NetMoE: Accelerating MoE Training through Dynamic Sample Placement

Xinyi Liu[1] **Yujie Wang**[1] **Fangcheng Fu**[1] **Xupeng Miao**[2] **Shenhan Zhu**[1]
Xiaonan Nie[1] **Bin Cui**[1,3]
[1]School of CS & Key Lab of High Confidence Software Technologies (MOE), Peking University
[2]Purdue University [3]Institute of Computational Social Science, Peking University (Qingdao)
{xy.liu, alfredwang, ccchengff}@pku.edu.cn
xupeng@purdue.edu, {shenhan.zhu, xiaonan.nie, bin.cui}@pku.edu.cn

Abstract

Mixture of Experts (MoE) is a widely used technique to expand model sizes for better model quality while maintaining the computation cost constant. In a nutshell, an MoE model consists of multiple experts in each model layer and routes the training tokens to only a fixed number of experts rather than all. In distributed training, as experts are distributed among different GPUs, All-to-All communication is necessary to exchange the training tokens among the GPUs after each time of expert routing. Due to the frequent and voluminous data exchanges, All-to-All communication has become a notable challenge to training efficiency.

In this paper, we manage to accelerate All-to-All communication in MoE models from the training sample perspective, which is unexplored so far. In particular, we put forward the observation that tokens in the same training sample have certain levels of locality in expert routing. Motivated by this, we develop NetMoE, which takes such locality into account and dynamically rearranges the placement of training samples to minimize All-to-All communication costs. Specifically, we model the All-to-All communication given the sample placement and formulate an integer programming problem to deduce the optimal placement in polynomial time. Experiments with 32 GPUs show that NetMoE achieves a maximum efficiency improvement of $1.67\times$ compared with current MoE training frameworks.

1 Introduction

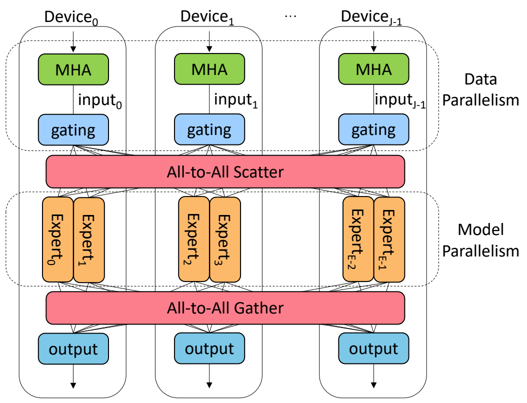
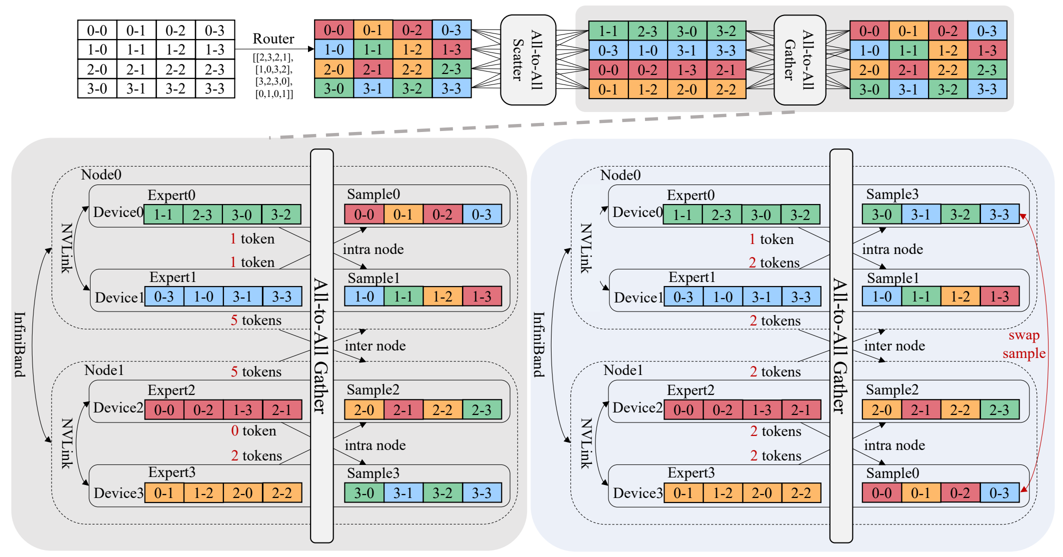


Figure 1: An example of expert parallelism applied to an MoE model with J devices and $E = 2J$ experts (each device has two different experts).

In recent years, large language models (LLMs) have shown impressive performance in language understanding and generation (OpenAI, 2023; Touvron et al., 2023; Zhou et al., 2024; Dubey et al., 2024; Shao et al., 2024; Zhang et al., 2024a) due to the increasing model size. However, larger models often come with greater computational costs. To address this, Mixture of Experts (MoE) models have been introduced to expand the model size greatly without increasing the computational cost. Combining MoE with Transformer-based models can yield outstanding performance across various tasks, including natural language processing (Lepikhin et al., 2021; Fedus et al., 2022), computer vision (Riquelme et al., 2021; Liang et al., 2022), recommendation systems (Tang et al., 2020; Zou et al., 2022), and speech recognition (You et al., 2022; Kwon & Chung, 2023).

MoE models often replace the feed-forward network (FFN) layer with the MoE layer, which consists of a gating network and several small FFNs, representing different experts. In the MoE layer, each token is routed by the gating network to only a few selected experts, and the final output is

(a) An overview of a MoE layer example.



(b) A gather operation in the MoE layer without adjusting the sample placement.

(c) A gather operation in the MoE layer after sample placement adjustment is enabled.

Figure 2: An example of sample exchange. The figure illustrates the All-to-All gather operation in a MoE layer with two nodes, each containing two devices, and each device having one expert. Different colors represent tokens sent to different experts, and i-j denotes the j-th token in the i-th sample. Fig. 2(a) illustrates the complete process of a MoE layer during forward propagation. Fig. 2(b) shows the All-to-All gather operation in the MoE layer without adjusting the sample placement, where the inter-node communication volume of each node is 5 tokens. Fig. 2(c) displays the All-to-All gather operation after sample placement adjustment is enabled — the positions of samples on the devices change (samples 0 and 3 are exchanged), reducing the inter-node communication volume to 2 tokens per node.

obtained by a weighted sum of the computations from the selected experts. By such means, we can increase the number of experts to expand the model size for better performance, while keeping the computation complexity constant.

Despite the above benefit, given the potentially large number of experts, the memory capacity of a single device is often insufficient. As a result, expert parallelism (Lepikhin et al., 2021; Fedus et al., 2022) is a common technique to facilitate the distributed training of MoE models. As shown in Fig. 1, each device holds only a subset of the experts to reduce memory consumption. Meanwhile, other model parameters are replicated and stored on all devices, and the training data assigned to each device are different. In each MoE layer, based on the routing result of the gating network, each token is sent to the device where the selected expert is located. The output from the expert is then sent back to the original device of the corresponding token. This involves two communication operations, namely the All-to-All scatter and All-to-All gather (He et al., 2021), respectively.

Due to the dynamic nature of routing, training MoE models efficiently faces several challenges, with the All-to-All communication, being the most significant one. Particularly, the All-to-All communication time can account for up to 80% of the total training time (Hwang et al., 2023; Liu et al., 2023; He et al., 2022; Li et al., 2023; Yu et al., 2024). One reason is because all tokens need to participate in the All-to-All operation, leading to a high communication volume. Another reason is the communication frequency. Considering both the forward and backward propagation, each MoE layer requires four All-to-All communications per training iteration. Such frequent and extensive communication incurs significant time costs. Therefore, accelerating All-to-All communication is essential to improve training efficiency.

