32]	/	$2(N-1)\times\frac{BSH}{N}$	$(N-1)\times\frac{BSH}{N}$	50%
TP + PP [75]	$2(N_1-1)\times\frac{BSH}{N_1}$	$\frac{BSH}{N_1}+(N_2-1)\times\frac{BSH}{N_2}$	$BSH+(N_2-1)\times\frac{BSH}{N_2}$	60% – 80%
TP + EP	$2(N_1-1)\times\frac{BSH}{N_1}$	$(N_2-1)\times\frac{BSHk}{N_2}$	$(N_1-1)\times\frac{BSH}{N_1}+(N_2-1)\times\frac{BSHk}{N_2}$	66.7% – 100%
PP + EP [16]	BSH	$(N-1)\times\frac{BSHk}{N}$	BSH	0% – 66.7%
SP + PP	$(N-1)\times\frac{BSH}{N}$	BSH	BSH	50% – 66.7%
SP + EP	$(N_1-1)\times\frac{BSH}{N_1}$	$(N_2-1)\times\frac{BSHk}{N_2}$	$(N_2-1)\times\frac{BSHk}{N_2}$	50% – 100%

Table 3: Fused communication for all the combinations of adjacent communications, where those with ★ indicate fusions used in hybrid parallelism. Here we use R-S, A-G, A-R, A2A as respective abbreviations for Reduce-Scatter, All-Gather, All-Reduce, and All-to-All.

Way 1			Way 2		
Adjacent Comm	**Fused Comm**	**Overhead**	**Adjacent Comm**	**Fused Comm**	**Overhead**
R-S+A-G	A-R	equal	A-G+R-S	Zero	lower
R-S+P2P	M2MS	equal	P2P+R-S	M2MS	lower
R-S+M2MS	M2MS	equal	M2MS+R-S	M2MS	lower
R-S+A2A	R-S	lower	A2A+R-S	R-S	lower
A-G+P2P	M2MS	lower ★	P2P+A-G	M2MS	equal
A-G+M2MS	M2MS	lower ★	M2MS+A-G	M2MS	equal
A-G+A2A	A2A	lower ★	A2A+A-G	A2A	lower
P2P+M2MS	M2MS	lower	M2MS+P2P	M2MS	lower
P2P+A2A	M2MS	lower ★	A2A+P2P	M2MS	lower ★
M2MS+A2A	M2MS	lower	A2A+M2MS	M2MS	lower
R-S+R-S	N/A	equal	A-G+A-G	A-G	lower
P2P+P2P	P2P	lower	M2MS+M2MS	M2MS	lower
A2A+A2A	A2A	lower ★			

and after fusion. We find that, except for two cases that cannot be further fused (R-S + A-G and R-S + R-S), nearly all possible fusions offer optimization. Those marked with an asterisk (★) are applicable to the actual hybrid parallelism fusions shown in Table 2.

Taking adjacent Reduce-Scatter and P2P as an example, we use Figure 7 to explain the communication overhead. Figure 7a shows the communication process of R-S+P2P, where initially node 1 and 2 perform Reduce-Scatter, each NPU sending and receiving $(N-1)S/N$ data. Upon completion, node 1 and 2 each performs respective point-to-point transmission to node 3 and 4, and S/N data is transferred in each transmission. Communication size in the whole process is S. Similarly, Figure 7b illustrates the process of P2P+R-S, with a total volume of $2-S/N$. Communication fusion in Figure 7c replaces R-S+P2P or P2P+R-S with M2MS, removing



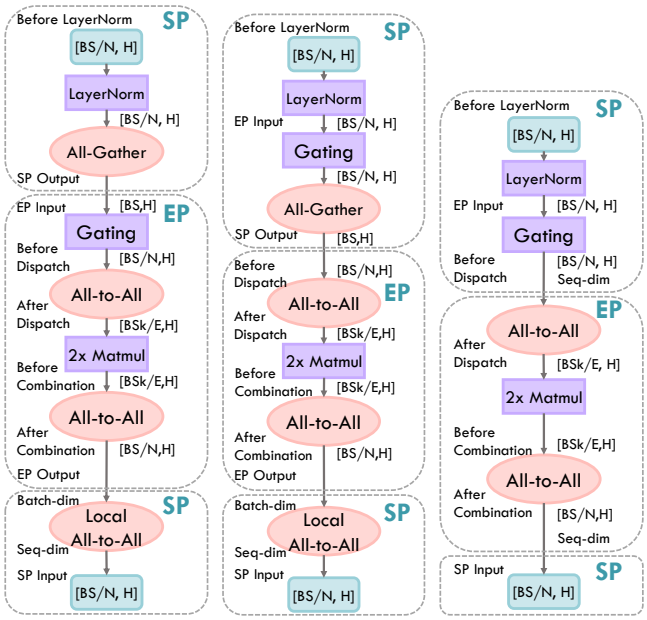
(a) PP+EP Baseline (b) PP+EP: Reorder (c) PP+EP: Fusion

Figure 8: Communication fusion process for PP+EP

the redundancy for achieving intermediate states shown in Figure 7a and 7b. Node 1 sends $In[1][1]$ and $In[1][2]$ to node 3 and 4, respectively, with a communication volume of S per NPU. The process remains the same for node 2. We find that the communication volume after fusion remains unchanged compared to R-S+P2P, but significantly reduced compared to P2P+R-S. This indicates that the former does not introduce additional communication volume for the intermediate state, whereas the latter does. In this way, we summarize all patterns in Table 3.

4.2 Case Study I: PP+EP

Figure 8 illustrates the computational graphs before and after fusion for hybrid pipeline parallelism and sequence parallelism during parallel pattern transitioning. In Figure 8a, P2P communication transfers the output from PP to the input of EP. Since a gating layer separates P2P from All-to-All, we first reorder the gating operator before P2P, as shown in Figure 8b. The reordering of gating does not affect the correctness of computational graph because the gating layer only determines the destination without altering data.



(a) SP+EP+SP Baseline (b) SP+EP+SP: Reorder (c) SP+EP+SP: Fusion

Figure 9: Communication fusion process for SP+EP+SP

Thus, the inputs and outputs of the gating layer remain consistent in both value and dimension. As Table 3 indicates, P2P+All-to-All can be fused into M2MS and bring lower overhead. Therefore, Figure 8c replaces two original communication operators with M2MS, eliminating the redundancy introduced by P2P in synchronizing PP output. Note that communication fusion changes the original data layout by deploying the gating operator on the NPU where the previous layer resides.

