

# FSMoE: A Flexible and Scalable Training System for Sparse Mixture-of-Experts Models

Xinglin Pan\*

The Hong Kong University of Science  
and Technology (Guangzhou)  
Guangzhou, China  
xpan413@connect.hkust-gz.edu.cn

Shaohuai Shi

Harbin Institute of Technology,  
Shenzhen  
Shenzhen, China  
shaohuais@hit.edu.cn

Bo Li

Hong Kong University of Science and  
Technology  
Hong Kong SAR, China  
bli@cse.ust.hk

Wenxiang Lin\*

Harbin Institute of Technology,  
Shenzhen  
Shenzhen, China  
wenxianglin@stu.hit.edu.cn

Zhenheng Tang

The Hong Kong University of Science  
and Technology  
Hong Kong SAR, China  
zhtang.ml@ust.hk

Lin Zhang

Hong Kong University of Science and  
Technology  
Hong Kong SAR, China  
lhangbv@connect.ust.hk

Rui Wang

The Hong Kong University of Science  
and Technology (Guangzhou)  
Guangzhou, China  
rwang132@connect.hkust-gz.edu.cn

Xiaowen Chu†

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

## Abstract

Recent large language models (LLMs) have tended to leverage sparsity to reduce computations, employing the sparsely activated mixture-of-experts (MoE) technique. MoE introduces four modules, including token routing, token communication, expert computation, and expert parallelism, that impact model quality and training efficiency. To enable versatile usage of MoE models, we introduce **FSMoE**, a flexible training system optimizing task scheduling with three novel techniques: 1) Unified abstraction and online profiling of MoE modules for task scheduling across various MoE implementations. 2) Co-scheduling intra-node and inter-node communications with computations to minimize communication overheads. 3) To support near-optimal task scheduling, we design an adaptive gradient partitioning method for gradient aggregation and a schedule to adaptively pipeline

communications and computations. We conduct extensive experiments with configured MoE layers and real-world MoE models on two GPU clusters. Experimental results show that 1) our FSMoE supports four popular types of MoE routing functions and is more efficient than existing implementations (with up to a 1.42 $\times$  speedup), and 2) FSMoE outperforms the state-of-the-art MoE training systems (DeepSpeed-MoE and Tutel) by 1.18 $\times$ -1.22 $\times$  on 1458 MoE layers and 1.19 $\times$ -3.01 $\times$  on real-world MoE models based on GPT-2 and Mixtral using a popular routing function.

**CCS Concepts:** • Computing methodologies → Parallel algorithms; Machine learning.

**Keywords:** Distributed Deep Learning; Large Language Model; Mixture-of-Experts; Training System; Scheduling

## ACM Reference Format:

Xinglin Pan, Wenxiang Lin, Lin Zhang, Shaohuai Shi, Zhenheng Tang, Rui Wang, Bo Li, and Xiaowen Chu. 2025. FSMoE: A Flexible and Scalable Training System for Sparse Mixture-of-Experts Models . In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands*. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3669940.3707272>

\*Equal contribution.

†Also with Hong Kong University of Science and Technology, Hong Kong SAR.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org. *ASPLOS '25, March 30-April 3, 2025, Rotterdam, Netherlands*

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0698-1/25/03...\$15.00

<https://doi.org/10.1145/3669940.3707272>

## 1 Introduction

In recent years, the concept of sparsely-activated Mixture-of-Experts (MoE) layers has garnered considerable attention [7, 10, 11, 20, 22, 40] in large language models (LLMs) as MoE can scale up the model size while keeping the computational cost for training be sub-linearly increased. MoE models incorporate multiple experts in MoE layers, and each

expert represents a specialized feed-forward network (ffn) trained for specific subtasks (with particular input tokens). These MoE layers utilize a gating mechanism using a routing function (e.g., employing softmax activation [22]) to dynamically assign data samples to the appropriate experts. This approach allows for scaling up model sizes. For instance, the Switch Transformer [12] scales up to 1.5 trillion parameters with 15 MoE layers, each consisting of 2048 experts, surpassing its dense model (w/o MoE) that has only several billion parameters. The MoE technique has significantly improved model performance in domains such as natural language processing [51, 52], computer vision [16], speech recognition [48], and recommendation systems [27].

Though the MoE technique has achieved remarkable success in many AI models, it is still actively studied in both algorithms and systems. From the algorithm's point of view, how to train the experts effectively and efficiently with what input tokens is still an open problem, which means the routing function plays an important role in the model generalization capability [6, 12, 22, 23, 36, 40, 51]. From the system's point of view, dedicated MoE systems (e.g., FastMoE [13], DeepSpeed-MoE [39], FlexMoE [30], Tutel [17], and ScheMoE [43]) are designed to support the study of algorithms and deployment of MoE models. Atop of these systems, there exist many optimization strategies to improve MoE training efficiency by designing efficient communication collectives (e.g., AlltoAll algorithms [2, 17, 28, 39]), optimizing sparsity computation [50], and scheduling communication and computation tasks [5, 14, 17, 21, 24, 25, 33, 42, 43, 49] to enable the overlap of communication tasks with computing tasks. However, these systems have two main limitations: 1) they only support limited routing functions and are inflexible to support newly introduced routing mechanisms, and 2) they are optimized with particular parallelisms and are sub-optimal in some commonly used scenarios.

As the scale of LLM continues to expand, various parallel paradigms have emerged to address the escalating computational demands. In addition to the three well-established parallel paradigms for traditional models, namely Data Parallelism (DP) [9], Model Parallelism (MP) [9], and Pipeline Parallelism (PP) [15], MoE models introduce two additional parallel paradigms: Expert Parallelism (EP) [40] and Expert-Sharding Parallelism (ESP) [39] to enable large experts to be trained on GPU clusters (§2.2). These parallelisms significantly impact the system scalability by the communication time incurred by the various parallel paradigms. Research studies such as [17, 22, 24, 26] have reported that the All-toAll communication time comprises a substantial portion, ranging from 30% to 60%, of the overall time required for executing the MoE layers on high-end GPU or TPU clusters. This issue becomes more severe when DP, MP, EP, and ESP (DP+MP+EP+ESP) are employed simultaneously for large-scale training [17]. Consequently, rapid changes in MoE routing mechanisms and complicated parallelisms

present a significant modularisation difficulty to flexibly support newly designed MoE components and a performance challenge to optimally schedule different time-consuming tasks.

To this end, in this work, we propose FSMoE<sup>1</sup>, a flexible and efficient MoE system with near-optimal task scheduling, to efficiently train MoE models. First, to enhance extensibility, we design unified abstraction and online profiling of MoE modules for task scheduling across various MoE implementations. Second, to better schedule the communication and computing tasks in DP+MP+EP+ESP, we analyse the possibility of overlapping the inter-node and intra-node communications and propose an efficient schedule to pipeline inter-node and intra-node communication tasks as well as computation tasks in a common scenario where the MP and ESP group is configured to align with the number of GPUs within a node. Third, to support near-optimal task scheduling, we design a gradient partitioning method for fully overlapping the gradient aggregation with other tasks. Our main technical contributions are summarized as follows.

- We modularize all possible operations in an MoE layer to support various MoE components, including gating function, data layout, collective communication, expert computation, etc. We have provided the implementations of four popular types of gating functions in our system, which are more efficient than their original implementations.
- Based on the modularized operations, we propose an adaptive optimal scheduling algorithm to pipeline both intra-node and inter-node communication tasks as well as computing tasks to improve the training efficiency in common scenarios where the group of MP and ESP matches the number of GPUs per node.
- We design an adaptive gradient partitioning method to hide the communication cost of the gradient aggregation by pipelining communications with computations and avoid the contention between different inter-node communications.
- We conduct extensive experiments on a 48-GPU cluster and a 32-GPU cluster using customized MoE layers and real-world MoE models. Experimental results show that: (1) FSMoE outperforms Tutel [17] (with its improved algorithm PipeMoE [42]) by 1.18× to 1.22× on training 1458 different configured cases. (2) FSMoE runs 1.19× to 3.01× faster on average than the state-of-the-art MoE systems (Tutel and DeepSpeed-MoE [39]) on training two real-world MoE models based on GPT-2 and Mixtral.

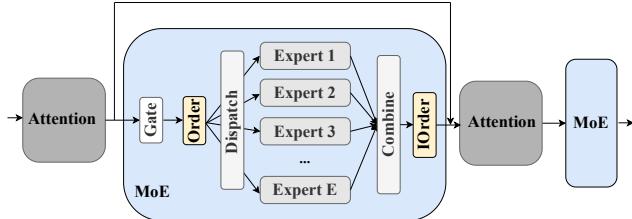
## 2 Background and Motivations

For ease of presentation, we provide a summary of the essential notations employed in the paper, presented in Table 1.

<sup>1</sup>Code Repository: <https://github.com/xpan413/FSMoE>.

**Table 1.** Notations.

Name	Description
$P$	# of GPUs
$r$	# of the pipeline degree
$B$	# of samples per GPU (or local mini-batch size)
$L$	# of tokens per sample (or sequence length)
$E$	total number of experts
$k$	top- $k$ experts should be selected for each token
$f$	factor to control expert's maximum token count
$M$	embedding size of a token
$H$	hidden size of the feed-forward layer in experts
$N_{head}$	# of heads in the attention layer
$N_{DP}$	# of workers in each DP group
$N_{MP}$	# of workers in each MP group
$N_{EP}$	# of workers in each EP group
$N_{ESP}$	# of workers in each ESP group
$N_{PP}$	# of workers in each PP group

**Figure 1.** A typical MoE structure with  $E$  experts.

## 2.1 Mixture-of-Experts Layer

In modern MoE models, which are typically built atop the Transformer [45] architecture, an MoE layer is used to replace the ffn layer. As shown in Fig. 1, the MoE layer comprises three core components: a gating function, an ordering function (and its reverse operation, i.e., the I-ordering function) and a set of  $E$  experts.