Motivation: Recent studies have demonstrated that expert routing exhibits a certain degree of *data locality*. To be specific, input tokens may have distinct preferences for experts, and the corresponding distribution is often skewed (He et al., 2022; Nie et al., 2023; Xue et al., 2024; Jiang et al., 2024). In other words, the All-to-All operation in MoE models can be highly unbalanced across different devices, thus bounded by the device pair with the highest communication volume. Meanwhile, it is well known that *network locality* is an inherent characteristic of modern clusters for deep learning training. In particular, there are various communication channels in modern clusters, e.g. intra-node devices usually communicate via PCIe or NVLINK, while inter-node devices use Ethernet or InfiniBand, with intra-node communication usually faster than inter-node ones. To achieve load balancing, existing methods propose techniques from the model perspective (Nie et al., 2023; Lewis et al., 2021). Typically, they either dynamically adjust the model placement but introduce a lot of additional communication, or modify the model definition but sacrifice the model performance (see §2.2 for more discussion). Yet optimization from the data perspective is under explored. Inspired by this, we propose NetMoE, which accelerates the All-to-All communication by combining the data locality in expert routing with the network locality among training devices. The essential idea of NetMoE is to dynamically adjust the placement of data samples during training based on expert routing results so that more tokens will be transmitted through high-speed channels rather than low-speed ones. As illustrated in Fig. 2, $sample_0$ shows a preference for $expert_2$ that resides on $node_1$, while $sample_3$ favors $expert_0$ that resides on $node_0$. Using the vanilla All-to-All communication method would result in significant inter-node communication overhead, as shown in Fig. 2(b). However, by swapping the positions of $sample_0$ and $sample_3$ as depicted in Fig. 2(c), part of the inter-node communication can be converted into intra-node communication or even intra-device memory copying, significantly reducing the time cost (detailed in §3.1). In this way, we can accelerate the All-to-All communication without affecting the computing results.

However, it is non-trivial to achieve dynamic sample placement. For one thing, how to adjust the placement to maximize efficiency is a complex and unexplored question. For another, since the adjustment should be done for every layer in every iteration, it is vital to devise an efficient algorithm to deduce the placement on the fly. To address these problems, we first revisit the cost modeling for All-to-All communication and formulate the dynamic sample placement problem into a combinatorial optimization problem. Subsequently, we split it into two stages to ease the solving and design a corresponding polynomial-time algorithm to ensure a timely solution.

In short, the technical contributions of this work are summarized as follows:

- We propose NetMoE, the first effort that leverages both the data locality and network locality to accelerate the All-to-All communication through dynamic sample placement.
- We formulate the dynamic sample placement problem as a combinatorial optimization problem, which aims to find the best sample placement that maximizes efficiency given the expert routing.
- We dissect the problem into two stages and develop a polynomial-time solution to efficiently derive the sample placement during training.
- We conduct experiments with various models on 32 NVIDIA A800 GPUs. Results show that NetMoE outperforms current MoE training systems by up to $1.67\times$ in terms of training efficiency.

2 PRELIMINARY

2.1 PARALLELISM IN DISTRIBUTED TRAINING

Data and Model Parallelism: In data parallelism (Li et al., 2020; Sergeev & Balso, 2018; Wang et al., 2023; Zhang et al., 2024b), each device maintains a complete copy of the model parameters, while different training samples are assigned to each device. After the backward computation is completed, the model gradients from all devices are aggregated before updating the model parameters. In model parallelism (Narayanan et al., 2021b; Huang et al., 2019; Narayanan et al., 2021a; Guan et al., 2024), model parameters are distributed across multiple devices, with each device responsible for only a portion of the model. Communication operations are necessary to transmit the intermediate results (a.k.a. forward activations and their backward gradients) to accomplish the forward and backward propagation.

Expert Parallelism: As shown in Fig. 1, expert parallelism (Lepikhin et al., 2021; Fedus et al., 2022) can be regarded as combining model parallelism and data parallelism. It distributes expert parameters across different devices like model parallelism, while replicating other parameters on all devices like data parallelism. In each MoE layer, each token will be routed by the gating network to top K different experts for processing, where K is a hyperparameter, typically a small value, such as 1 or 2, which helps to reduce the computational complexity. After the MoE layer obtains the gating routes, tokens are sent to the devices where the corresponding experts are located based on the routing. The results from the expert computations are then sent back to the original devices where the tokens are located. Since the experts are distributed across different devices, communication during this process involves all devices sending and receiving messages with one another, leading to what is known as All-to-All communication.

2.2 Distributed Training Acceleration Techniques for MoE Models

Dynamic Expert Placement: The efficiency of MoE models is constrained by the extensive and frequent All-to-All communication required during training. In response to this issue, some studies have observed that data tends to show a preference for certain experts during training. Then, based on this observation, they further propose to dynamically adjust the placement of experts to reduce communication volume (He et al., 2022; Nie et al., 2023; Zhai et al., 2023). For instance, popular experts can be placed on more devices in the data parallel manner, so that the communication volume related to them would decrease. However, due to the growing size of experts, these approaches incur substantial overhead of transmitting expert parameters among the devices, so they cannot adjust the expert placement for every iteration, leading to sub-optimality. In contrast, our work tries to reduce the communication volume from a different perspective: we dynamically adjust the placement of samples in every iteration to accelerate the All-to-All communication. To be specific, we formulate an optimization problem to deduce the best sample placement that minimizes the time cost of All-to-All communication. As we will evaluate in §4, our work outperforms existing works based on dynamic expert placement when training MoE models.

Modification in Model Definition: To achieve better workload balance in MoE training, there are many existing works developed to modify the model definition (e.g., routing mechanisms, model architectures). Some approaches modify the routing mechanism to balance the load across experts, which helps reduce synchronization time between devices (Lewis et al., 2021). Recognizing the network locality in distributed training, several works introduce a routing topology loss to prioritize routing tokens within the same node, thereby reducing inter-node communication (Li et al., 2024; Chen et al., 2022). Other approaches (Zeng & Xiong, 2023) map tokens to smaller hidden layer dimension before inter-node communication, further decreasing the communication load. SC-MoE (Cai et al., 2024) proposes feeding the output of the current attention layer directly into the next MoE layer, enabling parallel forward propagation with the current MLP layer in order to fully overlap All-to-All communication with computation. Although these methods improve training efficiency, they inevitably impact model convergence.

When applying these methods, we usually need to run numerous trials to tune the hyper-parameters, such as adjusting the weight of the topology-aware routing loss (Chen et al., 2022) or tuning the hyper-parameters for different communication channels (Zeng & Xiong, 2023). Given that each trial of LLM training can take days or even months, their utility is inevitably hampered. In contrast, our work focuses on how to accelerate All-to-All communication without affecting model convergence.

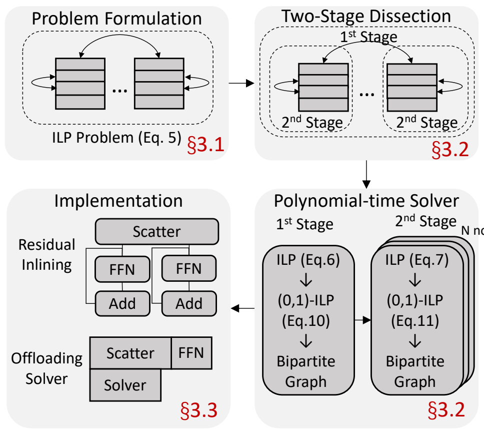


Figure 3: The overview of the method of NetMoE.

3 NetMoE

In this section, we introduce NetMoE, a novel framework designed to optimize distributed training for MoE models by considering both data and network locality. Given a target MoE model and the hardware environment, NetMoE aims to minimize the All-to-All communication cost. Its core innovation lies in optimizing the placement of samples within each MoE layer to maximize the utilization of faster intra-node bandwidth, thereby reducing the communication volume over slower inter-node connections. Specifically, NetMoE swaps the samples across devices during each MoE layer, enabling more tokens to communicate within the node during All-to-All communication.

Table 1: Notations used throughout this work. We assume I is divisible by J, and J is divisible by N, which are common in distributed training.

L	The number of tokens per sample.
H	The hidden size for each token.
E	The number of experts in the MoE layer.
K	The number of experts to be routed per token.
I	The number of samples per iteration (a.k.a. global batch size).
J	The number of devices (i.e., GPUs).
N	The number of nodes (machines).
$\mathbb{I}[\cdot]$	The indicator function.
$[\![n]\!]$	The set of natural numbers less than n, i.e., $\{0, 1, \cdots, n-1\}$.

Table 2: Bandwidth of each channel of the NVIDIA A800 GPU cluster used in our experiments.

Channel	Bandwidth
Intra-device	~2TB/s
Intra-node	400GB/s
Inter-node	100GB/s

Fig. 3 illustrates the overview of this section. We begin by introducing the modeling of All-to-All communication in MoE training and formulate our optimization problem in §3.1. We then illustrate how to solve the problem in §3.2, with the detailed algorithm shown in Alg. 1. We also present our implementation details in §3.3. For clarity, the frequently used notations are listed in Table 1.