4.3 Case Study II: SP+EP+SP

Figure 9a displays the computational graph transitioning from sequence parallelism to expert parallelism and back to sequence parallelism. This involves four communication operators: an All-Gather for synchronization at the end of SP, two All-to-Alls for expert dispatch and combination in EP, and a local All-to-All at the end of EP to transform data from the batch dimension to the sequence dimension. In EP, inputs are split first by batch dimension before entering the gating layer, and the output from gating is used as the input for the first All-to-All dispatch. After the second All-to-All, each NPU still receives data slices across the batch dimension, which is the partial batches across complete sequence length. The subsequent SP requires the inputs to be split along the sequence dimension, hence introducing a local All-to-All within the SP group. As shown in Figure 9b, we reorder the gating operator and ensure that communication operators corresponding to the two parallel pattern transitions are adjacent. Subsequently, we perform communication fusion, resulting in the new computational graph shown in Figure 9c. Notably, by directly splitting along the sequence dimension at the gating layer, we eliminate the local All-to-All used for dimension transformation, thereby removing the redundant communication.

5 Evaluation

5.1 Methodology

To conduct a comprehensive evaluation of Chimera's performance, our experiments comprise both simulation and performance evaluation on real machines. We conduct synthetic experiments to validate the improvements of network bandwidth with communication fusion. To demonstrate the effectiveness of Chimera as the network bandwidth scales, we conduct sensitivity study. Additionally, we evaluate the performance speedups in real workloads, accessing the impact of Chimera's enhancements in end-to-end forward pass and backward pass. These three experiments are simulation-based and aim to demonstrate Chimera's broad applicability across various typical network topologies and scales. Finally, we implement and evaluate Chimera's communication optimization of parallelism transitions on real system to validate the communication performance improvements in practical scenarios.

Table 4: System Configurations

Parameter		Configuration
PE	MAC array	256×256
	Dataflow	Output Stationary
	Precision	32 bits
Accelerator	Number of PEs	16
	Clock	1 GHz
Network	Number of Accelerators	small: 5×5, 4×4, 2×2×2, 8×2 large: 8×8, 8×8, 4×4×4, 32×2
	Topology	2D-Mesh, 2D/3D-Torus, Fat-Tree
	Algorithm	Ring-2D, Halving Doubling
	Flow Control	Virtual Cut-Through
	Router Clock	1 GHz
	Number of VCs	4
	VC Buffer Depth	318 flits
	Data Packet Payload	256 Bytes
	Bandwidth	50 GBps
	Link Latency	100 ns

In our simulation, we utilize SCALE-Sim v2 [57, 58] and BookSim [28] for the estimation of computation time and interconnection modeling, respectively. We use SCALE-Sim to simulate both forward propagation and aggregation time for Reduce-Scatter and All-Reduce. We configure a TPU-like accelerator with 16 processing elements (PEs) and each PE has a 256×256 systolic array. We ensure the accelerator achieves peak compute throughput by utilizing double buffering and sufficient memory bandwidth.

We implement a Python interface to connect BookSim through network interface, which implements the communication scheduling with given algorithm. To demonstrate the effectiveness and generality of Chimera across system configurations, we study several typical topologies used in parallel and distributed machine learning systems, including 2D Mesh [19, 39, 49, 64], 2D Torus [31], 3D Torus [30], and Fat-Tree (similar to NVIDIA DGX-2 [20]). For the network size of each topology, we also adopt common and practically used configurations like 5×5 for mesh [64], 4×4 for 2D torus [31], 2×2×2 for 3D torus [30], and 8×2 for fat-tree [20]. We also scale up the target platforms to 64 nodes for each topology with 8×8 mesh, 8×8 2D torus, 4×4×4 3D torus [30], and 32×2 for fat-tree. For All-Reduce, Reduce-Scatter, All-Gather, and All-to-All