**Gating Function.** The gating function plays a pivotal role in assigning tokens to specific experts. During each training iteration, the input data (denoted as  $I$ ) of the MoE layer has a shape of  $(B, L, M)$ , where  $B$  represents the mini-batch size,  $L$  represents the sequence length per sample, and  $M$  represents the embedding size. To determine the activation of experts,  $I$  is divided into multiple parts based on the gating function.

GShard [22] employs a noisy Top-k Gate, denoted as  $G(I) = \text{Softmax}(\text{KeepTopK}(H(I), k))$ , where  $H(I)$  adds noises to the input  $I$  through a specific transformation:

$$H(I)_i = (I \cdot W_g)_i + \mathcal{N}(0, 1) \cdot \text{Softplus}((I \cdot W_{\text{noise}})_i),$$

and the function  $\text{KeepTopK}(v, k)$  retains the top  $k$  values of a vector  $v$ , setting the rest to negative infinity:

$$\text{KeepTopK}(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ values of } v. \\ -\infty & \text{otherwise.} \end{cases}$$

In  $\text{KeepTopK}(v, k)$ ,  $W_g$  and  $W_{\text{noise}}$  are two trainable weights.

In BASE [23] and StableMoE [49] models, the sigmoid gate is employed, defined by  $H(I)_i = (I \cdot W_g)_i$ . The output from the expert is scaled by  $\sigma(H(I)_i)$ . If this output contributes positively to  $I$ , optimizing the training goal (such as minimizing cross-entropy loss in language modelling) increases the gate value, favouring the selection of the same expert. In X-MoE [6], a low-rank linear projection  $W_{\text{proj}}I$  is employed to segregate the direct interaction between the hidden vector  $I$  and the expert embedding  $W_g$ . This approach effectively mitigates the issue of cascaded collapse in representations. Subsequently, these representations undergo an l2 normalization process to be scaled appropriately. The formula can be expressed as follows:  $s_i = \cos(W_{\text{proj}}I, W_g)$ . An expert choice method [51] independently selects top- $k$  tokens for each expert, denoted as  $G(I) = \text{Softmax}(\text{KeepTopK}((I \cdot W_g)^T, k))$ .

The effectiveness of gating functions is assessed using specific models and datasets. For example, EC [51] is evaluated through casual language modelling tasks, whereas X-MoE [6] is assessed via masked language modelling tasks. When encountering new challenges, developers cannot determine the most suitable gating functions for the task without conducting practical tests. Therefore, incorporating a diverse range of gating functions enhances the robustness for developers.

**Ordering and I-Ordering Functions.** The ordering function transforms the input tensor layout before dispatched. Typically, the format changes from  $(B, L, M)$  to  $(E, T, M)$ , where  $T$  denotes the maximum tokens per expert.  $T$  is determined using the formula  $T := k \times f \times B \times L/E$ , where  $f$  is a control factor. Each row of  $G$  (i.e.,  $G[i, :, :]$ ) aligns with the data for the  $i$ -th expert ( $i$  ranges from 1 to  $E$ ). There are two main types of ordering functions: 1) GShard [22] ordering, which uses a combination of einsum and matrix multiplication, and 2) Tutel [17] ordering, which employs SIMD-efficient sparse operations. The I-ordering function serves as a reverse function of the ordering function, allowing for the data layout to be adjusted back to its original form.

**Experts.** Typically, each expert in the MoE layer is a compact neural network consisting of several feed-forward layers followed by an activation function [20, 22]. Take a two-layer expert as an example, the first layer has a weight matrix with a shape of  $(M, H)$ , while the second layer has a shape of  $(H, M)$ , where  $H$  represents the size of the hidden layer so that the output of expert has the shape with the input. For an MoE layer with  $E$  experts, we denote the  $i$ -th expert as  $e_i$ , and its output can be denoted as  $Q_i = e_i(G[i, :, :])$ . To obtain the final output of the MoE layer, the outputs of all experts are combined into a unified tensor  $[Q_1, Q_2, \dots, Q_E]$ . This tensor can be reshaped as  $(B, L, M)$  to facilitate further processing in subsequent layers.

Despite the expansion in the model size in MoE models, the increase in their computational cost is marginal. However, the size of these models has grown to such an extent

that they cannot be loaded into the memory of a single device. As a result, distributed training becomes essential for training MoE models, leveraging multiple devices to handle the computational and memory demands, which easily introduces significant communication overheads. A benchmark of the training time breakdown with two popular MoE models is conducted on our 32-GPU and 48-GPU testbeds (details in §6) is shown in Table 2. It demonstrates that communication overhead typically contributes over 50% to the overall training, indicating the necessity of optimizing communication performance.

## 2.2 Paradigms of Parallelism

The hybrid parallelism with DP, MP, EP, and ESP is required to train large-scale MoE models on a GPU cluster.

**Data Parallelism.** In distributed DL, the data parallelism (DP) training technique has become a de-facto method [9, 19, 47], where a mini-batch of samples is distributed to the workers in the DP group. During backpropagation, the gradients of each worker in the same DP group are aggregated through an AllReduce operation (we call Gradient-AllReduce afterwards) so that they can use the identical gradient to update model parameters.

**Model Parallelism.** Model Parallelism (MP) [9, 29] is a technique that divides model parameters among multiple workers to facilitate parallel computation. Each worker performs its computations independently, and subsequently, the outputs from all workers are combined through an AllReduce collective operation. Notably, when the MP group is configured as the number of GPUs within a node, which is very common, the communication involved in MP is considered intra-node communication, while the collective communication for gradient aggregation involves inter-node communication.

**Expert Parallelism.** In Expert Parallelism (EP) [22, 40], experts are assigned to different GPUs, ensuring that each device handles a specific subset of experts. After the data passes through the gating function, the rows of the tensor  $G$  ( $G[i, :, :]$ ) on each device correspond to the data assigned to the respective  $i$ -th expert ( $i = 1, 2, \dots, E$ ). As the experts are distributed across multiple devices, the dispatch operation uses a collective communication technique called AlltoAll Dispatch. This approach facilitates sending tokens to their respective experts for computation. Subsequently, the outputs generated by all experts are combined using another AlltoAll operation, known as AlltoAll Combine, for further processing.

**Expert-Sharding Parallelism.** When training large-scale MoE models, the number of workers  $P$  may exceed the number of experts  $E$ . In such cases, expert-sharding parallelism (ESP) [17, 39, 44] can be employed to distribute the workload evenly across all workers. ESP groups are formed to uniformly partition the experts among the GPUs within each

group, similar to MP. This enables parallel computation of expert outputs across all workers within the ESP group.

The combination of EP and ESP is required to place each MoE layer across multiple GPUs, which introduces additional communication operators [33, 44], namely ESP-AllGather and ESP-ReduceScatter. ESP-AllGather ensures that the input data is uniformly distributed among all workers within the ESP group, while ESP-ReduceScatter is used to aggregate the outputs of expert shards within the ESP group and split them back into the original structure of the input. The number of GPUs in an ESP group is denoted as  $N_{\text{ESP}}$ . Notably, when the ESP group is configured to align with GPUs within a node, the ESP-AllGather and ESP-ReduceScatter operations involve intra-node communication while the AlltoAll operation introduced by EP entails inter-node communication, enabling the overlaps between ESP-AllGather/ESP-ReduceScatter and AlltoAll. In this work, we mainly discuss the schedule under this case.

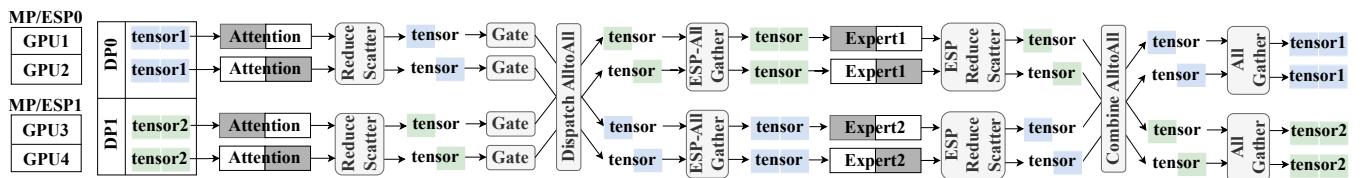
An example of training an MoE model [44] with DP, MP, EP, and ESP is shown in Fig. 2, where  $N_{\text{DP}} = N_{\text{MP}} = N_{\text{EP}} = N_{\text{ESP}} = 2$ . In this example, two different tensors (or two mini-batches of samples) from the *DP group* go through the attention layer partitioned across *two MP groups* and are divided into half by using a ReduceScatter operation introduced by MP. Then two split tensors find selected experts partitioned across *two ESP groups* by the gating function and are dispatched into the corresponding devices across *two EP groups* (GPU1 and GPU3; GPU2 and GPU4) through an AlltoAll operation. Before the expert computation, split tensors should be combined through an AllGather collective across the two ESP groups called ESP-AllGather. Then, after the experts computation, tensors are divided into half again by another ReduceScatter operation introduced by ESP, which is called ESP-ReduceScatter, and they are sent back to their original workers through another AlltoAll operation. Finally, another AllGather operation is performed for these tensors across the MP groups to finalize the output. It is seen that it requires several key components and complicated parallelisms to train MoE models, which motivates our designed system to provide a flexible and scalable MoE training system.

