Characterizing Communication in Distributed Parameter-Efficient Fine-Tuning for Large Language Models

Nawras Alnaasan*, Horng-Ruey Huang*, Aamir Shafi, Hari Subramoni, Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering, The Ohio State University, Columbus, Ohio, USA
{alnaasan.1, huang.5282, shafi.16, subramoni.1, panda.2}@osu.edu

***Abstract*—Parameter-efficient Fine-tuning (PEFT) methods have emerged as powerful techniques for adapting pre-trained Large Language Models (LLMs) to specific tasks with reduced computational and memory overhead. However, despite their promising potential, there remains a gap in understanding how these methods perform in distributed computing settings. In this paper, we present a comprehensive characterization of the communication dynamics in distributed PEFT for LLMs. Our study emphasizes the crucial role of communication efficiency in the performance and scalability of PEFT methods when utilized across GPU clusters. Through systematic analysis of various PEFT techniques and LLM sizes, we have illustrated how communication overhead can significantly impact throughput, training time, and overall model performance. Our findings indicate that PEFT methods inherently reduce communication and computational burdens compared to full fine-tuning. These improvements yield up to 1.75x speedup by using PEFT methods such as Low-Rank Adaptation (LoRA) to fine-tune GPT-like billion parameter generative LLMs. We conduct this characterization study on modern GPU clusters with InfiniBand interconnect with up to 32 NVIDIA A100 GPUs. To the best of our knowledge, this is the first effort that evaluates the performance of PEFT methods in distributed computing environments. Ultimately, this work strives to provide valuable insights into the efficacy and practical implications of employing PEFT methods and facilitate the development of scalable approaches for fine-tuning large-scale models.**

***Index Terms*—Parameter-efficient Fine-tuning, Distributed Training, Large Language Models, GPU Clusters**

I. Introduction

The recent development of Large Language Models (LLMs) [1]–[3] represents a significant milestone in natural language processing (NLP) [4] and artificial intelligence (AI). These models [1]–[3], known for their massive scale and impressive abilities, have transformed various NLP tasks [5]–[10], including text generation [5], [6], translation [7], [8], sentiment analysis [9], and question-answering [10]. In particular, transformer architectures [11] with billions of parameters such as BERT [12], GPT [13], and T5 [14] have been trained on extensive text data. This thorough pre-training allows them to acquire complex and contextually nuanced language representations.

This research is supported in part by NSF grants #1818253, #1854828, #2007991, #2018627, #2112606, #2311830, #2312927, and XRAC grant #NCR-130002.

* Denotes equal contribution

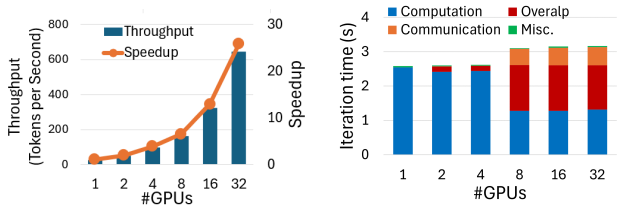
The pre-training phase of LLMs [15] is often followed by fine-tuning on a downstream task [16]. Fine-tuning an LLM is the process of adapting a pre-trained model to a specific task or domain by continuing the training on a task-specific dataset. Fine-tuning leverages pre-existing knowledge and refines the model's abilities using the task-specific dataset. However, just like pre-training, as model size increases, fine-tuning becomes computationally intensive due to the sheer volume of parameters involved. Updating all these parameters requires significant computational power and memory. This highlights the need for efficient strategies to address computational constraints while leveraging the benefits of transfer learning [17].

Parameter-efficient fine-tuning (PEFT) methods [18]–[22] have proven to be substantial techniques in the realm of LLMs. These methods, such as Low-Rank Adaptation (LoRA) [22], focus on reducing the number of parameters that need to be adjusted during fine-tuning. PEFT approaches not only make the fine-tuning process more resource-efficient but also help in maintaining the performance and generalization ability of the original model, making it feasible to deploy sophisticated LLMs in practical settings where computational resources may be limited.

However, PEFT methods often necessitate a multi-node setup to finish execution within a reasonable timeframe. This is primarily because PEFT methods, while optimizing the process by updating fewer parameters or incorporating lightweight modules, still involve complex operations on a large scale. These operations include forward propagation and back propagation on the full model parameters, communication and synchronization, and parameter updating on the active parameters. Furthermore, the scale of the datasets used for fine-tuning LLMs means that even with reduced parameter updates, the volume of data processed can be immense.