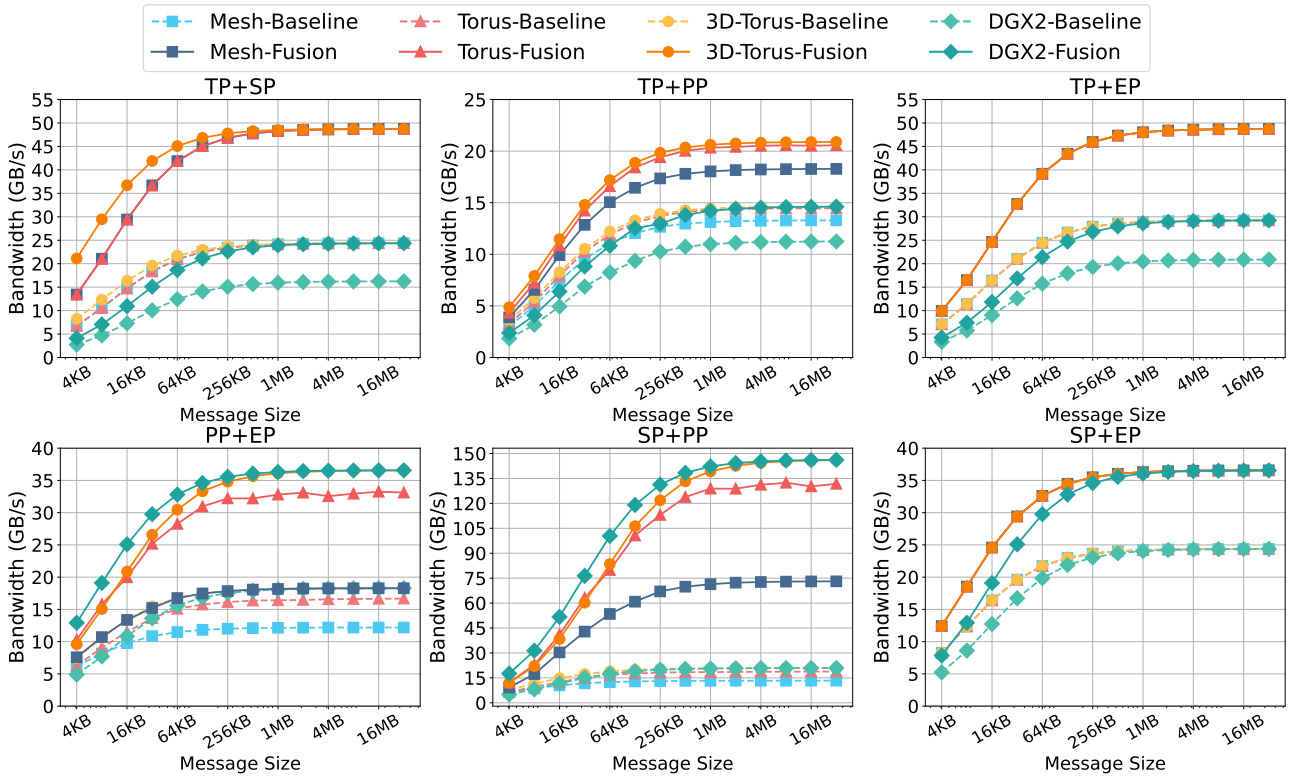


Figure 10: Effective Bandwidth with the Scaling of Message Size

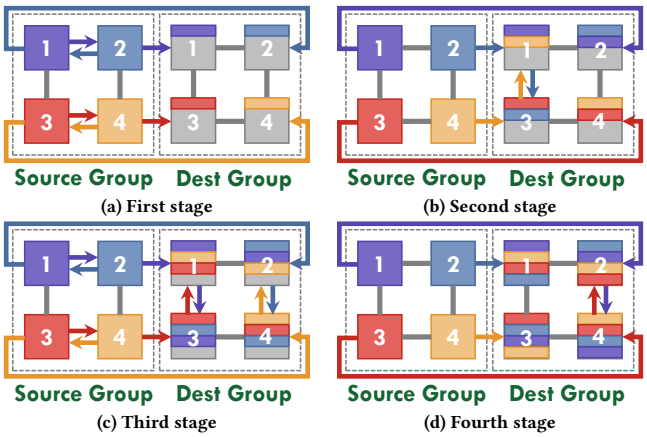


Figure 11: An example of M2MS on 2D torus, which shows the 4-stage data transmission for every node in the source group.

collectives, we implement common Hierarchical Ring [33] algorithm for mesh and torus, Halving Doubling [66] for fat-tree. For M2MS, we implement an algorithm based on asynchronous point-to-point communication. Taking an M2MS between group 0 with M points and group 1 with N points as an example, each node i in group 0 needs to perform a separate point-to-point transfer to each point in group 1, resulting in N transmissions. For the order of these point-to-point communications, we orchestrate the destinations from $i\%N$, $(i+1)\%N$, ... up to $(i+N-1)\%N$. In this way, we can mitigate the possibilities of one-to-many or many-to-one data transmission. We can further reduce network congestion and enable efficient M2MS communication by combining dimension-order routing (DOR) with random direction routing for equidistant paths. We show the example of contention-free M2MS implementation on 2D torus in Figure 11. All the system configuration parameters are listed in Table 4.

For performance evaluation on real system, we conduct experiments on one 8×RTX 4080 GPU node with PCIe 4.0 interconnection. GPUs within the node are connected to the host via PCIe 4.0×16, offering a total data transfer bandwidth of 32 GB/s. We implement communication operators with PyTorch 2.5.1 [3], CUDA 12.4 [41], and NCCL 2.21.5 [45]. Through extensive experiments with each hybrid parallelism mode, we validate the benefits of Chimera in a practical deployment.

5.2 Synthetic Experiments

We evaluate the effective bandwidth on different topologies with and without communication fusion, respectively. The parallel degrees for various parallelism patterns are: TP=4, SP=4, EP=4, DP=2, PP=2. Parallel degree indicates the number of NPUs involved in each parallelism pattern. Take TP+SP as an example, we evaluate the time consumption for the transitioning of parallelism pattern from TP to SP on four NPUs. The shapes of parallelism group are 2×2 for mesh, 2D torus and 3D torus, and 4×1 for fat-tree. We define effective bandwidth as:

$$\textit{Effective Bandwidth} = \textit{Message Size} \,/\, \textit{Time}$$

With the increasing of message size, effective bandwidth is gradually saturated. Across all topologies and hybrid parallelism patterns, communication fusion can achieve considerable speedup as shown in Figure 10. For saturate points of TP+SP, TP+PP, TP+EP, PP+EP, SP+PP, and SP+EP, the effective bandwidth can achieve the respective speedup of 1.50-2.56×, 1.26-1.43×, 1.27-1.67×, 1.23-2.65×, 1.42-7.06×, 1.49-1.50×. The results align well with our theoretical analysis in Table 2 and Section 3.2. For SP+PP, our fused M2MS can further alleviate the link contention in the network, leading to a higher speedup brought by reducing redundancy.

To better understand the improvements achieved by Chimera with the evolving of network, we conduct sensitivity study of effective bandwidth to link bandwidth. The results demonstrate the good linear scalability. The scalability with the number of nodes is more related to the implementation of communication patterns including scheduling strategies and collective algorithms [23, 55].

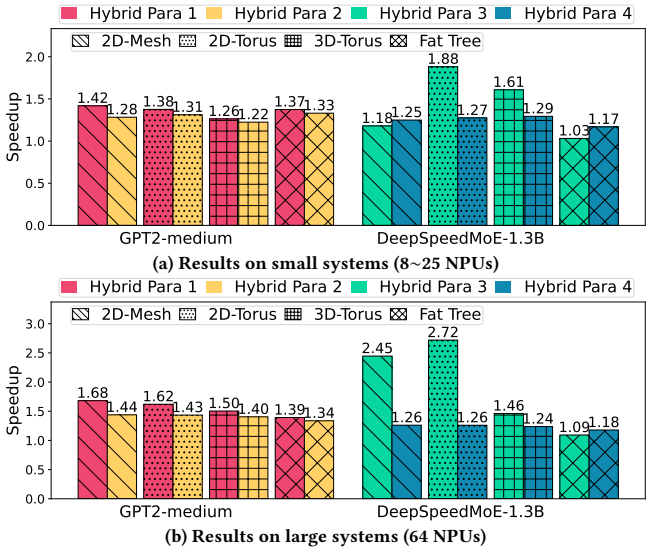
5.3 End-to-End Performance

We simulate end-to-end forward pass for common transformer-based and MoE models including GPT2 and DeepSpeedMoE [52, 53]. We find the parameters of the released model and feed them as input to the simulator [12, 42]. We set B=1, S=256 for the forwarding process and other parameters align with the models. We implement four complicated hybrid parallelism *HP1-HP4* as summarized in Table 5. *HP1* and *HP2* are aimed at GPT2 and both adopt TP+SP in each transformer block. They vary from the way combined with PP. *HP3* and *HP4* target DeepSpeedMoE, introducing SP+EP and TP+EP, respectively. For GPT2, parallel degree of TP and SP are both 2, while the degree of PP varies with the topology: the 5×5 mesh has 12 pipeline stages, the 4×4 torus and 8×2 fat-tree have 8 stages, and the 2×2×2 torus has 4 stages. For DeepSpeedMoE, DP and EP have the same parallel degree. PP is 6 for mesh, 4 for 2D torus and fat-tree, and 2 for 3D torus.