## 2.3 Motivations

**A Flexible MoE framework.** A flexible MoE framework should efficiently combine different routing functions [6, 12, 22, 23, 36, 40, 51], order functions [14, 17], expert blocks [3, 20], and AlltoAll algorithms [2, 17, 28, 39]. This integration should be achieved with minimal complex programming for additional customization. The aim is to comprehensively address all types of overlaps, like communication with communication or computing, particularly when dealing with diverse parallel groups like integrating DP, MP, EP, and ESP (§3).

**Table 2.** Time performance (iteration time in millisecond) of each operation in a transformer layer of two real-world models, GPT2-XL [38] and Mixtral7B [20], with  $B = 4$  and  $L = 1024$  for two testbeds in Table 3. The numbers in the brackets represent each operation's portion of the forward and backward time.

Testbeds/Breakdown	Communication				Computation			
	AlltoAll	AllReduce	AllGather	ReduceScatter	Experts	Routing	Order	Attention
A	GPT2-Forward	6.9(31.16%)	0(0%)	4.6(20.83%)	5.4(24.46%)	3.1(14.04%)	0.1(0.45%)	0.3(1.36%)
	GPT2-Backward	6.9(21.27%)	5.26(16.26%)	4.6(14.22%)	5.4(16.7%)	6.1(18.86%)	0.1(0.31%)	0.4(1.24%)
	Mixtral-Forward	19.5(29.8%)	0(0%)	12.3(18.73%)	13.7(20.86%)	15.6(23.76%)	0.1(0.15%)	0.3(0.46%)
	Mixtral-Backward	19.6(17.45%)	26.45(23.59%)	12.3(10.97%)	13.7(12.22%)	31.8(28.36%)	0.1(0.09%)	0.5(0.45%)
B	GPT2-Forward	11.2(20.7%)	0(0%)	15.5(28.7%)	15.7(29.1%)	6.7(12.4%)	0.1(0.2%)	0.3(0.6%)
	GPT2-Backward	11.2(15.7%)	7.3(10.3%)	15.5(21.8%)	15.2(21.3%)	13(18.3%)	0.1(0.1%)	0.3(0.4%)
	Mixtral-Forward	28.3(15.9%)	0.0(0.0%)	39.6(22.3%)	40.8(23.0%)	58.5(33.0%)	0.1(0.1%)	0.7(0.4%)
	Mixtral-Backward	30.8(10.8%)	32.1(11.3%)	40.1(14.1%)	41.8(14.7%)	119.7(42.1%)	0.2(0.1%)	1.2(0.4%)
								18.1(6.4%)



**Figure 2.** An example of  $N_{DP} = N_{MP} = N_{EP} = N_{ESP} = 2$ . The attention is partitioned into two parts across MP groups, and the two experts are distributed to the two EP groups (GPU1 and GPU3, as well as GPU2 and GPU4) in EP, and each expert is further partitioned into two shards across the ESP group. The blue and green rectangles indicate the data tensors.

**Optimizing Network Communication.** As shown in Fig. 3a, various parallel paradigms (e.g., DP, MP, EP, ESP) comprise a substantial portion of the overall iteration time. To mitigate the communication cost associated with the MoE layer, prior research (e.g., Tutel [17], PipeMoE [42], Faster-MoE [14]) has explored overlapping AlltoAll with experts as illustrated in Fig. 3b. However, they do not explore the overlapping ESP-AllGather/ESP-ReduceScatter (intra-node communication) with AlltoAll Dispatch/Combine (inter-node communication), diminishing network efficiency. This motivates us to pipeline inter-node and intra-node communication as shown in Fig. 3c (§4).

**Optimizing Forward and Backward Separately.** Existing systems (e.g., Tutel [17] and DeepSpeed-MoE [39]) typically use the same pipeline degree (i.e., the number of split input chunks for the overlaps) for both forward and backward propagation during training. However, the ideal degree may vary between these two phases due to their distinct computational requirements. For example, backward propagation involves additional computations to calculate the gradient of weights. Our extensive experiments on 1,458 MoE configurations (details in Table 4) reveal that 912 cases exhibit varied optimal pipeline degrees, tested on a 32-GPU cluster with 8 nodes (details in Table 3). Therefore, adaptively determining the pipeline degrees for both forward and backward phases is needed to achieve better training efficiency (§4.4).

**Co-Design in Backward Propagation and Gradient Synchronization.** Since Gradient-AllReduce (introduced

by the weight synchronization in DP) and AlltoAll are both inter-node communication, Gradient-AllReduce can not be directly overlapped with the whole MoE layer as shown in Fig. 3b and Fig. 3c which only overlap Gradient-AllReduce with non-MoE parts. Consequently, designing overlaps for Gradient-AllReduce without considering MoE layers tends to result in sub-optimal solutions. A co-design that considers the AlltoAll operation and adjusts the partitioning of gradients for optimal overlapping remains unexplored (§5).

### 3 FSMoE: System Design

We propose FSMoE, a flexible and scalable MoE framework for distributed training. Our framework has three main characteristics: 1) modularization and non-invasive modification, 2) isolation of front-end API definition and back-end task scheduling, and 3) easy schedule of different tasks.

#### 3.1 Modularization and Non-Invasive Modification

In our FSMoE framework, the MoE layer is divided into six distinct sub-modules, namely: *Gate*, *Order*, *I-Order*, *Dispatch*, *Combine*, *Expert*.

*Gate:* The *Gate* sub-module determines how tokens are assigned to different experts for calculation. We pre-implement four routing functions: GShard routing [22], Sigmoid [8, 23] routing, X-MoE routing [6], and SoftMoE routing [36].

*Order & I-Order:* The *Order* sub-module transforms the input tensor layout before it is dispatched. Typically, the format changes from  $(B, L, M)$  to  $(E, T, M)$ . We pre-implement two

types of ordering functions: 1) GShard [22] ordering, which uses a combination of einsum and matrix multiplication, and 2) Tutel [17] ordering, which employs SIMD-efficient sparse operations. The *I-Order* sub-module serves as a reverse operation of the *Order* sub-module, allowing for the data layout to be adjusted back to its original form.

*Dispatch & Combine:* The *Dispatch* sub-module handles the collective communication for the token-to-expert dispatch. It allows users to customize the collective communication algorithm without impacting our scheduler. To facilitate this customization, we pre-implement the default A2A algorithm provided by NCCL (NCCL-A2A) [1], 1DH-A2A proposed by Hetu [31], 2DH-A2A proposed by Tutel [17] and DeepSpeed-MoE [39]. This customization ensures optimal dispatching based on user-specific needs. The *Combine* sub-module serves as a reverse operation of the *Dispatch* sub-module.

*Expert:* The Expert sub-module manages the computation task. Modules derived from “`torch.nn.Module`” can serve as the expert component. We offer two variants of these networks: the GPT feed-forward network [3] and the Mixtral feed-forward network [20].

*Hooks:* In our framework, we offer a range of hooks for non-intrusive modification, including *BeforeMoeStartHook*, *BeforeDispatchHook*, *AfterDispatchHook*, *BeforeCombineHook*, and *AfterCombineHook*, as well as *BeforeMoeEndHook*. These hooks facilitate various adjustments without requiring invasive changes. For example, in handling multimodal data, *BeforeMoeStartHook* and *BeforeMoeEndHook* can be utilized to reformat inputs to conform to the standard MoE layer configuration. In another scenario, such as communication compression, *BeforeDispatchHook* is used to compress the tensor before dispatch, while *AfterDispatchHook* serves to decompress it afterward, ensuring efficient extension without the need for fundamental code modifications.

### 3.2 Generic Scheduler

The FSMoE framework boasts a versatile scheduling capability for task pipelines, independent of specific API definitions. It includes a profiler for evaluating the time efficiency of various tasks. Utilizing data collection and predictive modeling, it strategically coordinates sub-modules in MoE, achieving higher efficiency. It includes two main parts: *front-end* and *back-end*.

**Front-end.** Developers select or build their MoE layers. A profiling model then evaluates the execution time of API definitions across various input sizes, using machine learning algorithms (e.g., linear regression) to fit performance. This process allows the scheduler to operate without needing detailed knowledge of each sub-module’s implementation.

**Back-end.** The scheduler utilizes the performance models developed for each module. This enables the automatic optimization of task workflows. The back-end, while not delving into detailed programming, recognizes and arranges tasks at

the sub-module level. The execution of these tasks is then managed by the *front-end*.

### 3.3 Implementation

We implement our system, FSMoE, atop PyTorch with its C/C++ and CUDA extension features. For customized algorithms, users can implement their own MoE components by inheriting our abstract interfaces as shown in Listing 1. To use the MoE layer, as shown in Listing 2, one can instantiate a new instance of an MoE layer, which can be used as a normal `nn.Module` in PyTorch.

```

1 from FSMoE import ExpertBase, CallbackBase
2
3 class CustomizedExpert(ExpertBase):
4     def do_experts(self, args):
5         pass
6
7 class CustomizedCallBack(CallbackBase):
8     def before_moe_start_hook(self, args):
9         pass