3.1 Problem Formulation

Communication Modeling: We first discuss the mathematical modeling of All-to-All communication, which is the optimization target of NetMoE. We use the α-β model (Sarvotham et al., 2001) to analyze All-to-All communication, where α represents the latency cost and β represents the bandwidth cost. Specifically, we classify communication into three categories: intra-device, intra-node, and inter-node communication, each using different channels. Table 2 lists the bandwidth of each channel used in our experiments. Since intra-device communication is typically achieved via memory copying, it is significantly faster than the other two categories and thus not considered in our modeling. Therefore, the communication time is determined by the maximum time required for data transfer across the intra-node and inter-node channels. The bandwidths of these channels are represented by v_{intra}, and v_{inter}, respectively. Thus, for each All-to-All communication, its time cost can be expressed by the following formula, where $s_{\cdot}$ represents the communication volume for the corresponding channel.

$$t = \max(t_{intra}, t_{inter}), \quad \text{where} \quad \begin{aligned} t_{intra} &= \alpha_{intra} + \beta_{intra} s_{intra}, \beta_{intra} = 1/v_{intra}, \\ t_{inter} &= \alpha_{inter} + \beta_{inter} s_{inter}, \beta_{inter} = 1/v_{inter} \end{aligned} \tag{1}$$

The bandwidth ($v_{\cdot}$) and latency ($\alpha_{\cdot}$) can be obtained by profiling the hardware environment before training, while the communication volume ($s_{\cdot}$) needs to be dynamically determined based on the routing results within the MoE layer. We then analyze how to calculate the communication volume.

Let $route \in \mathbb{N}^{I \times L \times K}$ be the token routing results of the gating network, which represents the K experts that each token will be sent to. Then, the number of tokens that the i-th sample needs to send to the e-th expert can be counted as

$$num_{i,e} = \sum_{l,k} \mathbb{I}[route_{i,l,k} = e] \text{ for } i \in [\![I]\!], e \in [\![E]\!] \tag{2}$$

Next, $num \in \mathbb{N}^{I \times E}$ can be used to model the communication volume across different channels. Let $\texttt{ExpDev}(e)$ be the device index of the e-th expert, $\texttt{SmpDev}(i)$ the device index where the i-th sample should be routed to, and $\texttt{Node}(j)$ the node index of the j-th device. By considering the communication volume as the number of tokens that need to be transmitted, we have

$$s_{intra} = \sum_{(i,e) \in S_{intra}} num_{i,e}, \quad s_{inter} = \sum_{(i,e) \in S_{inter}} num_{i,e} \tag{3}$$

where S_{intra} and S_{inter} can be calculated via the device indices of experts and samples:

$$\begin{aligned} S_{intra} &= \{(i,e) | \texttt{Node}(\texttt{SmpDev}(i)) = \texttt{Node}(\texttt{ExpDev}(e)) \wedge \texttt{SmpDev}(i) \neq \texttt{ExpDev}(e)\} \\ S_{inter} &= \{(i,e) | \texttt{Node}(\texttt{SmpDev}(i)) \neq \texttt{Node}(\texttt{ExpDev}(e))\} \end{aligned} \tag{4}$$

Rationality of Dynamic Sample Placement: Given the aforementioned modeling, there is no doubt that the time cost of All-to-All communication is highly related to the placement of experts and samples. In practice, dynamically adjusting the placement does not affect the training results as the All-to-All communication is still correctly performed. Combining with the common fact of network locality that $v_{intra} > v_{inter}$, we can adjust the placement of samples and/or experts to reduce the inter-node communication volume, even if the intra-node communication volume becomes slightly higher. In fact, with a similar goal, several existing works have proposed to dynamically adjust the placement of experts based on their popularities (He et al., 2022; Nie et al., 2023; Zhai et al., 2023), as introduced in §2. However, all these works overlook the data locality — tokens of the same sample are usually routed to the same expert (Xue et al., 2024; Jiang et al., 2024), thereby missing the optimization opportunity of dynamic sample placement. More importantly, the size of the parameters of experts is usually much larger than the size of the samples. This prevents previous works from adjusting the expert placement in every iteration. In contrast, the adjustment of sample placement can be fused with the All-to-All communication by nature (detailed below), requiring zero extra communication. Consequently, this work focuses on the unexplored aspect, aiming to accelerate MoE training by dynamically adjusting the sample placement.[1]

To help readers better understand the strength of dynamic sample placement, we take Fig. 2 as an example, where $I = 4, L = 4, E = 4, K = 1$, and both the experts and samples are placed sequentially, i.e., $\texttt{ExpDev} = [0, 1, 2, 3], \texttt{SmpDev} = [0, 1, 2, 3]$. Fig. 2(b) shows the communication without changing the placement of samples. According to Eq. 3 and Eq. 4, if we only consider the sending volume of node 0, then $S_{inter} = \{(0, 2), (0, 3), (1, 2), (1, 3)\}$, indicating that $s_{inter} = 5$. However, after optimizing the placement of samples as in Fig. 2(c), i.e., $\texttt{SmpDev} = [3, 1, 2, 0]$, the corresponding inter-node communication volume changes into $S_{inter} = \{(3, 2), (3, 3), (1, 2), (1, 3)\}$, which gives $s_{inter} = 2$. Furthermore, it is worth noting that the sample placement adjustment can be combined with the All-to-All gather operation. To be specific, instead of restoring tokens to their original positions, they are directly placed in their new positions according to the altered sample placement. This method directly optimizes the current communication operation without introducing any extra communication.

Problem Formulation: After the sample placement adjustment is determined, it can be seen that altering SmpDev affects two All-to-All operations: the gather operation of the current MoE layer and the scatter operation of the next MoE layer. Thus, our optimization targets these two operations. For the l-th layer, the optimization problem can be written as follows.

$$\begin{aligned} \underset{\texttt{SmpDev}(i)\in[\![J]\!] \text{ for } i\in[\![I]\!]}{\arg\min} \quad & t^{(l,gather)} + t^{(l+1,scatter)} \\ & = \max\left(t_{intra}^{(l,gather)}, t_{inter}^{(l,gather)}\right) + \max\left(t_{intra}^{(l+1,scatter)}, t_{inter}^{(l+1,scatter)}\right) \\ \text{s.t.} \quad & \sum_{i\in[\![I]\!]} \mathbb{I}[\texttt{SmpDev}(i) = j] = I/J \text{ for } j \in [\![J]\!] \end{aligned} \tag{5}$$

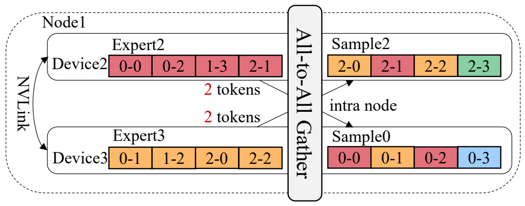
Since a single change in sample placement affects two All-to-All operations, both communication times are included in the optimization objective. Additionally, to ensure computational and memory balance across devices, each device should retain the same number of samples before and after the sample placement adjustment. This forms the basis for the constraints in our optimization.

3.2 Problem Solving

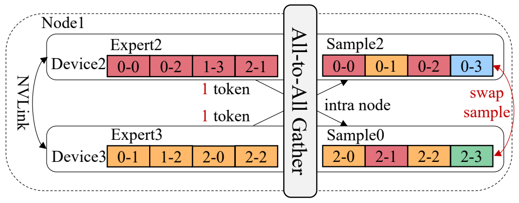
Eq. 5 is a complex combinatorial optimization problem, which cannot be solved optimally in polynomial time. As the cluster size increases, even finding an approximate solution may take a significant amount of time. Since this problem needs to be solved before each gather operation, solving it directly would result in unbearable additional time costs. To address this, we design an efficient method to obtain approximate solutions. In particular, we first dissect the optimization problem into two stages and develop a polynomial-time algorithm to achieve the solution, as introduced below.

Two-Stage Dissection: Although Eq. 1 takes the maximum value of the two kinds of communication cost, in practice, due to the significant bandwidth difference between the inter- and intra-node

[1]Our work is fully compatible with the dynamic expert placement technique. Specifically, in the problem formulation and solving of NetMoE, we do not make any assumption on the expert placement. Instead, it is treated as an input. Thus, we can dynamically adjust the expert placement like previous works, and NetMoE can still deduce the optimal sample placement. We would like to leave the combination as our future work.



(a) Sample placement after the first stage.



(b) Sample placement after the second stage.