Table 5: Hybrid Parallelism in End-to-End Simulations

Type	Parallelism Transitions	DP	TP	SP	PP	EP
HP 1	TP+SP, SP+PP	1	2	2	12/8/4	0
HP 2	TP+SP, TP+PP	1	2	2	12/8/4	0
HP 3	TP+SP, SP+EP, SP+PP	2	2	2	6/4/2	2
HP 4	TP+SP, TP+EP, TP+PP	2	2	2	6/4/2	2

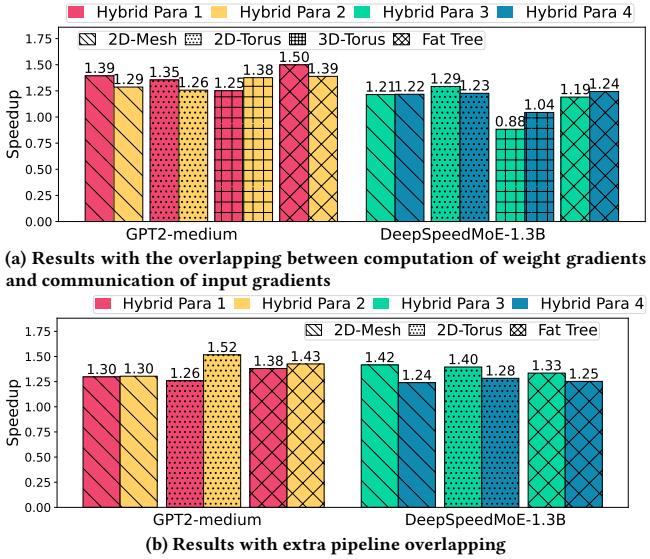
We evaluate Chimera's performance for LLM acceleration through two forward and two backward scenarios. As shown in Figure 12a, we assess the end-to-end speedup across different topologies and hybrid parallelism configurations for two model inference processes. Chimera achieves an average speedup of 1.32× and 1.34× for GPT2 and DeepSpeedMoE, respectively. We also test larger-scale configurations (with bigger TP/SP degree), as described in Section 5.1, yielding speedups of 1.48× and 1.58× for GPT2 and DeepSpeedMoE, as shown in Figure 12b. For the backward pass scenario, we explore two kinds of overlapping strategies across topologies. First, we overlap weight gradient computation with input gradient communication, which reduces the overall execution time but does not solely target communication, resulting in speedups of 1.35× and 1.16× for GPT2 and DeepSpeedMoE, respectively, as shown in



(a) Results on small systems (8~25 NPUs)

(b) Results on large systems (64 NPUs)

Figure 12: End-to-End Speedups for Forward Pass on Common Models.



(a) Results with the overlapping between computation of weight gradients and communication of input gradients

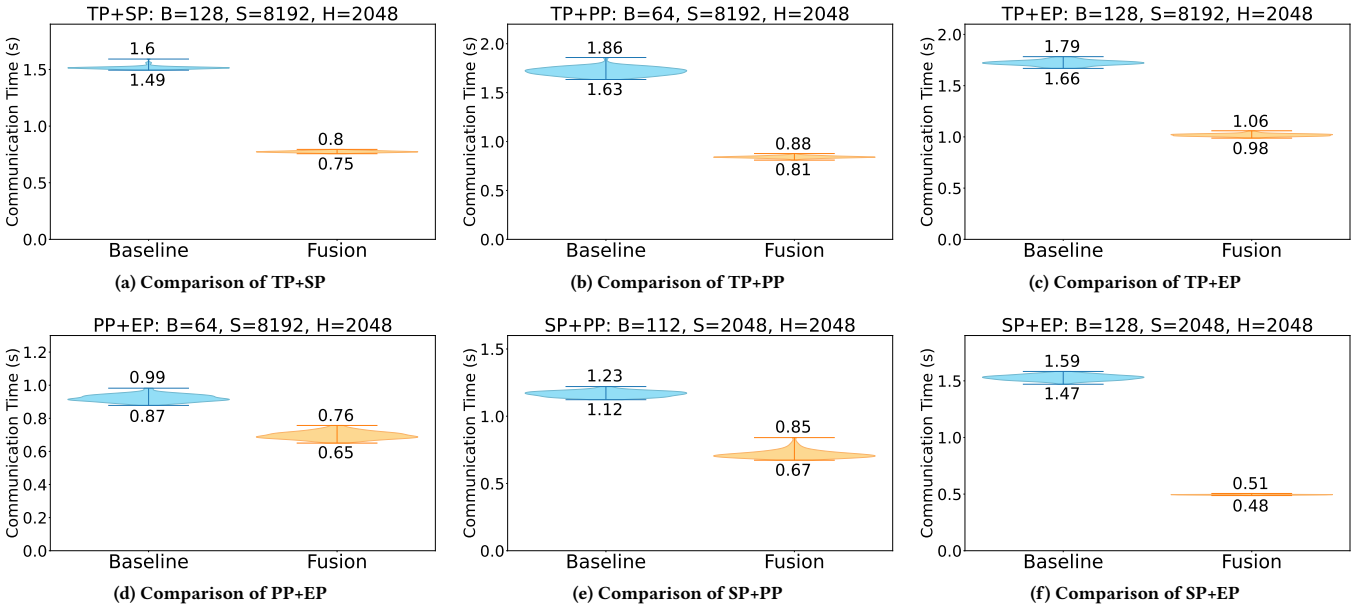
(b) Results with extra pipeline overlapping

Figure 13: End-to-End Speedups for Backward Pass on Common Models.

Figure 13a. Next, we implement a two-stage pipeline for the backward process that overlaps both communication and computation via dependency-checking. As illustrated in Figure 13b, the average speedups achieved are 1.36× and 1.32× for the two models.