```

**Listing 1.** Code sample of implementing the abstractions.

```

1 from FSMoE import LinearGate, SimpleOrder,
2 MOELayer
3
4 gate_impl = LinearGate()
5 order_impl = SimpleOrder()
6 moe_module = MOELayer(gate_impl, order_impl, **
7 kwargs)

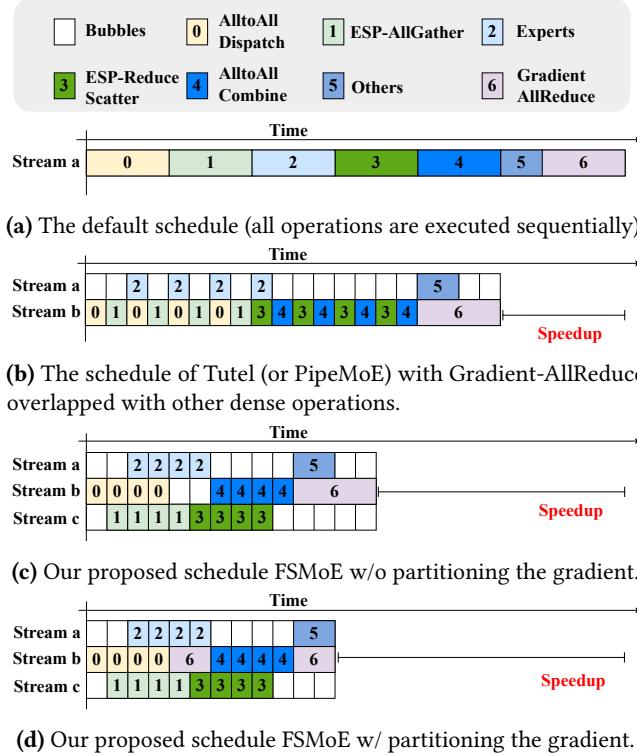
```

**Listing 2.** Code sample of using `FSMoE.moe_module` can be used as a normal `nn.Module` instance in PyTorch.

## 4 Optimized Scheduling of Tasks

Motivated by the potential overlap between inter-node and intra-node communications, we design a new schedule to pipeline all time-consuming communication tasks (ESP-AllGather, ESP-ReduceScatter, AlltoAll Dispatch/Combine, and Gradient-AllReduce communications) and computing tasks (expert and attention computations) when the group of MP and ESP is aligned with the number of GPUs in a node. In such a scenario, ESP-AllGather and ESP-ReduceScatter are intra-node communications, while AlltoAll Dispatch/Combine and Gradient-Allreduce are inter-node communications.

This scenario is frequently encountered in practice. With respect to the MoE framework, each layer comprises a limited number of experts, but each expert’s model is considerably large, preventing it from fitting entirely on a single GPU. For instance, models like Mixtral-8x7B and Qwen1.5-MoE-A2.7B necessitate dividing an expert across multiple GPUs during training. Meanwhile, considering the training hardware system’s topology, inter-node communication (via InfiniBand or Ethernet) generally trails behind the faster intra-node communication methods (such as Shared Memory or NVLink). For instance, contemporary GPU clusters such as Nvidia

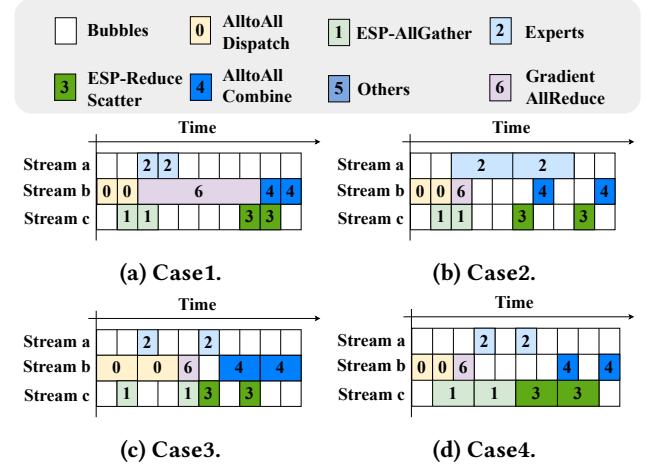


**Figure 3.** Backpropagation of four schedules in DP+MP+EP+ESP with the pipeline degree  $r = 4$  including (a) the default schedule, (b) an improved Tutel version (Tutel-Improved) where Gradient-AllReduce is overlapped with other dense operations using PipeMoE, (c) our proposed schedule FSMoE without partitioning the gradient, and (d) our proposed schedule FSMoE. The forward process is similar to the backpropagation except for the absence of the Gradient-AllReduce.

H100 DGX servers are equipped with eight 200Gb/s network interface cards (NICs), which collectively offer a peak bandwidth of 800Gb/s (equivalent to 100GB/s) for communication between any two nodes. In contrast, the NVLink within a server enables a bandwidth of 900GB/s, illustrating that the bandwidth within a single node is significantly greater than that between nodes. To balance both accuracy and training speed effectively, a practical approach is to align the MP and ESP with the number of GPUs contained within each node. For instance, when training Mixtral-8x7B with settings of  $N_{MP} = N_{ESP} = 8$  on servers that feature 8 A100-SXM4-80G GPUs, the approach is exactly feasible. This setup can also be simulated using a simulator<sup>2</sup>.

As shown in Fig. 3d, the inputs are split into several chunks and sequentially processed in a pipeline. Notably, Gradient-Allreduce is followed by the AlltoAll Dispatch on the last

<sup>2</sup><https://llm-system-requirements.streamlit.app/>



**Figure 4.** Four cases when scheduling the pipelining of ESP-AllGather/ESP-ReduceScatter, AlltoAll Dispatch/Combine, expert computations and Gradient-AllReduce with the pipeline degree  $r = 2$ . (a) **Case1:** The AlltoAll communications are slower than intra-node communication and expert computations, but the inter-node communications (AlltoAll and Gradient-AllReduce) are not slower than intra-node communication and expert computations. (b) **Case2:** Expert computations are not slower than inter-node communications and intra-node communications. (c) **Case3:** The AlltoAll communications are not slower than intra-node communication and expert computations. (d) **Case4:** The intra-node communications (AllGather and ReduceScatter) are not slower than inter-node communications and expert computations.

partitioned input as it can also be overlapped with ESP-AllGather/ESP-ReduceScatter and expert computations in the backward phase. The forward phase is similar to the backward phase, except for Gradient-Allreduce. In addition, the optimal pipeline degree varies by phase, necessitating phase-specific solutions. To achieve the new proposed schedule, we first build performance models of different time-consuming computing and communication tasks like PipeMoE [42] and FasterMoE [14]. We then formulate an optimization problem based on the performance model and propose an efficient solution.

#### 4.1 Performance Models

The time required for each chunk in the AlltoAll, AllGather, ReduceScatter, and expert computation processes on inputs divided into  $r$  chunks is represented by  $t_{a2a,r}$ ,  $t_{ag,r}$ ,  $t_{rs,r}$ , and  $t_{exp,r}$  respectively. These times are modelled via linear models [42] as follows (will verify in §6.2):

$$\begin{aligned} t_{a2a,r} &= \alpha_{a2a} + \frac{n_{a2a}}{r} \cdot \beta_{a2a}, \\ t_{ag,r} &= \alpha_{ag} + \frac{n_{ag}}{r} \cdot \beta_{ag}, \\ t_{rs,r} &= \alpha_{rs} + \frac{n_{rs}}{r} \cdot \beta_{rs}, \\ t_{exp,r} &= \alpha_{exp} + \frac{n_{exp}}{r} \cdot \beta_{exp}, \end{aligned} \quad (1)$$

where  $n_*$  represents the volume of the communication message or the computational workload,  $\alpha_*$  denotes the startup time and  $\beta_*$  represents the time per byte transmitted or per unit of workload processed. Particularly, when each expert computation includes multiple identical general matrix-multiplication (GEMM) operations,  $\alpha_{exp}$  and  $\beta_{exp}$  are determined by multiplying  $\alpha_{gemm}$  and  $\beta_{gemm}$  by the number of these operations.

## 4.2 Optimizing the Pipeline Degree

The performance model for both computation and communication supports optimizing the pipeline degree  $r$  to minimize time costs.

Direct optimization of overall time consumption is challenging because it relies on numerous factors. For instance, the start time of an ESP-ReduceScatter is constrained by both ESP-AllGather (inter-node communication contention) and Expert (data dependence). These constraints complicate finding effective solutions. We classify all general cases into four scenarios, as shown in Fig. 4 according to the main source of time consumption in each. For each case, we ease the complexity of the problem by focusing on certain constraints, thereby allowing the optimal solution to be obtained more straightforwardly. Specifically, (a) **Case1:** The AlltoAll communications are slower than intra-node communication and expert computations, but the inter-node communications (AlltoAll and Gradient-AllReduce) are not slower than intra-node communication and expert computations. (b) **Case2:** Expert computations are not slower than inter-node communications and intra-node communications. (c) **Case3:** The AlltoAll communications are not slower than intra-node communication and expert computations. (d) **Case4:** The intra-node communications (AllGather and ReduceScatter) are not slower than inter-node communications and expert computations. In situations where multiple time-consuming factors are equally significant, they can be categorized into one of these cases. For instance, when the time consumption for inter-node communication equals that of computation, it can fall into either Case1 or Case2. Before discussing these scenarios, the paper formulates seven constraints characterizing these cases, presented as follows.

$$\text{Q1: } t_{a2a,r} > t_{ag,r}.$$

**Q1 is True:** implies AlltoAll consumes more time than All-Gather for the chunked input. Assuming AllGather and ReduceScatter require similar durations, AlltoAll also exceeds ReduceScatter in time consumption.

$$\text{Q2: } r \cdot t_{exp,r} > 2(r - 1) \cdot t_{a2a,r}.$$

**Q2 is True:** indicates that expert computations exceed the duration of communication tasks, excluding AlltoAll Dispatch for the first and AlltoAll Combine for the last chunk. When **Q1 is True**, this also applies to AllGather and ReduceScatter for the first and last chunks, respectively.

$$\text{Q3: } r \cdot t_{exp,r} > (r - 1) \cdot (t_{ag,r} + t_{rs,r}).$$

**Q3 is True:** means that the time cost of expert computations is large enough to affect the time cost when **Q1 is False**.

$$\text{Q4: } t_{gar} > t_{ag,r} + t_{rs,r}.$$

**Q4 is True:** means that the time cost of Gradient AllReduce is large enough to affect the time cost when **Q1 is True and Q2 is False**.

$$\text{Q5: } t_{gar} > r \cdot t_{exp,r} - 2(r - 1) \cdot t_{a2a,r} + t_{ag,r} + t_{rs,r}.$$

**Q5 is True:** means that the time cost of Gradient AllReduce is large enough to affect the time cost when **Q1 is True and Q2 is True**.

$$\text{Q6: } t_{gar} > r \cdot t_{ag,r} + r \cdot t_{rs,r} - 2(r - 1) \cdot t_{a2a,r}.$$

**Q6 is True:** means that the time cost of Gradient AllReduce is large enough to affect the time cost when **Q1 is False and Q3 is False**.

$$\text{Q7: } t_{gar} > t_{ag,r} + t_{rs,r} + r \cdot t_{exp,r} - 2(r - 1) \cdot t_{a2a,r}.$$

**Q7 is True:** means that the time cost of Gradient-AllReduce is large enough to affect the time cost when **Q1 is False and Q3 is True**.