Figure 4: An example of the second stage optimization. Fig. 4(a) shows the MoE layer in $Node_1$ after the first stage optimization in Fig. 2(c). By applying the second stage optimization within the node, the intra-node communication can be reduced by 1 token (by swapping $sample_0$ and $sample_2$), as shown in Fig. 4(b).

connections, the most time-consuming term is usually the inter-node one. Therefore, we propose a two-stage solving strategy: the first stage optimizes t_{inter} at the global scale, while the second stage minimizes t_{intra} within each node, without affecting t_{inter}. Formally, suppose there are N nodes and each node consists of J/N devices, then the optimization formula of the first stage can be written as the following integer linear programming (ILP) problem:

$$\begin{aligned} \mathop{\arg\min}_{\texttt{Node}(\texttt{SmpDev}(i))\in[\![N]\!] \text{ for } i\in[\![I]\!]} \quad & t_{inter}^{(l,gather)} + t_{inter}^{(l+1,scatter)} \\ \text{s.t.} \quad & \sum_{i\in[\![I]\!]} \mathbb{I}[\texttt{Node}(\texttt{SmpDev}(i)) = n] = I/N \text{ for } n \in [\![N]\!] \end{aligned} \tag{6}$$

The constraint of balance across devices in Eq. 5 is turned into the balance across nodes since we focus on inter-node communication in the first stage. After obtaining the optimal solution of the first stage, the second stage considers rearranging the samples within each node individually. For the n-th node, denote $[\![I]\!]_n^* \subseteq [\![I]\!]$ as the set of samples appointed to it after solving Eq. 6. And let $[\![J]\!]_n = \{j|j \in [\![J]\!] \wedge \texttt{Node}(j) = n\}$ be the set of experts reside on it ($[\![J]\!]_n$ is determined by the device placement rather than obtained by Eq. 6). Then, to optimize for the n-th node, we should solve the following ILP problem:

$$\mathop{\arg\min}_{\texttt{SmpDev}(i)\in[\![J]\!]_n \text{ for } i\in[\![I]\!]_n^*} \quad t_{intra}^{(l,gather)} + t_{intra}^{(l+1,scatter)} \quad \text{s.t.} \sum_{i\in[\![I]\!]_n^*} \mathbb{I}[\texttt{SmpDev}(i) = j] = I/J \text{ for } j \in [\![J]\!]_n \tag{7}$$

Specifically, Fig. 2 can be regarded as the optimization of the first stage, while Fig. 4 demonstrates the second stage of optimization built upon it. Although the second stage consists of N ILP problems, each for one node, they are independent and can be solved concurrently.

Polynomial-time Solver: By dissecting the original combinatorial optimization problem, we obtain $N+1$ ILP problems, which can be solved via existing libraries like PuLP (Mitchell et al., 2011). However, recall that we need to solve these problems for each layer in each training iteration, the efficiency of problem-solving is vital. Unfortunately, since ILP problems are NP-hard, when we try to solve them via PuLP, the time cost of solving exceeds the time cost of scatter communication and experts' computation (as evaluated in §4.4), making it impractical. Given the fact that each sample must be assigned to one device, we reconsider the ILP problems as assignment problems by transforming them into weighted bipartite matching problems, and subsequently develop a polynomial-time solver based on the widely used Kuhn-Munkres (KM) algorithm.

We first introduce how to transform the ILP problems into weighted bipartite matching problems. Let $c_{i,n}$ and $c'_{i,j}$ represent the inter- and intra-node communication volume when placing the i-th sample on the j-th device in the n-th node, They can be calculated using the following formulas:

$$\begin{aligned} & c_{i,n} = \sum_{e\in S} num_{i,e}, \quad c'_{i,j} = \sum_{e\in S'} num_{i,e}, \quad \text{where} \\ & S = \{e|\texttt{Node}(\texttt{ExpDev}(e)) \neq n\}, S' = \{e|\texttt{Node}(\texttt{ExpDev}(e)) = \texttt{Node}(j) \wedge \texttt{ExpDev}(e) \neq j\}. \end{aligned} \tag{8}$$

To make the expression clearer, let $p_{i,n}, p'_{i,j} \in \{0,1\}$ indicate whether the i-th sample is placed on the n-th node and the j-th device, respectively. Then, the optimization objective can be expressed as

$$\begin{aligned} & t_{inter} = \alpha_{inter} + \beta_{inter} \sum_{i\in[\![I]\!], n\in[\![N]\!]} c_{i,n} p_{i,n}, \quad t_{intra} = \alpha_{intra} + \beta_{intra} \sum_{i\in[\![I]\!], j\in[\![J]\!]} c'_{i,j} p'_{i,j}, \quad \text{where} \\ & p_{i,n} = \mathbb{I}[\texttt{Node}(\texttt{SmpDev}(i)) = n], \quad p'_{i,j} = \mathbb{I}[\texttt{SmpDev}(i) = j] \quad \text{for } i \in [\![I]\!], n \in [\![N]\!], j \in [\![J]\!] \end{aligned} \tag{9}$$

Algorithm 1 NetMoE Optimization

```
1: function Solve($num$)
2:     Get $c, c'$ via Eq. 8 and build bipartite graphs
3:     Get the optimal solution $p^*$ via the Kuhn-Munkres (KM) algorithm
4:     return the optimal sample placement according to $p^*$
5: The Main Training Process:
6: for submodule in model do
7:     if submodule is a MoE layer then
8:         Get $route$ from the gating network and calculate $num$ via Eq. 2
9:         Invoke Solve($num$) in a background thread        ▷ Offloading solving process
10:        Get $input$ from All-to-All scatter
11:        Get $output$ from expert computation
12:        $output = input + output$        ▷ Expert residual inlining
13:        Get the optimal sample placement from the background thread
14:        Perform All-to-All gather with the optimal sample placement
15:    else
16:        submodule.forward()
```

After modifying the corresponding constraints, we transform the ILP problems into (0,1)-ILP problems. For instance, below presents the transformed problem for the first stage[2] (i.e., Eq. 6):

$$\begin{aligned} \underset{p_{i,n} \text{ for } i\in[\![I]\!],n\in[\![N]\!]}{\arg\min} \quad & \alpha_{inter} + \beta_{inter} \sum_{i\in[\![I]\!],n\in[\![N]\!]} \left(c_{i,n}^{(l,gather)} + c_{i,n}^{(l+1,scatter)}\right) p_{i,n} \\ \text{s.t.} \quad & \sum_{i\in[\![I]\!],n\in[\![N]\!]} p_{i,n} = I/N \text{ for } n \in [\![N]\!], \quad \sum_{n\in[\![N]\!]} p_{i,n} = 1 \text{ for } i \in [\![I]\!] \end{aligned} \tag{10}$$

This (0,1)-ILP problem can be modeled as a weighted bipartite matching problem. In particular, consider a bipartite graph with two sets of graph nodes, P and Q. The set P represents all training samples, and $|P| = I$. The set Q represents all training nodes (machines), where each training node can handle $B := I/N$ training samples. To model this, each graph node in Q is duplicated B times, resulting in $|Q| = I$. A weighted edge exists between every pair of graph nodes from P and Q. Let P_i represent the i-th training sample and Q_n the $\lfloor n/B \rfloor$-th training node. The weight of the edge between P_i and Q_n is denoted as $W_{i,n} = c_{i,\lfloor n/B \rfloor}^{(l,gather)} + c_{i,\lfloor n/B \rfloor}^{(l+1,scatter)}$. This transformation reduces the problem of finding a minimum weight perfect matching in this bipartite graph, which can be efficiently solved to optimality in polynomial time using the Kuhn-Munkres (KM) algorithm. Fig. 5 illustrates an example of constructing a bipartite graph during the first stage in Fig. 2. The graph nodes on the left represent set P, and the graph nodes on the right represent set Q . Each pair of graph nodes is connected by a weighted edge, depicted by a dotted line. The red edges indicate the final matching scheme, where the total weight of all matched edges is minimized.

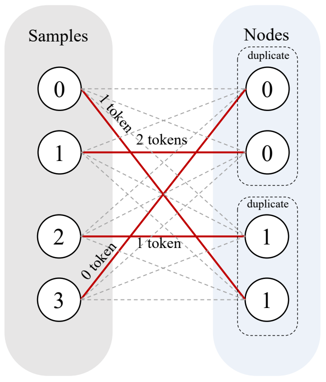


Figure 5: An example of a bipartite graph.