5.4 Real Performance

To prove the practical effectiveness of Chimera on real system, we evaluate the performance speedup when parallelism pattern transitions. We run the experiment for 100 times with the same configuration for each hybrid parallelism under typical parameter settings and show the results by violin graph as shown in Figure 14. We adopt hybrid parallelism with the configurations as shown in



(a) Comparison of TP+SP (b) Comparison of TP+PP (c) Comparison of TP+EP

(d) Comparison of PP+EP (e) Comparison of SP+PP (f) Comparison of SP+EP

Figure 14: Communication fusion performance in one 8×RTX4080 GPU node with PCIe 4.0 interconnection

Table 6: Real Performance Evaluation

Hybrid Parallelism	Input Shape	Speedup
[TP, SP] = [8, 8]	[128, 8192, 2048]	1.92 - 2.03× (Avg. 1.96×)
[TP, PP] = [4, 2]	[64, 8192, 2048]	1.93 - 2.22× (Avg. 2.04×)
[TP, EP] = [4, 2]	[128, 8192, 2048]	1.62 - 1.77× (Avg. 1.69×)
[PP, EP] = [2, 4]	[64, 8192, 2048]	1.20 - 1.45× (Avg. 1.32×)
[SP, PP] = [4, 2]	[112, 2048, 2048]	1.38 - 1.74× (Avg. 1.63×)
[SP, EP] = [4, 2]	[128, 2048, 2048]	2.97 - 3.20× (Avg. 3.09×)

Hybrid Parallelism of Table 6. *Input Shape* indicates the shape of input tensor per GPU and *Speedup* demonstrates the range and average value of speedup brought by Chimera. The performance also accords well with analysis in Table 2 where SP+PP and SP+EP could achieve extra optimization with less contention.

6 Discussion

6.1 Application Scenarios of Chimera

Chimera is a communication fusion mechanism designed to eliminate communication redundancy in hybrid parallelism for LLMs, making it applicable to training, inference, and even fine-tuning scenarios that adopt hybrid parallelism. In training, the process is divided into three stages: forward propagation, backward propagation, and weight update [8]. Chimera can optimize the communication of activation values during forward propagation and input gradient communication during backward propagation when transitioning between parallelism strategies, as evaluated in Section 5.3. However, during the weight update stage, if data parallelism is used, the All-Reduce operation for synchronizing the weight gradients cannot be fused, as there are no other communications.

Due to the autoregressive nature of LLM inference, the process is divided into two phases: *prefilling* and *decoding* [48, 76]. The *prefilling* phase processes the user's prompt, which consists of multiple tokens, to generate the first token of response in a single step. The *decoding* phase then generates one token at a time based on the previously generated tokens, continuing until a termination token is produced. With the growing demand for longer user inputs, some studies use sequence parallelism (SP) during the prefill stage to alleviate memory pressure caused by long sequences [13, 51]. For example, DeepSeek v3 combines DP, TP, SP, and EP to efficiently handle the *prefilling* phase [13]. Chimera can be applied to accelerate such scenarios.

The benefits of Chimera extend beyond large language models to other transformer-based multi-modal models employing hybrid parallelism, enabling its broader applicability. For instance, TP and PP are used in large-scale Vision Transformer (ViT) training [65]. Models like Qwen2.5-VL [4], which process speech and video frames as long sequences, further increase the demand for multiple parallelism strategies. Therefore, alleviating distributed communication bottlenecks becomes more urgent. Chimera can be seamlessly extended to these scenarios, as its communication fusion does not require introducing new communication operators or custom optimizations.

6.2 Chimera on Heterogeneous Network

Heterogeneous networks are commonly used in current machine learning systems, where intra-node interconnection bandwidth is usually significantly higher than inter-node bandwidth, and network topologies across different hierarchies may also vary [47]. Chimera is also applicable to these heterogeneous networks and can bring benefits. This is because, unlike some approaches that

trade inter-node communication with more intra-node communication, Chimera fundamentally provides optimization by reducing communication volume. The reduction in communication volume is orthogonal to the network heterogeneity, allowing Chimera to be more broadly adaptable to various network topologies and configurations. Meanwhile, Chimera can be combined with efforts which optimize communication in heterogeneous networks [53, 55, 73, 75]. After eliminating communication redundancies, it can further optimize heterogeneous communications.

6.3 Relationship with Other Work

Existing proposals optimize communication mainly by three kinds of methods: computation-communication kernel fusion for fewer memory accesses [21, 24, 50], fine-grained scheduling for better computation-communication overlap [8, 26], and algorithm design for common collective communications [11, 15, 17, 23, 34, 55, 60, 73]. These methods can hardly coordinate to achieve synergistic optimization effects. Kernel fusion may overlook the potential optimization space of a wider range graph-level scheduling. Collective algorithms allocate links and workloads for each timestamp, which could conflict with fine-grained scheduling. Chimera offers a distinctly different optimization view from these mentioned methods by defining and identifying communication redundancies and eliminating them through the fusion of communication operators.

Chimera differs from kernel fusion from three aspects. First, they perform different operations. Kernel fusion combines adjacent computational operators (e.g., GEMM) or adjacent computation and communication operators into a customized kernel, while Chimera reduces communication redundancy by replacing two communication operators with a single communication operator. Second, their implementation methods differ. Kernel fusion requires customizing kernels for different applications, and changes in the computation type and size involved in the application can lead to a decrease in kernel performance. Chimera, on the other hand, replaces adjacent communication operators with a general-purpose communication operator, without the need to customize operators for different applications. Third, their goals are different. Kernel fusion can reduce redundant data access when executing adjacent computations separately, while Chimera eliminates redundant communication between adjacent communications.

Notably, Chimera is compatible with all previous methods. This is because Chimera removes redundancies simply through operator reordering and fusion, without introducing new operators. This means that communication operators after fusion can still be fused with adjacent computation operators, allowing for further partitioning and overlapping at the computational graph level or optimizing algorithm for fused communication operator. By combining Chimera with existing methods, communication bottlenecks in LLM hybrid parallelism can be further addressed.

7 Related Work

7.1 Hybrid Parallelism in LLM

Megatron framework enables DP, TP, PP and SP for LLM to accelerate the training process [32, 43, 62]. Megatron enables some dedicated communication optimizations including using All-Gather to replace All-Reduce for hybrid tensor and sequence parallelism [32]. DeepSpeed-TED adopts hybrid tensor, expert and data parallelism to enable efficient training of larger MoE models [63]. Alpa automates inter- and intra- operator parallelism with better partition and mapping results for hybrid tensor and pipeline parallelism [75]. Megascale scales LLM training to more than 10,000 GPUs with hybrid data, tensor, and pipeline parallelism [29].

7.2 Kernel Fusion

CoCoNet segments All-Reduce collective and adjacent computation operators to generate fused operator for training with data, tensor, and pipeline parallelism, leading to higher efficiency [24]. Operators with compatible iteration space implementation can be fused in transformer-based models for sufficient data reuse [21]. Self-contained GPU kernel can fuse All-to-All collective and adjacent computations for emerging MoE models to maintain high ALU utilization [50]. All these efforts are dedicated for computation-computation or computation-communication fusion without exploring the possibility for communication-communication fusion.