With these constraints, four cases can be represented as follows:

1) **Case1:** (*Q1 is True, Q2 is False and Q4 is True*) or (*Q1 is True, Q2 is True and Q5 is True*) or (*Q1 is False, Q3 is False and Q6 is True*) or (*Q1 is False, Q3 is True and Q7 is True*), which indicates that Gradient-AllReduce is large enough so that the inter-node communications (AlltoAll and Gradient-AllReduce) dominate the time cost in Fig. 4a. So we have

$$t_1^{moe} = 2r \cdot t_{a2a,r} + t_{gar} = 2r\alpha_{a2a} + 2n_{a2a}\beta_{a2a} + t_{gar}. \quad (2)$$

Therefore, to find its minima,  $t_1^*$ , we should solve

$$\text{minimize: } f_1(r) = t_1^{moe},$$

$$\text{s.t. } r \geq 1,$$

$$(Q1 \wedge \neg Q2 \wedge Q4) \vee (Q1 \wedge Q2 \wedge Q5)$$

$$\vee (\neg Q1 \wedge \neg Q3 \wedge Q6) \vee (\neg Q1 \wedge Q3 \wedge Q7).$$

2) **Case2:** (*Q1 is True, Q2 is True and Q5 is False*) or (*Q1 is False, Q3 is True and Q7 is False*), which indicates that Gradient-Allreduce is too small to influence the time cost and expert computation occupies a dominant position in Fig. 4b. So we have

$$\begin{aligned} t_2^{moe} &= 2t_{a2a,r} + t_{ag,r} + t_{rs,r} + r \cdot t_{exp,r} \\ &= 2\alpha_{a2a} + \frac{2n_{a2a}}{r}\beta_{a2a} + \alpha_{ag} + \frac{n_{ag}}{r}\beta_{ag} \\ &\quad + \alpha_{rs} + \frac{n_{rs}}{r}\beta_{rs} + r\alpha_{exp} + n_{exp}\beta_{exp}. \end{aligned}$$

Therefore, to find its minima,  $t_2^*$ , we should solve

$$\text{minimize: } f_2(r) = t_2^{moe},$$

$$\text{s.t. } r \geq 1,$$

$$(Q1 \wedge Q2 \wedge \neg Q5) \vee (\neg Q1 \wedge Q3 \wedge \neg Q7).$$

3) **Case3:** *Q1 is True, Q2 is False and Q4 is False*, which indicates that Gradient-Allreduce and expert computation are

**Algorithm 1** FindOptimalPipelineDegree

---

**Input:**  $\alpha_{a2a}, \beta_{a2a}, n_{a2a}, \alpha_{ag}, \beta_{ag}, n_{ag}, \alpha_{rs}, \beta_{rs}, n_{rs}, \alpha_{exp}, \beta_{exp}, n_{exp}, t_{gar}$

**Output:**  $r$  and  $t^{moe}$

```

1:  $r1, t1 = solve(f_1)$                                 ▷ Solve with SLSQP
2:  $r2, t2 = solve(f_2)$ 
3:  $r3, t3 = solve(f_3)$ 
4:  $r4, t4 = solve(f_4)$ 
5:  $candidate\_mins = [t1, t2, t3, t4]$ 
6:  $candidates = [r1, r2, r3, r4]$ 
7:  $r = candidates[\text{argmin}(candidate\_mins)]$ 
8:  $t^{moe} = \min(candidate\_mins)$ 
9: return  $r$  and  $t^{moe}$ .
```

---

too small to influence the time cost. The communications dominate the time cost. And AlltoAll also takes more time than AllGather and ReduceScatter on a chunked tensor in Fig. 4c. So we have

$$\begin{aligned} t_3^{moe} &= 2r \cdot t_{a2a,r} + t_{ag,r} + t_{rs,r} \\ &= 2r\alpha_{a2a} + 2n_{a2a}\beta_{a2a} + \alpha_{ag} + \frac{n_{ag}}{r}\beta_{ag} + \alpha_{rs} + \frac{n_{rs}}{r}\beta_{rs}. \end{aligned}$$

Therefore, to find its minima,  $t_3^*$ , we should solve

$$\begin{aligned} \text{minimize: } f_3(r) &= t_3^{moe}, \\ \text{s.t. } r &\geq 1, \\ Q1 \wedge \neg Q2 \wedge \neg Q4. \end{aligned}$$

4) **Case4:**  $Q1$  is False,  $Q3$  is False and  $Q6$  is False, which indicates that Gradient-Allreduce and expert computation are too small to influence the time cost. And AllGather and ReduceScatter also take more time than AlltoAll on a partitioned tensor. Intra-node communications dominate the time cost in Fig. 4d. So we have

$$\begin{aligned} t_4^{moe} &= 2t_{a2a,r} + r \cdot t_{ag,r} + r \cdot t_{rs,r} \\ &= 2\alpha_{a2a} + \frac{2n_{a2a}}{r}\beta_{a2a} + r\alpha_{ag} + n_{ag}\beta_{ag} + r\alpha_{rs} + n_{rs}\beta_{rs}. \end{aligned}$$

Therefore, to find its minima,  $t_4^*$ , we should solve

$$\begin{aligned} \text{minimize: } f_4(r) &= t_4^{moe}, \\ \text{s.t. } r &\geq 1, \\ \neg Q1 \wedge \neg Q3 \wedge \neg Q6. \end{aligned}$$

### 4.3 Algorithm

Algorithm 1 determines the optimal pipeline degree using MoE-related coefficients ( $n_{a2a}, n_{ag}, n_{rs}, n_{exp}$ ) and cluster-related coefficients ( $\alpha_{a2a}, \beta_{a2a}, \alpha_{ag}, \beta_{ag}, \alpha_{rs}, \beta_{rs}, \alpha_{exp}, \beta_{exp}$ ). In particular,  $t_{gar}$  is a manually entered value that is set to zero in the forward process and determined by §5 in the backward process. FSMoE supports varied pipeline degrees in both phases. The algorithm executes once before training, following the estimation of cluster-related coefficients. The “solve” function employs a sequential least squares programming (SLSQP) [32] solver. This algorithm is quadratic

convergence in solving  $f_1, f_2, f_3$  and  $f_4$  (Lines 1-4), and other operations take  $O(1)$  time complexity.

### 4.4 Schedule Forward and Backward Separately

Because of the calculation of gradient w.r.t. the weight and the gradient synchronization among DP workers, the tasks in backpropagation are different from the forward phrase. The optimal pipeline degree thus differs. Therefore, we manually implement the backpropagation by storing the activation of each computational operation and computing the gradient.

Specifically, the parameters  $\alpha_{exp}, \beta_{exp}$ , and  $n_{exp}$  in the backward phase are twice those in the forward phase to accommodate the derivatives of both weight and input. Meanwhile,  $t_{gar}$  is set to zero in the forward phase as gradient synchronization does not occur, and it is determined by the algorithm detailed in §5 for the backward phase.

## 5 Scheduling for Backpropagation

Due to the inter-node communication in the MoE layer, Gradient-AllReduce of the gradient synchronization can not be directly overlapped with MoE layers. A dedicated co-design is necessary to further hide the time cost of Gradient-AllReduce. We propose to adaptively partition the gradients to achieve the maximal overlap of Gradient-AllReduce with other operations.

Our approach contains two steps. Step 1: We calculate the time cost of the parts that can be overlapped with Gradient-AllReduce (denoted as *overlappable parts*) for all layers. Then we slice the gradient and assign them to these overlapable parts as far as possible. Step 2: We arrange the remaining gradient after the first step and set the remaining gradient partitioned to each MoE layer as variables to optimize the assignment.

### 5.1 Performance Model

Similar to §4.1, the performance model of AllReduce can be represented as  $t_{ar}(n_{ar}) = \alpha_{ar} + n_{ar} \cdot \beta_{ar}$ , where  $t_{ar}$  denotes the elapsed-time,  $n_{ar}$  represents the amount of the communication message,  $\alpha_{ar}$  denotes the starup time and  $\beta_{ar}$  represents the transmission time per byte. The inverse function of  $t_{ar}(n_{ar})$  is represented as  $g_{grad}^{inv}(t_{ar}) = (t_{ar} - \alpha_{ar})/\beta_{ar}$ .

### 5.2 Step 1: Calculate Partitioned Gradients

In this step, we first optimize the pipeline degree of each MoE layer with  $t_{gar} = 0$  by Algorithm 1 to calculate the time cost of *overlappable parts*. Then we try to slice the gradient and assign them to *overlappable parts* of each layer in order to minimize the training time. According to the above performance model, we are able to calculate the gradient assigned to each layer.