3.3 Implementation

NetMoE is implemented on top of PyTorch (Paszke et al., 2019), with custom operations (e.g., the calculation of num, c, c', and the KM algorithm) implemented in C++ and CUDA. The complete workflow of NetMoE is presented in Alg. 1. In addition to the problem-solving introduced in §3.2, NetMoE has been optimized in the following ways.

Expert Residual Inlining: In classic MoE models, residual connections are independent of the MoE layers. However, in NetMoE, the position of the training data changes after the All-to-All

[2]The problems of the second stage (Eq. 7) can also be transformed into (0,1)-ILP problems and solved in polynomial time similarly. We omit them here due to the space constraint and only discuss the first stage.

Table 3: Configurations of the evaluated models.

Model Name	Base	$\frac{I}{J}$	S	H	$\frac{E}{J}$	K
MoE-GPT-S	GPT-2	4	1024	768	2	2
MoE-GPT-M	GPT-2	4	1024	1024	2	2
MoE-GPT-L	GPT-2	4	1024	1280	2	2
MoE-GPT-XL	GPT-2	4	1024	1600	2	2
MoE-GPT-XXL	GPT-3	4	1024	4096	2	2

Table 4: Time cost of different solvers vs. the summed time cost of All-to-All scatter and expert computation in milliseconds (MoE-GPT-S, $J = 16$).

$\frac{I}{J}$	KM	PuLP	Scatter + Computation
2	0.08	42.8	3.69 (2.63 + 1.06)
4	0.48	50.1	7.13 (5.34 + 1.79)
8	1.48	72.9	13.50 (10.31 + 3.19)
16	10.82	143.7	27.31 (21.49 + 5.82)
24	31.09	266.5	41.65 (33.82 + 7.83)

gather operation, while the samples in the residual connections remain in their original positions. To ensure the correctness of the model, we inline the residual connections into the expert computation, as shown in line 12 of Alg. 1. This optimization ensures consistency in model accuracy before and after applying the algorithm. More details about inlining is elaborated in Appendix A.1.

Offloading Solving Process: The KM algorithm is hard to parallelize, making it unsuitable for highly parallelized accelerators like GPUs, so we perform the solving process on the CPU. As shown in line 9 of Alg. 1, after obtaining the routing results for the current layer, each device calculates and transfers num to the CPU memory. The routing results for the next layer, required by the optimization algorithm, can be predicted by directly passing the current layer's input to the router of the next layer (Eliseev & Mazur, 2023; Tang et al., 2024). The solving process only needs to provide the new sample positions before the All-to-All gather operation. In this way, the solving process can be overlapped with the All-to-All scatter and expert computation. As we will show in §4.4, the solving time is fully hidden and thus introduces zero overhead. More discussion of algorithm selection and overlap potential is described in Appendix A.2.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUPS

We compare NetMoE with state-of-the-art methods based on dynamic expert placement, including FasterMoE (He et al., 2022) and SmartMoE (Zhai et al., 2023). We also included FastMoE (He et al., 2021) to represent a baseline without adjusting the placement of experts or samples. All experiments are conducted on a cluster consisting of 4 nodes, each equipped with 8 NVIDIA A800-SXM4-40GB GPUs. As listed in Table 2, the GPUs within each node are connected via NVLink with a 400 GB/s bandwidth, while the nodes are interconnected via InfiniBand with a 100 GB/s bandwidth. The configurations of the evaluated models are listed in Table 3. We select the GPT model architecture (Radford et al., 2019; Brown et al., 2020) as the backbone and replace all FFN layers in each model with MoE layers. In particular, since SmartMoE requires at least 2 experts on each device, we set the number of experts as $E = 2 \times J$, where J is the number of GPUs in the corresponding experiment, and we fix the number of selected experts for each token as $K = 2$. By default, we utilize 8 GPUs per node to carry out the experiments, and we present the results for scenarios with fewer GPUs per node in Appendix B. All results are averaged over 50 iterations.

4.2 END TO END PERFORMANCE

As shown in Fig. 6, NetMoE demonstrates up to a $1.67\times$ speedup over FastMoE, a $1.37\times$ speedup over FasterMoE, and a $1.33\times$ speedup over SmartMoE. FasterMoE achieves significant optimization by overlapping expert computation and supporting dynamic expert placement. However, as the model's hidden dimension increases, the cost of communicating with experts rises, making it difficult for it to maintain the same level of acceleration. This leads to a performance gap between FasterMoE and NetMoE. On the other hand, SmartMoE outperforms FasterMoE, which is expected since SmartMoE adjusts expert placement to ensure load balancing on top of FasterMoE's optimizations. However, SmartMoE primarily focuses on balancing the computational load, without emphasizing communication efficiency. When communication becomes the primary bottleneck, the benefits of load balancing are less pronounced. Consequently, by dynamically adjusting the sample placement, NetMoE consistently outperforms the state-of-the-art systems. Last but not least,

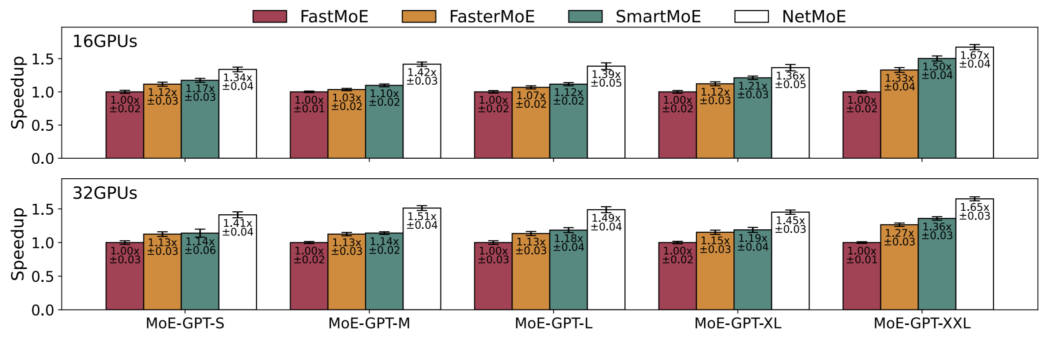


Figure 6: End-to-end speedup (mean and standard deviation) of different methods.

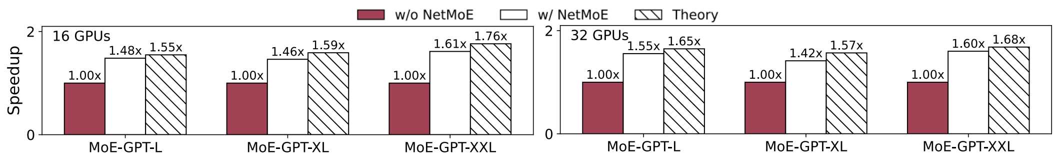


Figure 7: The actual and theoretic speedup in terms of All-to-All communication cost.

it is noteworthy that our method is compatible with dynamic expert placement. By adjusting the $\texttt{ExpDev}(\cdot)$ that is fed to our solver, NetMoE can be combined with dynamic expert placement to achieve even higher efficiency. We plan to explore this integration in our future work.

4.3 All-to-All Performance

As shown in Fig. 7, we conducted experiments on three kinds of model to observe the differences in All-to-All communication before and after applying NetMoE and compared these results with the theoretical optimization values provided by the solver. It can be seen that the actual speedup in All-to-All communication is slightly less than the theoretical values. This discrepancy is reasonable, as our modeling of All-to-All communication assumes ideal conditions and does not account for potential routing conflicts or hardware-induced errors. In Appendix C, we have provided more experimental results to analyze the acceleration of All-to-All communication.

4.4 Solver Performance

To verify the efficiency of the solver, we compared the solving time under different scales with the summed time cost of All-to-All scatter and expert computation, as shown in Table 4. KM represents the algorithm used in NetMoE, while PuLP (Mitchell et al., 2011) refers to the commonly used toolkit for solving linear programming problems. It can be observed that although the solving time exhibits super-linear growth with the increase in I, the solving process is consistently hidden by the All-to-All scatter and expert computation for various scenarios. In contrast, PuLP's solving time is difficult to get overlapped. This highlights the necessity of designing specialized optimization methods in scenarios with high real-time performance demands.

5 Conclusion

We proposed NetMoE to optimize All-to-All communication, which is the primary bottleneck in training MoE models. By leveraging data and network locality, our method dynamically adjusts the placement of training samples during training, transforming inter-node communication into intra-node communication to enhance All-to-All communication efficiency. We modeled the All-to-All communication time and the sample placement as an optimization problem and designed a polynomial-time approach to solve it. Empirical results demonstrate that NetMoE outperforms existing MoE training systems by up to $1.67\times$ in terms of training efficiency.