7.3 Communication Scheduling

Centauri carefully partitions communication workloads for computation - communication overlap in operation, layer and model level [8]. Lancet enables non-MoE computations to overlap All-to-All with fine-grained partitioning and pipeline scheduling, reducing the communication bottleneck in MoE models [26]. Themis proposes a efficient scheduling method for All-Reduce collective across multi-dimensional heterogeneous systems with data chunk overlapping [55]. C-Cube implements All-Reduce algorithm by overlapping Reduce-Scatter and All-Gather dedicated for Nvidia DGX-1 [47] systems [11]. These proposals focus on overlapping communication with computation or communication, without rethinking the optimization space by removing communication redundancy.

8 Conclusion

In this paper, we identify the communication bottleneck in current hybrid parallelism LLM training and inference. We carefully analyze various communication patterns and define the communication redundancy for data synchronization during parallel transitions. To alleviate the bottleneck, we propose Chimera mechanism that fuses adjacent communication operators and generates redundancy-free operator for existing and potential hybrid parallelism. The evaluation shows that communication fusion can bring 1.23-7.06× bandwidth speedup for communication during parallel transitions and the results have a good scalability to link bandwidth and latency. Furthermore, Chimera gains 1.16-1.58× speedup on end-to-end tests for complex hybrid parallelism, on average.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This work was supported in part by the National Key R&D Program of China (No. 2024YFB4505800), the National Natural Science Foundation of China (No. 62402411), and the Guangdong Provincial Project (No. 2023QN10X252). This research was conducted on the High-Performance Computing Platform of HKUST(GZ).

References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millicah, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. 2022. Flamingo: A Visual Language Model for Few-Shot Learning. *Advances in Neural Information Processing Systems* 35 (2022), 23716–23736.

[2] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '22)*. IEEE, 1–15.

[3] 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, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. 2024. PyTorch 2: Faster Machine Learning through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. 929–947.

[4] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. 2025. Qwen2.5-VL Technical Report. *arXiv preprint arXiv:2502.13923* (2025).

[5] Yoshua Bengio, Geoffrey Hinton, Andrew Yao, Dawn Song, Pieter Abbeel, Yuval Noah Harari, Ya-Qin Zhang, Lan Xue, Shai Shalev-Shwartz, Gillian Hadfield, Jeff Clune, Tegan Maharaj, Frank Hutter, Atılım Güneş Baydin, Sheila McIlraith, Qiqi Gao, Ashwin Acharya, David Krueger, Anca Dragan, Philip Torr, Stuart Russell, Daniel Kahnemann, Jan Brauner, and Sören Mindermann. 2023. Managing AI Risks in an Era of Rapid Progress. *arXiv preprint arXiv:2310.17688* (2023).

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models Are Few-Shot Learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901.

[7] Weilin Cai, Juyong Jiang, Le Qin, Junwei Cui, Sunghun Kim, and Jiayi Huang. 2024. Shortcut-connected Expert Parallelism for Accelerating Mixture of Experts. *arXiv preprint arXiv:2404.05019* (2024).

[8] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. 2024. Centauri: Enabling Efficient Scheduling for Communication-Computation Overlap in Large Model Training via Communication Partitioning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '24)*. 178–191.

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021).

[10] Daixuan Cheng, Shaohan Huang, and Furu Wei. 2024. Adapting Large Language Models via Reading Comprehension. In *Proceedings of the Twelfth International Conference on Learning Representations*.

[11] Sanghun Cho, Hyojun Son, and John Kim. 2023. Logical/Physical Topology-Aware Collective Communication in Deep Learning Training. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA '23)*. IEEE, 56–68.

[12] OpenAI Community. 2019. GPT2 Medium. https://huggingface.co/gpt2-medium/resolve/main/config.json. [Online; Accessed 21-Nov-2024].

[13] DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. *arXiv preprint arXiv:2412.19437* (2024).

[14] 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.

[15] Jianbo Dong, Zheng Cao, Tao Zhang, Jianxi Ye, Shaochuang Wang, Fei Feng, Li Zhao, Xiaoyong Liu, Liuyihan Song, Liwei Peng, Yiqun Guo, Xiaowei Jiang, Lingbo Tang, Yin Du, Yingya Zhang, Pan Pan, and Yuan Xie. 2020. EFLOPS: Algorithm and System Co-Design for A High Performance Distributed Training Platform. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA '20)*. IEEE, 610–622.

[16] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. 2022. 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 (PPoPP '22)*. 120–134.

[17] Jiayi Huang, Pritam Majumder, Sungkeun Kim, Abdullah Muzahid, Ki Hwan Yum, and Eun Jung Kim. 2021. Communication Algorithm-Architecture Co-Design for Distributed Deep Learning. In *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA '21)*. IEEE, 181–194.

[18] Zongle Huang, Shupei Fan, Chen Tang, Xinyuan Lin, Shuwen Deng, and Yongpan Liu. 2024. Hecaton: Training and Finetuning Large Language Models with Scalable Chiplet Systems. *arXiv preprint arXiv:2407.05784* (2024).

[19] Drago Ignjatović, Daniel W Bailey, and Ljubisa Bajić. 2022. The Wormhole AI Training Processor. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 356–358.

[20] Alex Ishii, Denis Foley, Eric Anderson, Bill Dally, Glenn Dearth, Larry Dennison, Mark Hummel, and John Schafer. 2018. NVSwitch and DGX-2: NVLink-Switching Chip and Scale-Up Compute Server. In *Proceedings of the Hot Chips 30 Symposium (HCS)*.

[21] Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. 2021. Data Movement Is All You Need: A Case Study on Optimizing Transformers. In *Proceedings of Machine Learning and Systems*, Vol. 3. 711–732.

[22] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. 2024. System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing (PODC '24)*. 121–130.

[23] Nikhil Jain and Yogish Sabharwal. 2010. Optimal Bucket Algorithms for Large MPI Collectives on Torus Interconnects. In *Proceedings of the 24th ACM International Conference on Supercomputing (ICS '10)*. 27–36.

[24] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. 2022. Breaking the Computation and Communication Abstraction Barrier in Distributed Machine Learning Workloads. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*. 402–416.