For convenience, we denote an MoE layer and other operations before the next MoE layer as a generalized layer. We denote the gradient for a generalized layer  $i$  as  $n_{grad}^i$  and the

time cost of *overlappable parts* as  $t_{olp}^i$ . The number of gradients assigned to each layer in this step can be represented as

$$n_{first}^i = g_{grad}^{inv}(\min(t_{grad}(n_{grad}^{i-1}), t_{olp}^i)). \quad (3)$$

If  $n_{grad}^{i-1}$  is not fully overlapped,  $n_{grad}^i$  should be updated by

$$n_{grad}^i = n_{grad}^{i-1} + g_{grad}^{inv}(\max(t_{grad}(n_{grad}^{i-1}) - t_{olp}^i, 0)). \quad (4)$$

Notably, the time cost of *overlappable parts*,  $t_{olp}$ , can be divided into sparse MoE parts  $t_{olp,moe}$  and dense parts  $t_{olp,dense}$ . The dense parts  $t_{olp,dense}$  can be measured before the training, while  $t_{olp,moe}$  can be calculated during the optimization of the pipeline degree. Specifically, when  $t_{gar} = 0$ , we will encounter Case2, Case3 and Case4 mentioned in §4.2. And  $t_{olp,moe}$  can be formulated as following

$$t_{olp,moe}(r) = \begin{cases} r \cdot t_{exp,r} + t_{ag,r} + t_{rs,r} - 2(r-1)t_{a2a,r}, & \text{Case 2} \\ t_{ag,r} + t_{rs,r}, & \text{Case 3} \\ r \cdot t_{ag,r} + r \cdot t_{rs,r} - 2(r-1)t_{a2a,r}, & \text{Case 4} \end{cases}$$

After the above process, we will enter the second step if gradients still remain.

### 5.3 Step 2: Optimize Partitioning

The second step is to assign the remaining gradients after the first step. Note that with different input time costs of Gradient-AllReduce, the optimization algorithm (Algorithm 1) would produce different degrees and time costs. It indicates that the remaining gradients can be further partitioned into MoE layers to minimize the training time.

We denote the remained gradient for the generalized layer  $i$  as  $n_{rem}^i$  and the Algorithm 1 as  $f_{moe}^i(t_{gar})$  who takes the time cost of Gradient-AllReduce as the input and produces the time cost of the MoE layer  $i$ . Then, set the remaining gradient assigned to the MoE layer  $i$  as  $x_g^i$ . The optimization model can be represented as

$$\begin{aligned} \text{minimize: } f_g(X_g) &= \sum_{i=1}^{n_l} f_{moe}^i(t_{grad}(x_g^i)), \\ \text{s.t. } 0 \leq x_g^i &< n_{rem}^i + \sum_{j=i-1}^{n_l} (n_{rem}^j - x_g^j), 0 < i < n_l, \end{aligned} \quad (5)$$

where  $n_l$  represents the number of layers. As the optimization will be conducted only once before the training, we do not need to care too much about the time cost. Therefore, we simply adopt the differential evolution algorithm [35] when we solve the above optimization problem.

## 6 EVALUATION

### 6.1 Experimental Settings

We mainly compare our FSMoE with Tutel [17] (w/ its optimized version PipeMoE [42]) which designs an adaptive schedule to determine the pipeline degree of the overlaps,

**Table 3.** The server configurations in our testbeds.

Name	Testbed A	Testbed B
CPU	Dual Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz	Dual Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz
GPU	8x Nvidia RTXA6000 @1.46GHz 48GB Mem	4x Nvidia RTX2080Ti @1.35GHz 11GB Memory
Memory	512GB DDR4	512GB DDR4
NVlink	112.5GB/s (4x)	-
PCIe	4.0 (x16)	3.0 (x16)
Network	Mellanox MT28908 @ 200Gb/s	Mellanox MT27800 @ 100Gb/s

**Table 4.** Configurations of attention and MoE layers.  $N_{hscale} = H/M$ .  $f = *$  means tokens will not be dropped when gating. *ffn-type* means the type of experts in MoE.

-	Candidate Values
$B$	{1,2,4}
$N_{heads}$	{8,16,32}
$L$	{512,1024,2048}/{256,512,1024}
$M$	{1024, 2048, 4096}
$N_{hscale}$	{2,3,4}
$f$	{1,2,2,4,*}
<i>ffn-type</i>	{simply,Mixtral}

with a focus on pipelining communications and computations in a typical structure of the MoE model in DP+MP+EP+ESP shown in Fig. 2. Additionally, we compare the end-to-end training performance of FSMoE with DeepSpeed-MoE [2, 39], which is a dedicated MoE training system. The code we implemented is accessible at <https://github.com/xpan413/FSMoE>.

**Testbeds:** Experiments are carried out on two distinct testbeds: Testbed-A, a 48-GPU cluster comprising six interconnected nodes, and each node is equipped with four Nvidia A6000 GPUs. Testbed-B, a 32-GPU cluster comprising eight interconnected nodes, and each node is equipped with four Nvidia GeForce RTX2080Ti GPUs. More details on the server configuration can be found in Table 3. The software environments are Ubuntu-20.04, CUDA-11.3, PyTorch-1.12 and NCCL-2.12.

**MoE model configurations.** We select a combination of input parameters whose ranges are shown in Table 4 to cover a variety of typical configurations of attention and MoE layers.  $L$  is set to {256, 512, 1024} on Testbed-B due to the memory limit of 2080Ti. Notably, we select a range of  $N_{hscale} = H/M$  rather than directly setting  $H$ , which is more common in real-world scenarios.  $f = *$  means tokens will not be dropped when gating. *ffn-type* means the type of experts in MoE. *simple* represents the conventional two feed-forward dense layers and *Mixtral* means the experts using in Mixtral [20]. Additionally,  $N_{MP}$  and  $N_{ESP}$  are both set to 4 in Testbed-B where ESP-AllGather and ESP-ReduceScatter

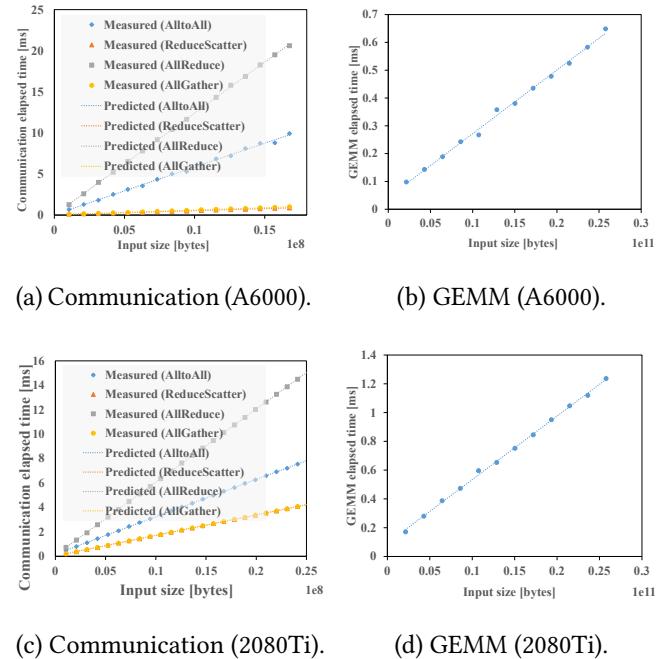
are intra-node communications while AlltoAll and Gradient-AllReduce are inter-node communications. Similarly,  $N_{MP}$  and  $N_{ESP}$  are both set to 8 in Testbed-A.

## 6.2 Performance Model

We require the input parameters that are related to the cluster for the performance models of computation and communication. We measure the elapsed time with a range of sizes for GEMM computation and four types of communication to fit the performance models in Eq. 1 using microbenchmark tools. In particular, we utilize the NCCL-2.12 collective communication primitives along with *nccl-tests*<sup>3</sup> to evaluate communication durations across diverse message sizes. Meanwhile, we employ the *torch.matmul*<sup>4</sup> function in PyTorch to assess the GEMM execution times for matrices of varying shapes. For communication modeling, float-type elements are chosen in a range from  $2^{18}$  to  $24 \times 2^{18}$ , with steps of  $2^{18}$ , to simulate different tensor sizes. Likewise, for the GEMM modeling, float-type elements are picked from a range between  $2^{19}$  and  $12 \times 2^{19}$ , with  $2^{19}$  increments. Each measurement is averaged over five runs to ensure consistency. The results are shown in Fig. 5. It is seen that our linear models with intercept terms (i.e., startup time) can well fit the measured performance. Specifically, the  $r^2$  for our GEMM model is 0.9987, and the corresponding  $r^2$  for the communication tasks are as follows: AllReduce: 0.9999896, AlltoAll: 0.9999, AllGather: 0.9999653, and ReduceScatter: 0.9999599. The total time required for both computation and communication in the performance models is under 100 seconds. Fitting through the least squares method takes under 10ms. Following fitting, the empirical time cost for SLSQP in solving  $r$  averages 193ms over 1458 configured cases. When dealing with a new GPU cluster, it is only necessary to estimate the parameters one time using micro-benchmarks prior to model training, without impacting the training efficiency.

**Table 5.** Averaged speedups of four schedules over Tutel (w/ its optimized version PipeMoE) on configured layers in Table 4. Tutel-Improved means using PipeMoE with Gradient-AllReduce overlapped with non-MoE parts, while FSMoE-No-IIo indicates using FSMoE without the overlaps between inter and intra node communications.