A. Motivation

The fine-tuning of LLMs in a distributed setting can lead to significant communication overhead, which hampers the overall training efficiency. When using data parallelism, the gradients of the model need to be synchronized across multiple nodes during backpropagation. This synchronization process requires extensive data exchange over the network, resulting in increased latency and potential bottlenecks as the size of



(a) Throughput/speedup (b) Iteration breakdown

Fig. 1: Throughput/speedup and iteration time breakdown of the full fine-tuning method running on 1-32 NVIDIA A100 GPUs [23] on a 1B parameters Transformer decoder model [24].

the LLM grows. The network bandwidth and latency become crucial factors, as insufficient bandwidth can slow down the synchronization process, while high latency can cause delays in gradient updates.

Figure 1 presents the throughput/speedup and iteration time breakdown for fine-tuning a Transformer decoder-based model with 1 billion parameters. The experiment is conducted on a GPU cluster comprising compute nodes with 4 A100 GPUs each and a 200G HDR InfiniBand cross-node interconnect. When fine-tuning on up to 32 GPUs, we observe that scaling out beyond a single node leads to degraded scaling efficiency. Specifically, fine-tuning on 32 GPUs resulted in a scaling efficiency of 81%. Additionally, we found that the scaling efficiency could decrease to as low as 28% with mixed precision training, as detailed later in the evaluation section V-C.

PEFT methods may offer better scaling efficiency compared to full fine-tuning. Understanding how PEFT methods operate in distributed computing environments, is crucial for unlocking their full potential, especially if they require lower communication volumes compared to full fine-tuning methods. Yet, the performance characteristics of PEFT methods in such distributed settings remain largely unexplored. Although several studies [18]–[21] have examined the convergence and memory savings of different PEFT methods, there remains a significant gap in the literature regarding the communication overhead and the potential overlap between computation and communication in multi-GPU and multi-node clusters. Therefore, in this paper, we aim to bridge this gap by comprehensively characterizing the performance of PEFT methods on modern GPU clusters. Ultimately, this work strives to facilitate the development of more efficient and scalable approaches for fine-tuning large-scale models.

B. Challenges

In light of the earlier discussion on motivation, this paper seeks to address the following overarching challenges:

- What is the impact of the communication overhead resulting from the distribution of PEFT workloads across multiple GPUs?
- How does the overlap between computation and communication influence the efficiency of PEFT workloads?
- How do PEFT methods perform when scaled on multiple GPUs with different model sizes?
- How does scaling distributed PEFT workloads compare to full fine-tuning approaches?
- What is the impact of mixed precision and quantized training on the performance of PEFT methods?
- What are the memory requirements of PEFT methods in comparison to full fine-tuning?

C. Contributions

To the best of our knowledge, this is the first effort which evaluates the performance of PEFT methods in distributed computing settings. The key contributions of this paper are as follows:

- Conducting comprehensive evaluations on a modern GPU system [25] with InfiniBand Interconnect, utilizing up to 32 NVIDIA GPUs [23].
- Characterizing the communication overhead and providing an in-depth analysis of the overlap between computation and communication for distributed PEFT workloads using four different methods: 1) LoRA [22], 2) AdaLoRa [26], 3) LoHa [27], and 4) LoKr [28].
- Analyzing the performance of distributed PEFT with varying model sizes: 160M, 410M, and 1B parameters [24].
- Comparing the scalability of PEFT methods with full fine-tuning as a baseline. We find that PEFT methods yield 95-99% scaling efficiency compared to 80% for full fine-tuning.
- Analyzing the impact of mixed precision [29] and quantized training methods [30] on PEFT workloads. We find that PEFT methods yield 84-90% scaling efficiency compared to 27% for full fine-tuning using mixed precision.
- Comparing the memory requirements of different PEFT methods with full fine-tuning as a baseline.

The remainder of this paper is organized as follows: Section II provides a background on the necessary concepts. Section III describes the proposed evaluation methodology. Section IV details the experimental setup. Section V conducts a comprehensive characterization and analysis study of using PEFT methods in distributed environments. Section VI discusses related work and relevant studies. Finally, we conclude the paper and discuss future work in section VII.