[25] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[26] Chenyu Jiang, Ye Tian, Zhen Jia, Chuan Wu, Yida Wang, and Shuai Zheng. 2024. Lancet: Accelerating Mixture-of-Experts Training by Overlapping Weight Gradient Computation and All-to-All Communication. In *Proceedings of Machine Learning and Systems*, Vol. 6. 74–86.

[27] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A Survey on Large Language Models for Code Generation. *arXiv preprint arXiv:2406.00515* (2024).

[28] Nan Jiang, Daniel U Becker, George Michelogiannakis, James Balfour, Brian Towles, David E Shaw, John Kim, and William J Dally. 2013. A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator. In *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 86–96.

[29] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. 2024. MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI'24)*. 745–760.

[30] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. 2023. TPU v4: An Optically

Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23)*. 1–14.

[31] Norman P Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. 2020. A Domain-Specific Supercomputer for Training Deep Neural Networks. *Commun. ACM* 63, 7 (2020), 67–78.

[32] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing Activation Recomputation in Large Transformer Models. In *Proceedings of Machine Learning and Systems*, Vol. 5. 341–353.

[33] Sameer Kumar and Norm Jouppi. 2020. Highly Available Data Parallel ML Training on Mesh Networks. *arXiv preprint arXiv:2011.03605* (2020).

[34] Sabuj Laskar, Pranati Majhi, Sungkeun Kim, Farabi Mahmud, Abdullah Muzahid, and Eun Jung Kim. 2024. Enhancing Collective Communication in MCM Accelerators for Deep Learning Training. In *Proceedings of the 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1–16.

[35] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. In *Proceedings of the International Conference on Learning Representations*.

[36] Dacheng Li, Rulin Shao, Anze Xie, Eric P Xing, Joseph E Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023. LightSeq: Sequence Level Parallelism for Distributed Training of Long Context Transformers. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@NeurIPS 2023)*.

[37] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. 2023. Sequence Parallelism: Long Sequence Training from System Perspective. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2391–2404.

[38] Heng Liao, Bingyang Liu, Xianping Chen, Zhigang Guo, Chuanning Cheng, Jianbing Wang, Xiangyu Chen, Peng Dong, Rui Meng, Wenjie Liu, Zhe Zhou, Ziyang Zhang, Yuhang Gai, Cunle Qian, Yi Xiong, Zhongwu Cheng, Jing Xia, Yuli Ma, Xi Chen, Wenhua Du, Shizhong Xiao, Chungang Li, Yong Qin, Liudong Xiong, Zhou Yu, Lv Chen, Lei Chen, Buyun Wang, Pei Wu, Junen Gao, Xiaochu Li, Jian He, Shizhuan Yan, and Bill McColl. 2025. UB-Mesh: a Hierarchically Localized nD-FullMesh Datacenter Network Architecture. *arXiv preprint arXiv:2503.20377* (2025).

[39] Sean Lie. 2022. Cerebras Architecture Deep Dive: First Look Inside the HW/SW Co-Design for Deep Learning: Cerebras Systems. In *Proceedings of the IEEE Hot Chips 34 Symposium (HCS)*. 1–34.

[40] Hao Liu, Matei Zaharia, and Pieter Abbeel. 2024. Ring Attention with Blockwise Transformers for Near-Infinite Context. In *Proceedings of the Twelfth International Conference on Learning Representations*.

[41] David Luebke. 2008. CUDA: Scalable Parallel Programming for High-Performance Scientific Computing. In *Proceedings of the 2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. IEEE, 836–838.

[42] Microsoft. 2023. DeepSpeed. https://github.com/microsoft/DeepSpeed/tree/master/deepspeed/moe. [Online; Accessed 21-Nov-2024].

[43] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*. 1–15.

[44] NVIDIA. 2020. NVIDIA A100 Tensor Core GPU Architecture. *Volume 1.0: Whitepaper, Part* 1, 2020 (2020), 82.

[45] NVIDIA. 2020. NVIDIA Collective Communication Library (NCCL) Documentation. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/index.html. [Online; Accessed 22-Feb-2025].

[46] NVIDIA. 2023. Megatron-LM. https://github.com/NVIDIA/Megatron-LM. [Online; Accessed 22-Feb-2025].

[47] NVIDIA. 2024. NVIDIA DGX Platform. https://www.nvidia.com/en-us/data-center/dgx-platform/. [Online; Accessed 29-Jul-2024].

[48] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *Proceedings of the 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 118–132.

[49] Raghu Prabhakar and Sumti Jairath. 2021. SambaNova SN10 RDU: Accelerating Software 2.0 with Dataflow. In *Proceedings of the IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 1–37.

[50] Kishore Punniyamurthy, Khaled Hamidouche, and Bradford M Beckmann. 2024. Optimizing Distributed ML Communication with Fused Computation-Collective Operations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '24)*. 1–17.

[51] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025. Mooncake: Trading More Storage for Less Computation —A KVCache-centric Architecture for Serving LLM Chatbot. In *Proceedings of the 23rd USENIX Conference on File and Storage Technologies (FAST '25)*. 155–170.

[52] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models Are Unsupervised Multitask Learners. *OpenAI Blog* 1, 8 (2019).

[53] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. In *Proceedings of the 39th International Conference on Machine Learning*. PMLR, 18332–18346.

[54] Saeed Rashidi, Matthew Denton, Srinivas Sridharan, Sudarshan Srinivasan, Amoghavarsha Suresh, Jade Nie, and Tushar Krishna. 2021. Enabling Compute-Communication Overlap in Distributed Deep Learning Training Platforms. In *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA '21)*. IEEE, 540–553.

[55] Saeed Rashidi, William Won, Sudarshan Srinivasan, Srinivas Sridharan, and Tushar Krishna. 2022. Themis: A Network Bandwidth-Aware Collective Scheduling Policy for Distributed Training of DL Models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA '22)*. 581–596.

[56] 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 (KDD '20)*. 3505–3506.

[57] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A Systematic Methodology for Characterizing Scalability of DNN Accelerators Using SCALE-Sim. In *Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 58–68.

[58] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN Accelerator Simulator. *arXiv preprint arXiv:1811.02883* (2018).

[59] Robert R Schaller. 1997. Moore's Law: Past, Present, and Future. *IEEE Spectrum* 34, 6 (1997), 52–59.

[60] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23)*. 593–612.

[61] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. 2024. S-LoRA: Serving Thousands of Concurrent LoRA Adapters. In *Proceedings of Machine Learning and Systems*, Vol. 6. 296–311.

[62] 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).