Schedule	Speedup	
	Testbed-A	Testbed-B
Tutel	1.00×	1.00×
Tutel-Improved	1.09×	1.08×
FSMoE-No-IIo	1.12×	1.16×
FSMoE	1.18×	1.22×



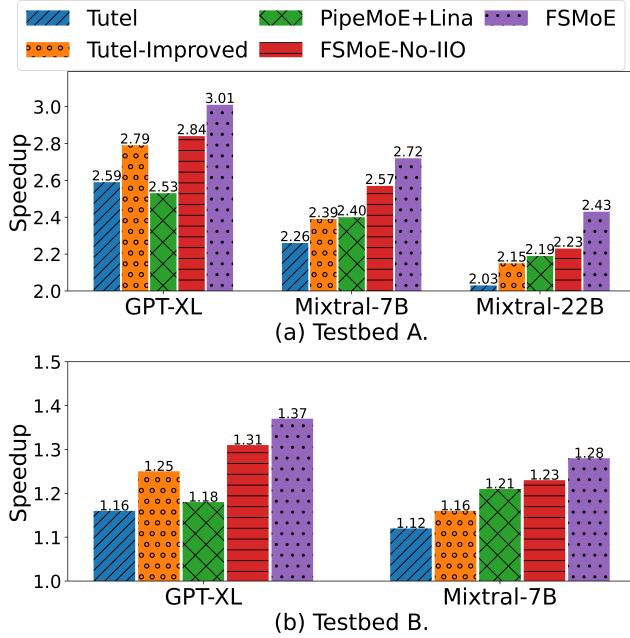
**Figure 5.** Performance models. Markers are measured values and lines are predicted values with estimated parameters. (a)  $\alpha_{gemm}=4.26e-2$ ,  $\beta_{gemm}=2.29e-11$  on Testbed-A. (b)  $\alpha_{a2a}=2.87e-1$ ,  $\beta_{a2a}=2.21e-7$ ,  $\alpha_{ag}=3.37e-1$ ,  $\beta_{ag}=2.32e-6$ ,  $\alpha_{rs}=3.95e-1$ ,  $\beta_{rs}=2.34e-7$ ,  $\alpha_{ar}=5.11e-1$ ,  $\beta_{ar}=4.95e-6$  on Testbed-A. (c)  $\alpha_{gemm}=9.24e-2$ ,  $\beta_{gemm}=4.42e-11$  on Testbed-B. (d)  $\alpha_{a2a}=1.75e-1$ ,  $\beta_{a2a}=3.06e-7$ ,  $\alpha_{ag}=3.20e-2$ ,  $\beta_{ag}=1.68e-7$ ,  $\alpha_{rs}=3.91e-2$ ,  $\beta_{rs}=1.67e-7$ ,  $\alpha_{ar}=8.37e-2$ ,  $\beta_{ar}=5.99e-7$  on Testbed-B.

## 6.3 Performance on Configured Layers

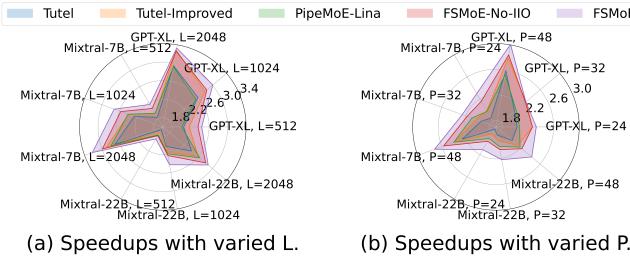
We conducted a comparison between our proposed method FSMoE and PipeMoE [42] in the structure illustrated in Fig. 2, using various configurations as outlined in Table 5. Notably, the gradient aggregation of a configured layer is added in order to validate our gradient partitioning method and schedule to overlap Gradient-AllReduce. For better comparison, experiments on two additional schedules (Tutel-Improved and FSMoE-No-IIo) are further conducted. Tutel-Improved means PipeMoE with Gradient-AllReduce overlapped with non-MoE parts, while FSMoE-No-IIo means FSMoE without the overlaps between inter and intra-node communications. The experimental results indicate that with a simple overlap between Gradient-AllReduce with non-MoE parts, we can achieve a speed up of 1.08× to 1.09× over Tutel (w/ PipeMoE). And with our gradient partitioning and well overlaps among inter-node and intra-node communication as well as computation tasks, FSMoE achieves an average speedup of 1.18× to 1.22× over Tutel across 1458 cases. By comparing the speed up of our FSMoE and FSMoE-No-IIo

<sup>3</sup><https://github.com/NVIDIA/nccl-tests>

<sup>4</sup><https://pytorch.org/docs/stable/generated/torch.matmul.html>



**Figure 6.** Speedups of FSMoE, FSMoE-No-IIo, Tutel, Tutel-Improved, PipeMoE+Lina (PipeMoE with the additional schedule introduced by Lina [24] that partitions the gradient into fixed chunk size) over DeepSpeed-MoE (DS-MoE) on MoE models (GPT2-XL, Mixtral-7B and Mixtral-22B).

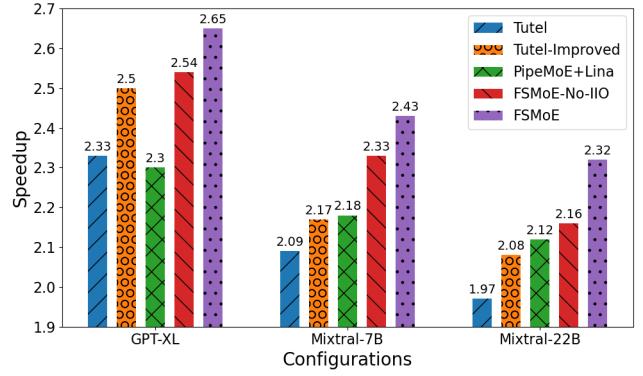


**Figure 7.** Speedups of five schedules over DS-MoE on Testbed-A with different configurations.

over Tutel in Table 5, we see that the overlaps between inter-node and intra-node communications further improve the performance.

#### 6.4 Performance on MoE Models

To evaluate the end-to-end training performance, we conduct experiments with Mixtral-7B [20] and an MoE model based on GPT-2 [38] on two testbeds. In addition, experiments in Mixtral-22B are also conducted on Testbed-A. We set  $B = 1$ ,  $k = 2$ ,  $f = 1.2$  during the experiment. To enable the overlap between inter and intra communication,  $N_{ESP} = N_{MP}$ , which is 8 and 4 on Testbed-A and Testbed-B, respectively. Furthermore, the number of experts ( $N_{EP}$ ) is the same as the number of nodes, which is 6 and 8 on



**Figure 8.** Speedups of five schedules over DS-MoE on Testbed-A when PP is enabled.

Testbed-A and Testbed-B, respectively.  $L$  is set to 256 on Testbed-B and to 1024 on Testbed-A. Ensuring the models to be held on Testbed-B (32x 2080Ti 11GB), we set the number of layers for Mixtral-7B to 7. Due to the memory limit, the number of layers for Mixtral-22B is set to 33 on Testbed-A. For further analysis, experiments on two additional schedules are conducted. Tutel-Improved indicates Tutel with the overlaps between Gradient-AllReduce with non-MoE parts using PipeMoE. PipeMoE+Lina means PipeMoE with the additional schedule introduced by Lina [24] that partitions the gradient into fixed chunk size (e.g., 30MB) and overlaps the partitioned gradient aggregation with expert computations and non-MoE parts in backpropagation.

The results in Fig. 6 indicate that FSMoE achieves a speedup of  $1.28\times$  to  $3.01\times$  compared to DeepSpeed-MoE (DS-MoE) while Tutel can only achieve a speedup of  $1.16\times$  to  $2.59\times$ . Additionally, FSMoE can achieve an average speedup of  $1.19\times$  over Tutel,  $1.12\times$  over Tutel-Improved,  $1.14\times$  over PipeMoE+Lina and  $1.07\times$  over FSMoE-No-IIo, which validates the efficiency of our adaptive gradient partitioning method and pipelining schedule. It is worth mentioning that Lina's idea of partitioning gradients and scheduling the gradient aggregation can not handle various configurations due to the fixed chunk size. Thus, its performance is hit or miss. And our FSMoE can adaptively partition the gradient and adjust the pipelining degree to achieve better results.

**Performance on MoE Models With PP Enabled.** We also conduct experiments on Testbed-A when PP is further enabled ( $N_{PP} = 2$ ), implemented using GPipe [15]. The results are shown in Fig 8. The results indicate that FSMoE can achieve an average speedup of  $2.46\times$  over DS-MoE,  $1.16\times$  over Tutel,  $1.10\times$  over Tutel-Improved,  $1.12\times$  over PipeMoE+Lina and  $1.05\times$  over FSMoE-No-IIo.

**Performance on MoE Models with Varied  $L$  and Varied  $P$ .** Moreover, we analyze the performance of FSMoE with varied  $L$  and  $P$  on Testbed-A.  $L$  is varied in {512, 1024 and 2048 } while  $P$  is varied in {16, 32 and 48 }. The results are

shown in Fig 7. The results indicate that FSMoE can achieve an average speedup of  $2.17\times$ ,  $2.72\times$  and  $3.14\times$  over DS-MoE and  $1.17\times$ ,  $1.19\times$  and  $1.17\times$  over Tutel when  $L$  is varied in {512, 1024 and 2048 } and  $P = 48$ . FSMoE can achieve an average speedup of  $2.25\times$ ,  $2.27\times$  and  $2.72\times$  over DS-MoE,  $1.20\times$ ,  $1.16\times$  and  $1.19\times$  over Tutel when  $P$  is varied in {16, 32 and 48 } and  $L = 1024$ . It indicates the robustness of FSMoE.

**Support Multiple Gating Functions.** Table 6 underscores the ability of our framework to support multiple gating functions while maintaining improved efficiency. Our framework shows potential scalability and flexibility in handling complex MoE architectures.

**Table 6.** Time performance on Testbed-B (average iteration time in milliseconds) of various gating on real-world MoE GPT2-XL. The lower is better. Speedup are provided in parentheses.