II. Background

A. Data Parallel Deep Neural Network Training

The process of training Deep Neural Networks (DNNs) can be divided into two main phases: the forward pass and the backward pass. During the forward pass, the input data is passed through the layers of the DNN to make predictions. These predictions are then compared to the actual values to calculate the training loss. In the backward pass, the training loss is propagated back through the network to calculate the gradients. These gradients are then used to update the model weights. In distributed data parallelism, the data samples are distributed across multiple computing devices. Each device

trains on its portion of the data using a local replica of the model weights. Initially, the DNN weights are communicated using a Broadcast operation, and then an Allreduce operation is used to synchronize the gradients for every training iteration to obtain the global gradients and use them for updating the model weights.

B. Parameter-efficient Fine-tuning

Parameter-efficient Fine-tuning (PEFT) is a method used in transfer learning, where a pre-trained model is used for training on a new dataset. Instead of training the entire DNN, only a portion of the model weights is updated, while the rest are frozen. This approach is particularly useful as model sizes grow, saving computing resources for tuning on specific downstream tasks. PEFT approaches are mainly categorized as additive [31], [32], selective [33], and reparameterized [22], [26]–[28] fine-tuning. Additive fine-tuning involves training only the added fully connected layers [31]. Selective fine-tuning trains specific model layers while freezing the rest [33]. Reparameterized fine-tuning minimizes the number of DNN trainable parameters using low-rank representation. LoRA is one of the most widely used PEFT methods [22]. It decomposes the weight matrix into two low-rank matrices and only updates these low-rank representations. There are multiple variants of the LoRA methods such as AdaLoRA [26], which transforms the weight matrix using singular value decomposition, Low-rank hadamard product (LoHa) [27], which combines low-rank matrices with Hadamard product, and Low-rank Kronecker product (LoKr) [28] use Kronecker product as matrix multiplication.

C. Mixed Precision and Quantized Training

The mixed precision training methods [29] can significantly speed up operations by executing them in half-precision format. In mixed precision training, two copies of the model parameters are saved in FP32 and FP16. The FP16 copy is used in the forward and backward passes, while the gradients are aggregated in FP32 and used to update the FP32 copy of the model weights. On the other hand, quantization is a technique used to compress a model by reducing the bit width of the model weights. This is achieved by scaling the input data to fit within a smaller range of data types through normalization based on the absolute maximum of the input data. To avoid large input data magnitudes, the input data is divided into blocks [34], and these blocks are quantized independently. Pre-trained model weights typically follow a zero-centered normal distribution with a standard deviation.

III. Proposed Evaluation Methodology

To comprehensively characterize the performance of PEFT methods on modern GPU clusters, we propose a systematic evaluation methodology that encompasses key performance metrics and experimental framework.

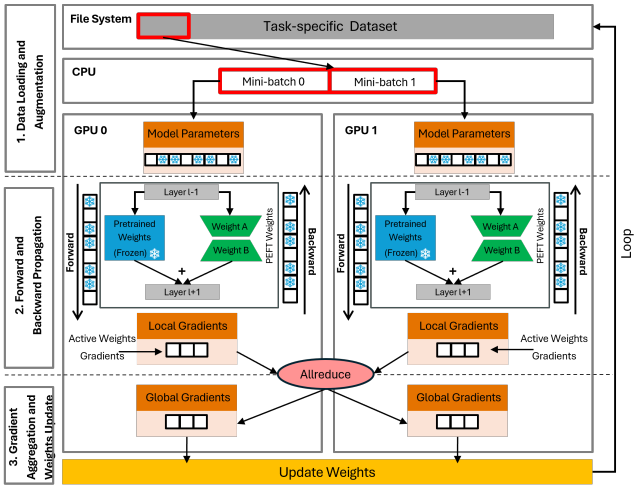


Fig. 2: Workflow of reparametrization-based PEFT approaches using distributed data parallelism on two GPUs.