ACKNOWLEDGEMENT

This work is supported by National Science and Technology Major Project (2022ZD0116315), National Natural Science Foundation of China (U22B2037, U23B2048, 62402011), Beijing Municipal Science and Technology Project (Z231100010323002), China National Postdoctoral Program for Innovative Talents (BX20230012), China Postdoctoral Science Foundation (2024M750103), Beijing Natural Science Foundation (4244080), the Fund of Kunpeng and Ascend Center of Excellence (Peking University), and High-performance Computing Platform of Peking University. Bin Cui is the corresponding author.

REFERENCES

Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H. Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan M. Cohen, Sirshak Das, Ayush Dattagupta, Olivier Delalleau, Leon Derczynski, Yi Dong, Daniel Egert, Ellie Evans, Aleksander Ficek, Denys Fridman, Shaona Ghosh, Boris Ginsburg, Igor Gitman, Tomasz Grzegorzek, Robert Hero, Jining Huang, Vibhu Jawa, Joseph Jennings, Aastha Jhunjhunwala, John Kamalu, Sadaf Khan, Oleksii Kuchaiev, Patrick LeGresley, Hui Li, Jiwei Liu, Zihan Liu, Eileen Long, Ameya Sunil Mahabaleshwarkar, Somshubra Majumdar, James Maki, Miguel Martinez, Maer Rodrigues de Melo, Ivan Moshkov, Deepak Narayanan, Sean Narenthiran, Jesus Navarro, Phong Nguyen, Osvald Nitski, Vahid Noroozi, Guruprasad Nutheti, Christopher Parisien, Jupinder Parmar, Mostofa Patwary, Krzysztof Pawelec, Wei Ping, Shrimai Prabhumoye, Rajarshi Roy, Trisha Saar, Vasanth Rao Naik Sabavat, Sanjeev Satheesh, Jane Polak Scowcroft, Jason Sewall, Pavel Shamis, Gerald Shen, Mohammad Shoeybi, Dave Sizer, Misha Smelyanskiy, Felipe Soares, Makesh Narsimhan Sreedhar, Dan Su, Sandeep Subramanian, Shengyang Sun, Shubham Toshniwal, Hao Wang, Zhilin Wang, Jiaxuan You, Jiaqi Zeng, Jimmy Zhang, Jing Zhang, Vivienne Zhang, Yian Zhang, and Chen Zhu. Nemotron-4 340b technical report. *CoRR*, abs/2406.11704, 2024.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Weilin Cai, Juyong Jiang, Le Qin, Junwei Cui, Sunghun Kim, and Jiayi Huang. Shortcut-connected expert parallelism for accelerating mixture-of-experts. *CoRR*, abs/2404.05019, 2024.

Chang Chen, Min Li, Zhihua Wu, Dianhai Yu, and Chao Yang. Ta-moe: Topology-aware large scale mixture-of-expert training. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 1280–1297, 2024.

Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61 (1):1:1–1:23, 2014.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris

Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading. *CoRR*, abs/2312.17238, 2023.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23:120:1–120:39, 2022.

Lei Guan, Dong-Sheng Li, Jiye Liang, Wen-Jian Wang, Ke-shi Ge, and Xicheng Lu. Advances of pipeline model parallelism for deep learning training: An overview. *J. Comput. Sci. Technol.*, 39 (3):567–584, 2024.

Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fastmoe: A fast mixture-of-expert training system. *CoRR*, abs/2103.13262, 2021.

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 *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*, pp. 120–134, 2022.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 103–112, 2019.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, HoYuen Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. In *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023*, 2023.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *CoRR*, abs/2401.04088, 2024.

Yoohwan Kwon and Soo-Whan Chung. Mole : Mixture of language experts for multi-lingual automatic speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2023, Rhodes Island, Greece, June 4-10, 2023*, pp. 1–5, 2023.

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 *The ninth International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

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, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139, pp. 6265–6274, 2021.

Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed moe training and inference with lina. In *Proceedings of the 2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, pp. 945–959, 2023.

Jing Li, Zhijie Sun, Xuan He, Li Zeng, Yi Lin, Entong Li, Binfan Zheng, Rongqian Zhao, and Xin Chen. Locmoe: A low-overhead moe for large language model training. *CoRR*, abs/2401.13920, 2024.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, 2020.

Hanxue Liang, Zhiwen Fan, Rishov Sarkar, Ziyu Jiang, Tianlong Chen, Kai Zou, Yu Cheng, Cong Hao, and Zhangyang Wang. M^{3}vit: Mixture-of-experts vision transformer for efficient multi-task learning with model-accelerator co-design. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

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, ACM SIGCOMM 2023, New York, NY, USA, 10-14 September 2023*, pp. 486–498, 2023.

Stuart Mitchell, Michael OSullivan, and Iain Dunning. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand*, 65:25, 2011.

Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel DNN training. In *International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7937–7947, 2021a.

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. Efficient large-scale language model training on GPU clusters using megatron-lm. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, pp. 58, 2021b.

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. *Proc. ACM Manag. Data*, 1(1):110:1–110:19, 2023.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

James B. Orlin and Ravindra K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Math. Program.*, 54:41–56, 1992.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035, 2019.

Pytorch. Gradient accumulation pytorch. `https://gist.github.com/thomwolf/ac7a7da6b1888c2eeac8ac8b9b05d3d3`, 2019.

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.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 8583–8595, 2021.

Shriram Sarvotham, Rudolf H. Riedi, and Richard G. Baraniuk. Connection-level analysis and modeling of network traffic. In *Proceedings of the 1st ACM SIGCOMM Internet Measurement Workshop, IMW 2001, San Francisco, California, USA, November 1-2, 2001*, pp. 99–103, 2001.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100, 2022.

Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018.

Yunfan Shao, Zhichao Geng, Yitao Liu, Junqi Dai, Hang Yan, Fei Yang, Li Zhe, Hujun Bao, and Xipeng Qiu. CPT: a pre-trained unbalanced transformer for both chinese language understanding and generation. *Sci. China Inf. Sci.*, 67(5), 2024.

Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. Progressive layered extraction (PLE): A novel multi-task learning (MTL) model for personalized recommendations. In *RecSys 2020: Fourteenth ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22-26, 2020*, pp. 269–278, 2020.

Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. HOBBIT: A mixed precision expert offloading system for fast moe inference. *CoRR*, abs/2411.01433, 2024.

Tensorflow. Gradient accumulation tensorflow. `https://github.com/tensorflow/tensorflow/pull/32576`, 2019.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023.

Guozheng Wang, Yongmei Lei, Zeyu Zhang, and Cunlu Peng. A communication efficient admm-based distributed algorithm using two-dimensional torus grouping allreduce. *Data Sci. Eng.*, 8 (1):61–72, 2023.

Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. Openmoe: An early effort on open mixture-of-experts language models. *CoRR*, abs/2402.01739, 2024.

Zhao You, Shulin Feng, Dan Su, and Dong Yu. Speechmoe2: Mixture-of-experts model with improved routing. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022, Virtual and Singapore, 23-27 May 2022*, pp. 7217–7221, 2022.

Dianhai Yu, Liang Shen, Hongxiang Hao, Weibao Gong, HuaChao Wu, Jiang Bian, Lirong Dai, and Haoyi Xiong. Moesys: A distributed and efficient mixture-of-experts training and inference system for internet services. *IEEE Trans. Serv. Comput.*, 17(5):2626–2639, 2024.

Zhiyuan Zeng and Deyi Xiong. Scomoe: Efficient mixtures of experts with structured communication. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

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 *Proceedings of the 2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, pp. 961–975, 2023.

Huangzhao Zhang, Kechi Zhang, Zhuo Li, Jia Li, Jia Li, Yongmin Li, Yunfei Zhao, Yuqi Zhu, Fang Liu, Ge Li, et al. Deep learning for code generation: a survey. *Sci. China Inf. Sci.*, 67(9), 2024a.

Zhen-Xing Zhang, Yuan-Bo Wen, Han-Qi Lv, Chang Liu, Rui Zhang, Xia-Qing Li, Chao Wang, Zi-Dong Du, Qi Guo, Ling Li, Xue-Hai Zhou, and Yun-Ji Chen. Ai computing systems for llms training: a review. *J. Comput. Sci. Technol.*, 2024b.

Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. DB-GPT: large language model meets database. *Data Sci. Eng.*, 9(1):102–111, 2024.