Gating	DeepSpeed-MoE	FSMoE
Gshard [22]	$968.1 \pm 1.4$	$707.7 \pm 1.6(1.37\times)$
X-MoE [6]	$1064.0 \pm 1.5$	$746.9 \pm 2.8(1.42\times)$
Sigmoid [23]	$986.6 \pm 1.4$	$721.0 \pm 1.8(1.37\times)$
EC [51]	$909.9 \pm 1.8$	$685.5 \pm 1.5(1.33\times)$

## 7 Related Work

In optimizing the training performance of MoE models, there are three main orthogonal directions that have been explored. These directions include MoE algorithms, AlltoAll algorithms, and scheduling algorithms. While MoE algorithms focus on workload balancing and designing gating functions, and AlltoAll algorithms aim to improve data dispatch and combine efficiency, our primary focus lies on MoE systems and scheduling algorithms that aim to reduce communication time, so we mainly introduce the related studies in this direction.

Tutel [17] and DeepSpeed-MoE [39] stand out as specialized optimized systems for training MoE models. These frameworks incorporate a multitude of optimization techniques. However, their current capabilities are limited to manual configuration of the pipeline degree or heuristic search methods within a constrained search space. Contrasting Tutel, FasterMoE [14] allows partitioning input tokens into two groups for the overlaps between expert computations and AlltoAll communications. Built on Tutel, PipeMoE [42] proposes an innovative and optimal partitioning methodology for input tokens. Lina [24] aims to alleviate network contention during backpropagation by addressing the challenges associated with AllReduce and AlltoAll operations.

Subsequently, various studies concern the fine-grain overlap between communication and computation. T3 [34] introduces a hardware-software co-design approach to seamlessly integrate serialized communication with computation,

thus reducing resource conflicts. Wang et al. [46] enhance overlapping by using semantically equivalent graph transformations, implemented in XLA. Punniyamurthy et al. [37] tackle the issue of collective communication overhead in DLRM. FLUX [4] and CoCoNet [18] break down the initial communication and computation into much smaller, more detailed tiles compared to current methods. Subsequently, it combines the tiled computation and communication into a unified kernel. Shi et al. [41] propose to exploit simultaneous communication streams to improve the bandwidth utilization of AllReduce communications. Their approaches could enhance our method by addressing the competition for resources between communication and computation.

## 8 Conclusion

In this work, we present a flexible training system named FSMoE to optimize task scheduling. To achieve this goal: 1) we design unified abstraction and online profiling of MoE modules across various MoE implementations, 2) we co-schedule intra-node and inter-node communications with computations to minimize communication overhead, and 3) we design an adaptive gradient partitioning method for gradient aggregation and a schedule to adaptively pipeline communications and computations. Experimental results on two clusters up to 48 GPUs show that our FSMoE outperforms the state-of-the-art MoE training systems (DeepSpeed-MoE and Tutel) with speedups of  $1.18\times$  to  $1.22\times$  on 1458 customized MoE layers and  $1.19\times$  to  $3.01\times$  on real-world MoE models based on GPT-2 and Mixtral.

## Acknowledgments

We extend our heartfelt gratitude to the anonymous reviewers whose insightful and constructive feedback has been instrumental in elevating the quality of this paper. Their astute comments and suggestions have significantly contributed to refining our research work. The research was supported in part by National Science Foundation of China (NSFC) grants under Grant No. 62272122, and Grant No. 62302123, Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies under Grant 2022B121201005, the Guangzhou Municipal Joint Funding Project with Universities and Enterprises under Grant No. 2024A03J0616, Shenzhen Science and Technology Program under Grant No. KJZD2023092311513026 and KJZD20230923114213027, a RGC RIF grant under the contract R6021-20, RGC TRS grant under the contract T43-513/23N-2, a Hong Kong RIF grant under the Grant No. R6021-20, Hong Kong CRF grants under Grant No. C2004-21G, C7004-22G, C1029-22G, and C6015-23G, and RGC GRF grants under the contracts 16200221, 16207922 and 16207423. Shaohuai Shi and Xiaowen Chu are the corresponding authors.

## References

- [1] Doubling all2all performance with nvidia collective communication library 2.12. <https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/>. Accessed: 2022-07-13.
- [2] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2022.
- [3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [4] Li-Wen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Ziheng Jiang, Haibin Lin, Xin Jin, and Xin Liu. FLUX: fast software-based communication overlap on gpus through kernel fusion. *CoRR*, abs/2406.06858, 2024.
- [5] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. 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*, pages 178–191, 2024.
- [6] Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, Xian-Ling Mao, et al. On the representation collapse of sparse mixture of experts. *Advances in Neural Information Processing Systems*, 35:34600–34613, 2022.
- [7] Aakanksha Chowdhery, Sharun Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.
- [8] Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. Stablemoe: Stable routing strategy for mixture of experts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7085–7095, 2022.
- [9] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc’arelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.
- [10] DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.
- [11] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.
- [12] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.
- [13] Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fastmoe: A fast mixture-of-expert training system. *arXiv preprint arXiv:2103.13262*, 2021.
- [14] 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.
- [15] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.
- [16] Yongqi Huang, Peng Ye, Xiaoshui Huang, Sheng Li, Tao Chen, and Wanli Ouyang. Experts weights averaging: A new general training scheme for vision transformers. *arXiv preprint arXiv:2308.06093*, 2023.
- [17] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhakar Ram, et al. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems*, 5, 2023.
- [18] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In *ASPLoS*, pages 402–416. ACM, 2022.
- [19] Xianyan Jia, Shutao Song, Shaohuai Shi, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Lijiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training ImageNet in four minutes. In *Proc. of Workshop on Systems for ML and Open Source Software, collocated with NeurIPS 2018*, 2018.
- [20] 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.
- [21] Chenyu Jiang, Ye Tian, Zhen Jia, Chuan Wu, Yida Wang, and Shuai Zheng. Lancet: Accelerating mixture-of-experts training by overlapping weight gradient computation and all-to-all communication. *Proceedings of Machine Learning and Systems*, 6:74–86, 2024.
- [22] 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. In *International Conference on Learning Representations*, 2020.
- [23] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. BASE layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pages 6265–6274. PMLR, 2021.
- [24] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed {MoE} training and inference with lina. In *USENIX Annual Technical Conference*, pages 945–959, 2023.
- [25] Juncai Liu, Jessie Hui Wang, and Yimin Jiang. Janus: A unified distributed training framework for sparse mixture-of-experts models. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 486–498, 2023.
- [26] Rui Liu, Young Jin Kim, Alexandre Muzio, and Hany Hassan. Gating dropout: Communication-efficient regularization for sparsely activated transformers. In *International Conference on Machine Learning*, pages 13782–13792. PMLR, 2022.
- [27] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1930–1939, 2018.
- [28] Zixuan Ma, Jiaao He, Jiezhong Qiu, Huanqi Cao, Yuanwei Wang, Zhenbo Sun, Liyan Zheng, Haojie Wang, Shizhi Tang, Tianyu Zheng, et al. Bagualu: targeting brain scale pretrained models with over 37 million cores. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 192–204, 2022.
- [29] 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.

- [30] 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.
- [31] 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.
- [32] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.
- [33] Xinglin Pan, Wenxiang Lin, Shaohuai Shi, Xiaowen Chu, Weinong Sun, and Bo Li. Parm: Efficient training of large sparsely-activated models with dedicated schedules. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, 2024.
- [34] Suchita Pati, Shaizeen Aga, Mahzabeen Islam, Nuwan Jayasena, and Matthew D. Sinclair. T3: transparent tracking & triggering for fine-grained overlap of compute & collectives. In *ASPLOS (2)*, pages 1146–1164. ACM, 2024.
- [35] Kenneth V Price. Differential evolution. In *Handbook of optimization: From classical to modern approach*, pages 187–214. Springer, 2013.
- [36] Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts. *arXiv preprint arXiv:2308.00951*, 2023.
- [37] Kishore Punniyamurthy, Khaled Hamidouche, and Bradford M. Beckmann. Optimizing distributed ml communication with fused computation-collective operations, 2024.
- [38] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [39] 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.
- [40] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2016.
- [41] Shaohuai Shi, Xiaowen Chu, and Bo Li. Exploiting simultaneous communications to accelerate data parallel distributed deep learning. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.
- [42] Shaohuai Shi, Xinglin Pan, Xiaowen Chu, and Bo Li. PipeMoe: Accelerating mixture-of-experts through adaptive pipelining. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, 2023.
- [43] 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.
- [44] 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.
- [45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [46] 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. Overlap communication with dependent computation via decomposition in large deep learning models. In *ASPLOS (1)*, pages 93–106. ACM, 2023.
- [47] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Sri-nadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, 2020.
- [48] Zhao You, Shulin Feng, Dan Su, and Dong Yu. Speechmoe: Scaling to large acoustic models with dynamic routing mixture of experts. *arXiv preprint arXiv:2105.03036*, 2021.
- [49] 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 *USENIX Annual Technical Conference*, pages 961–975, 2023.
- [50] Ningxin Zheng, Huiqiang Jiang, Quanlu Zhang, Zhenhua Han, Lingxiao Ma, Yuqing Yang, Fan Yang, Chengruidong Zhang, Lili Qiu, Mao Yang, et al. Pit: Optimization of dynamic sparse deep learning models via permutation invariant transformation. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 331–347, 2023.
- [51] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew Dai, Zhifeng Chen, Quoc Le, and James Laudon. Mixture-of-experts with expert choice routing. *arXiv preprint arXiv:2202.09368*, 2022.
- [52] Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Jianfeng Gao, and Tuo Zhao. Taming sparsely activated transformer with stochastic experts. In *International Conference on Learning Representations*, 2021.