[63] Siddharth Singh, Olatunji Ruwase, Ammar Ahmad Awan, Samyam Rajbhandari, Yuxiong He, and Abhinav Bhatele. 2023. A Hybrid Tensor-Expert-Data Parallelism Approach to Optimize Mixture-of-Experts Training. In *Proceedings of the 37th International Conference on Supercomputing (ICS '23)*. 203–214.

[64] Emil Talpes, Douglas Williams, and Debjit Das Sarma. 2022. Dojo: The Microarchitecture of Tesla's Exa-Scale Computer. In *Proceedings of the 2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 1–28.

[65] HPC-AI Tech. 2023. ColossalAI: Making Large AI Models Cheaper, Faster, and More Accessible. https://github.com/hpcaitech/ColossalAI. [Online; Accessed 22-Feb-2025].

[66] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *The International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

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

[68] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. Experience: Evaluating The Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems (CHI EA '22)*. 1–7.

[69] Guanhua Wang, Chengming Zhang, Zheyu Shen, Ang Li, and Olatunji Ruwase. 2024. Domino: Eliminating Communication in LLM Training via Generic Tensor Slicing and Overlapping. *arXiv preprint arXiv:2409.15241* (2024).

[70] Shibo Wang, Jinliang Wei, Amit Sabne, Andy Davis, Berkin Ilbeyi, Blake Hechtman, Dehao Chen, Karthik Srinivasa Murthy, Marcello Maggioni, Qiao Zhang, Sameer Kumar, Tongfei Guo, Yuanzhong Xu, and Zongwei Zhou. 2023. Overlap Communication with Dependent Computation via Decomposition in Large Deep Learning Models. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '23)*. 93–106.

[71] Lilian Weng. 2021. How to Train Really Large Models on Many GPUs? *Lil'Log* (Sep 2021). https://lilianweng.github.io/posts/2021-09-25-train-large/
[72] Lilian Weng. 2023. Large Transformer Model Inference Optimization. *Lil'Log* (Jan 2023). https://lilianweng.github.io/posts/2023-01-10-inference-optimization/
[73] William Won, Midhilesh Elavazhagan, Sudarshan Srinivasan, Swati Gupta, and Tushar Krishna. 2024. TACOS: Topology-Aware Collective Algorithm Synthesizer for Distributed Machine Learning. In *Proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture (MICRO '24)*. IEEE, 856–870.
[74] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. 2018. Image Classification at Supercomputer Scale. *arXiv preprint arXiv:1811.06992* (2018).
[75] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter-and Intra-Operator Parallelism for Distributed Deep Learning. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*. 559–578.
[76] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-Optimized Large Language Model Serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI '24)*. 193–210.

A Artifact Appendix

A.1 Abstract

The artifact contains the codes for Chimera, along with its setup and running descriptions. We provide instructions and click-to-run scripts for reproducing the main results presented in our paper. Specifically, we reproduce the results for Fig. 10, Fig. 12-14.

A.2 Artifact check-list (meta-information)

- **Program:** Python = 3.9, ScaleSim-v2, BookSim2
- **Run-time environment:** Ubuntu = 22.04
- **Experiments:** We include the scripts for running simulations and real machine tests.
- **Metrics:** We evaluate the communication bandwidth, end-to-end latency, and real-machine communication time in our evaluation.
- **Output:** The outputs of the artifact are figures in PDF format that reproduce the main results of our paper.
- **How much disk space required (approximately)?:** The disk space should be around 2 TB.
- **How much time is needed to prepare workflow (approximately)?:** Around 30 minutes.
- **How much time is needed to complete experiments (approximately)?:** The simulation time varies among different communication sizes and models. The shortest time spent on some small communication size is a few seconds, while the longest time spent on others can be up to 6 hours. Several days might take for all experiments. Parallel simulations are recommended. Real-machine experiments take 8 hours approximately.
- **Publicly available?:** Yes.

A.3 Description

A.3.1 How to access. The artifact is archived in Zenodo[4]. It can also be accessed from GitHub, as the command shown below:

```
$ git clone https://github.com/redbird-arch/
    isca2025-chimera-artifact.git
```

A.3.2 Hardware dependencies. For reference, we list our system configurations here:

For simulation experiments:

- OS: Ubuntu 22.04.5 LTS
- CPU: Intel 13th Generation Intel Core i9 Processors @ 3.00GHz (24 cores); Other CPU would work.
- DRAM: 64 GB
- Disk: 2 TB

For real machine experiments:

- OS: Ubuntu 22.04.5 LTS
- CPU: Intel(R) Xeon(R) Platinum 8352V Processor @2.10GHz (36 cores)
- GPU: 8×NVIDIA RTX 4080 GPUs, 8×32 GB
- Disk: 2 TB

A.3.3 Software dependencies. We ran our experiments on the Ubuntu 22.04 LTS operating system, but other versions of Ubuntu should also work. A Python runtime environment constitutes the fundamental requirement for operation for simulation experiments. For real-machine tests, the dependencies are PyTorch = 2.5.1, CUDA = 12.4, NCCL = 2.21.5. Complete dependency specifications are documented in the README.md file.

A.4 Installation

We provide three scripts: the first to build the anaconda environment, the second to activate the environment, and the last to install other required libraries. The commands are shown below:

```
# Update directory paths for scripts
$ python update_cfg.py

# Create anaconda environment
$ conda create --name Chimera python=3.9

# Enter the anaconda environment
$ conda activate Chimera

# Solve other dependencies
$ pip install -r requirements.txt
$ conda install -c conda-forge flex bison
```

A.5 Experiment workflow

We provide the scripts to compile BookSim2 simulator. The commands are shown below:

```
$ cd ./src/communication/backend/booksim2/
    src

# Compilation
$ make libdbg
$ cd ../../../../../
```

We also provide a single script for clicking to run all the simulation experiments. The command is shown below:

```
$ cd ./src/User/Chimera/experiment

# Run all the simulations
$ bash run-all.sh
```

We also provide a single script for clicking to run all the real machine experiments. The commands are shown below:

[4] https://doi.org/10.5281/zenodo.15348400

```
$ cd ./Real_Perf && bash run-all.sh
```

We provide the figure scripts. The commands are shown below:

```
$ cd ./src/User/Chimera/experiment/Pictures

# Draw the figures
$ conda activate Chimera
$ bash plot-figure.sh
```

A.6 Evaluation and expected results

The results and figures can be found in the directories `./src/User/Chimera/experiment/ISCA25_*` and `./src/User/Chimera/experiment/Pictures`, respectively.

A.7 Notes

The `README.md` file provides additional information on the organization of the code and detailed steps for running experiments.