Xinyu Zou, Zhi Hu, Yiming Zhao, Xuchu Ding, Zhongyi Liu, Chenliang Li, and Aixin Sun. Automatic expert selection for multi-scenario and multi-task search. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2022, Madrid, Spain, July 11 - 15, 2022*, pp. 1535–1544, 2022.

A More Details of Implementation

A.1 Detail of Expert Residual Inlining

As shown in Fig. 8, the original residual addition method adds the attention output to the result obtained from the gather operation. In NetMoE, however, it is added after the scatter operation but before the gather operation. Such an inlining facilitates the adjustment of sample placement, and meanwhile ensures the correctness of computation.

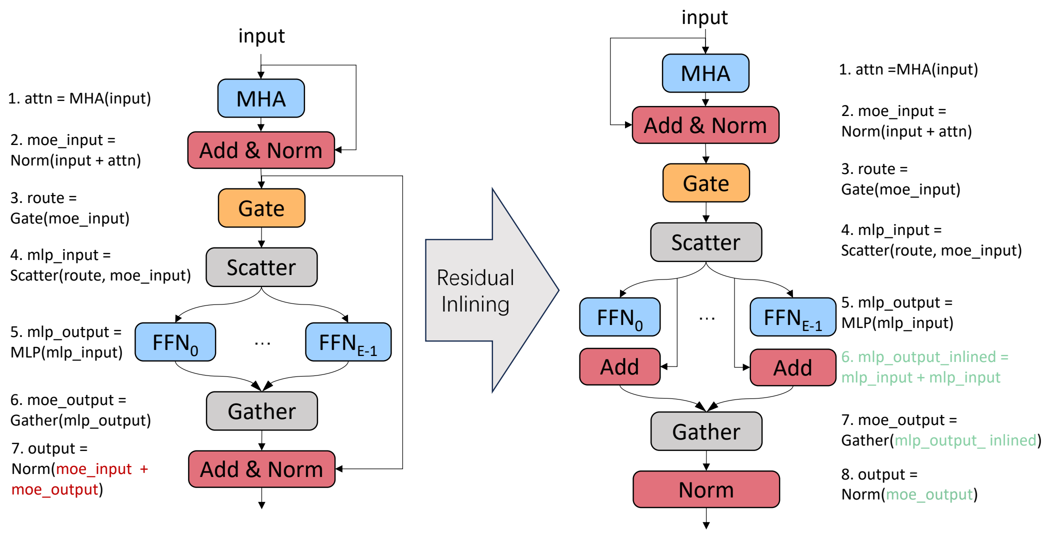

input
1. attn = MHA(input)
MHA
2. moe_input = Norm(input + attn)
Add & Norm
3. route = Gate(moe_input)
Gate
4. mlp_input = Scatter(route, moe_input)
Scatter
5. mlp_output = MLP(mlp_input)
FFN0
FFNE-1
6. moe_output = Gather(mlp_output)
Gather
7. output = Norm(moe_input + moe_output)
Add & Norm
Residual Inlining
input
MHA
1. attn =MHA(input)
Add & Norm
2. moe_input = Norm(input + attn)
Gate
3. route = Gate(moe_input)
Scatter
4. mlp_input = Scatter(route, moe_input)
FFN0
FFNE-1
5. mlp_output = MLP(mlp_input)
Add
Add
6. mlp_output_inlined = mlp_input + mlp_input
Gather
7. moe_output = Gather(mlp_output_ inlined)
Norm
8. output = Norm(moe_output)

Figure 8: Illustration of the Transformer layer with and without the expert residual inlining.

A.2 Discussion of Algorithm Selection and Overlap Potential

Our design adopts the KM algorithm based on two practical factors: (1) Although the time complexity of the KM algorithm is $\mathcal{O}(I^3)$, the current training process commonly employs gradient accumulation (Tensorflow, 2019; Pytorch, 2019) due to the limited GPU memory. Thus, the value of I is typically confined to an acceptable size, ensuring that the solving time can be effectively overlapped; (2) The algorithm's runtime is fully overlapped with communication phases, rendering further acceleration unnecessary for hiding the overhead of solver. While faster approximate solvers exist (Orlin & Ahuja, 1992; Duan & Pettie, 2014), their benefits would be marginal in current training configurations where computation-communication overlap already masks the optimization time.

B End To End Performance with Fewer GPUs Per Node

Fig. 9 illustrates the end-to-end speedup for configurations with 2 GPUs or 4 GPUs per node. The results demonstrate that NetMoE achieves the best performance across various experimental settings, which are consistent with the results obtained when there are 8 GPUs per node (as demonstrated in Fig. 6).

It is worth noting that standard server configurations typically accommodate up to 8 NVIDIA GPUs per node. Thus, 8 GPUs per node represent a standard setup for distributed training of large language models Dubey et al. (2024); Adler et al. (2024); Dai et al. (2024); Scao et al. (2022). Although superpods like the NVIDIA GB200 NVL72 support high-speed connections (e.g., NVLink) among more than 8 GPUs, they rely on custom hardware and are prohibitively expensive. Training scenarios on superpods are rare and significantly differ from the typical scenarios in GPU clusters or clouds. Therefore, this paper opts for experiments with configurations of up to 8 GPUs per node.

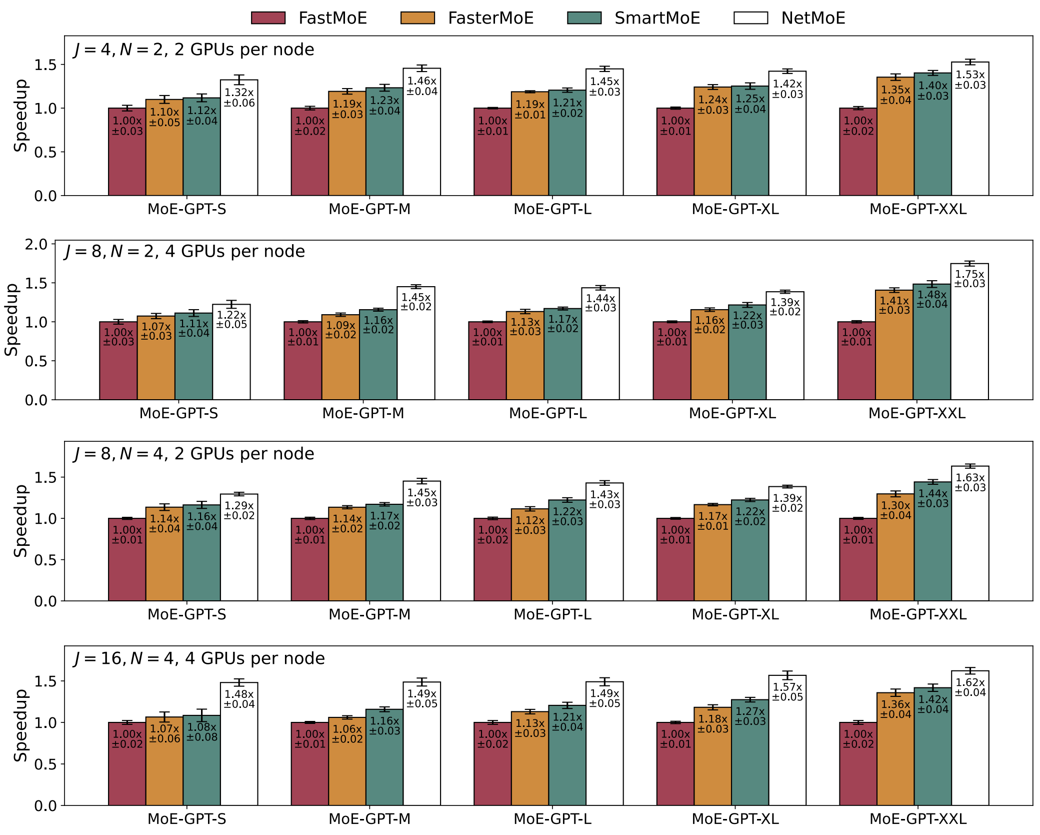


Figure 9: End-to-end speedup (mean and standard deviation) of different numbers of total devices (denoted as J) and numbers of nodes (denoted as N).

C Detailed Analysis of All-to-All Communication Optimization

To gain deeper insight into the source of NetMoE's optimization, we assess two kinds of statistics:

- The proportions of training samples that are exchanged across nodes or across devices by NetMoE, respectively. A higher proportion indicates more samples are adjusted across nodes/devices.
- The intra-node and inter-node communication volumes before and after applying NetMoE.

Firstly, Table 5 summarizes the mean and standard deviation across all iterations. After applying NetMoE, a great proportion of training samples are exchanged across nodes, leading to the reduction in the inter-node communication volume. It is noteworthy that although the intra-node communication volume accounts for a large proportion (i.e., s_{intra} or $\frac{s_{intra}}{s_{intra}+s_{inter}}$ increases) after applying NetMoE, it will not become the performance bottleneck since the inter-node communication bandwidth is much lower. As a result, the All-to-All communication can be accelerated due to the reduction in inter-node communication volume brought by sample placement adjustment.

Secondly, since the routing result dynamically changes during the training of MoE models, to discover the impact of routing distribution, in Fig. 10 we plot (1) the reduction in inter-node communication, and (2) the proportion of samples exchanged across nodes, across different iterations. Meanwhile, we follow prior works He et al. (2022); Nie et al. (2023) to record the distribution of expert selection across different iterations in order to describe the routing distribution. It can be observed that the routing distribution changes during the model training process. However, NetMoE consistently reduces the inter-node communication by adjusting the sample placement given the dynamic distributions. Consequently, the effectiveness of NetMoE is robust to the routing distribution.

Table 5: Summary of communication volume and proportion of sequence adjustment. For communication volume, we provide the intra-node and inter-node communication volumes before and after applying NetMoE, with the increase or reduction given in parentheses. For the proportion of sequence adjustment, "Across Nodes" indicates the proportion of sequences that are exchanged across nodes, and "All" indicates the proportion of all sequences that are adjusted.

(a) 2 nodes, 16 GPUs

	Communication Volume (MB)				Proportion of Sequence Adjustment (%)	
	w/o NetMoE		w/ NetMoE		Across Nodes	All
	s_{intra}	s_{inter}	s_{intra}	s_{inter}		
MoE-GPT-S	168.45 ± 5.43	191.07 ± 5.43	162.24 ± 11.69 (↓ 3.69%)	116.37 ± 11.69 (↓ 39.10%)	43.663 ± 3.560	91.394 ± 1.319
MoE-GPT-M	222.89 ± 5.72	258.20 ± 5.72	214.31 ± 7.42 (↓ 3.85%)	147.33 ± 7.42 (↓ 42.94%)	44.455 ± 2.122	91.929 ± 1.163
MoE-GPT-L	281.81 ± 5.18	318.56 ± 5.18	236.53 ± 8.81 (↓ 16.07%)	208.69 ± 8.81 (↓ 34.49%)	42.801 ± 2.232	91.799 ± 1.080
MoE-GPT-XL	347.44 ± 5.68	402.06 ± 5.68	313.00 ± 8.30 (↓ 9.91%)	256.94 ± 8.30 (↓ 36.10%)	43.619 ± 2.267	91.681 ± 1.369
MoE-GPT-XXL	922.40 ± 4.16	989.60 ± 4.16	872.00 ± 8.29 (↓ 5.46%)	570.40 ± 8.29 (↓ 42.36%)	45.688 ± 3.006	92.469 ± 1.294

(b) 4 nodes, 32 GPUs

	Communication Volume (MB)				Proportion of Sequence Adjustment (%)	
	w/o NetMoE		w/ NetMoE		Across Nodes	All
	s_{intra}	s_{inter}	s_{intra}	s_{inter}		
MoE-GPT-S	167.07 ± 7.52	575.88 ± 7.52	219.21 ± 10.30 (↑ 31.21%)	351.15 ± 10.30 (↓ 39.02%)	72.427 ± 1.361	96.427 ± 0.544
MoE-GPT-M	224.24 ± 6.82	766.87 ± 6.82	288.58 ± 13.00 (↑ 28.70%)	492.44 ± 13.00 (↓ 35.79%)	72.340 ± 1.094	96.122 ± 0.461
MoE-GPT-L	280.56 ± 6.74	958.44 ± 6.74	376.66 ± 10.47 (↑ 34.25%)	591.72 ± 10.47 (↓ 38.26%)	72.693 ± 1.249	96.292 ± 0.590
MoE-GPT-XL	350.62 ± 7.10	1199.12 ± 7.10	423.37 ± 11.69 (↑ 20.75%)	791.19 ± 11.69 (↓ 34.02%)	72.217 ± 1.499	96.159 ± 0.569
MoE-GPT-XXL	884.80 ± 6.13	3080.00 ± 6.13	1201.60 ± 8.85 (↑ 35.81%)	1884.00 ± 8.85 (↓ 38.83%)	72.305 ± 1.886	96.391 ± 0.661

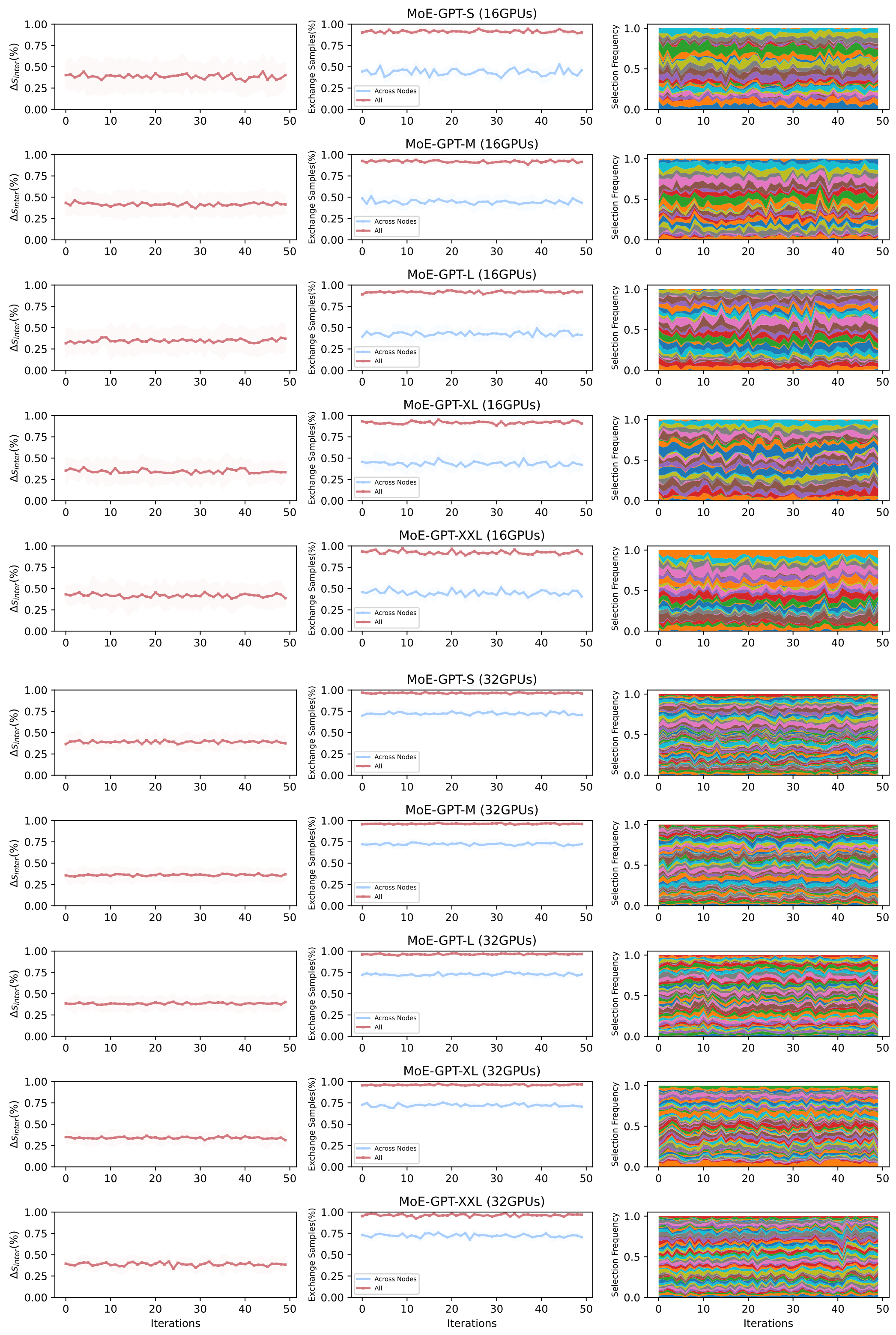


Figure 10: Left: The reduction in inter-node communication volume. Middle: The proportion of samples exchanged across nodes. Right: The distribution of expert selection (layer 0).