A. Understanding the Workflow of Distributed PEFT Methods

We first examine the communication and computation workflow of PEFT methods, specifically reparametrization-based approaches such as LoRA. Figure 2 illustrates the workflow of distributed data parallelism iteration for the LoRA method on two GPUs. The workflow is divided into three phases:

1) Data loading and augmentation: In this phase, the CPU randomly selects samples from the datasets and creates a tokenized minibatch for each GPU.
2) Forward and backward propagation: During this phase, pre-trained model weights are frozen, and a reparametrization of the weights is represented by matrices A and B. The forward pass uses all model weights to calculate the model activations. Subsequently, the loss is computed and propagated back to generate the gradients for the active weights. It's important to note that, according to the chain rule, in order to calculate the gradients for the frozen weights, the gradients of the activations are required for all preceding model parameters. At the end of this phase, a copy of the local gradients for the active weights only is created.
3) Communication and aggregation of gradients: In this phase, local gradients are communicated and aggregated across different GPUs, resulting in the obtainment of global gradients. The communication and computation can be overlapped for different model layers. The global gradients are then used for updating the model weights and proceeding to the next training iteration.

According to this workflow, PEFT methods only require a small subset of the gradients to be communicated and aggregated across devices. Therefore, we expect to see significantly reduced communication volumes and latencies compared to full fine-tuning methods.

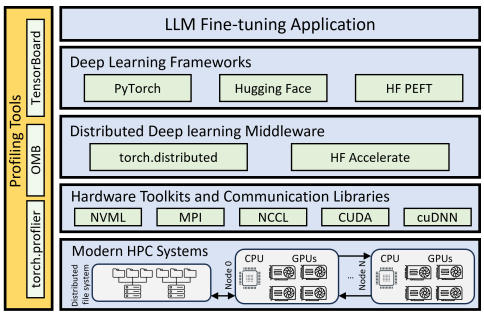


Fig. 3: Layered Architecture of the Distributed Fine-tuning Environment

B. Architecture of the Experimental Setup

Next, we will explain the layered architecture and the various hardware and software components involved in the fine-tuning process of LLMs. Figure 3 illustrates the layered representation of this architecture. The bottom layer depicts a modern HPC system containing compute nodes, each equipped with multiple CPU cores and GPU devices. These nodes are interconnected and linked to a distributed shared system. The next layer displays the toolkits and libraries required to facilitate computation and communication across GPUs and nodes. To perform distributed deep learning, a middleware layer is necessary to implement the data parallel primitives. The subsequent layer shows the different deep learning frameworks that implement LLM training and fine-tuning. This experimental architecture also requires profiling tools to analyze performance at different hardware and software levels.

C. Key Evaluation Metrics

We aim to assess the performance of PEFT methods based on several key metrics, including:

1) Speedup factor: The ratio of training time with multiple GPUs/nodes to the training time with a single GPU/node.
2) Scaling efficiency: The ratio of actual speedup to the ideal linear speedup when increasing the number of GPUs/nodes.
3) Model throughput: The number of tokens processed per second during fine-tuning.
4) Training iteration time: The elapsed time from the start of a training iteration with data loading until completion with model weights updating.
5) Training iteration breakdown: The computation, communication, and overlap times in a training iteration.
6) Pure Communication Latency: The latency needed to transfer a message of a certain size over multiple devices on a given HPC system.
7) Communication volume and latencies: The volume of data that needs to be transferred each training iteration and the total latency introduced for such transfer. The theoretical communication volume V can be calculated by multiplying the number of trainable parameters P by the size of the communicated datatype in bytes S as shown in equation 1.

$$V = S \times P \tag{1}$$

Finally, while the convergence behavior is also a critical factor for evaluating fine-tuning methods, we refer to several studies conducted in this direction later in the related works section.

IV. Experimental Setup

A. System Configurations

Our evaluations are performed on the Ascend [25] system at the Ohio Supercomputer Center (OSC). The cluster includes 24 PowerEdge XE8545 nodes and NVIDIA 200G HDR InfiniBand interconnect. Each node has 2 AMD EPYC 7643 2.3-GHz processors with 921GB of RAM, and 4 NVIDIA A100 graphics cards with 80GB of HBM.

B. Software Packages

NCCL version 2.18.1 is used for backend communication with MVAPICH2 2.3.7 as a job launcher for multi-node execution. NVIDIA CUDA Toolkit 12.1 [35] and cuDNN library 8.9.0 [36] are used for GPU support. PyTorch [37] version 2.1.0 is used for DNN training along with torch.distrubted as a distributed DL middleware. HuggingFace [38] version 4.35.2 is used to obtain the pre-trained LLM architectures and their weights. HF PEFT is used for running the fine-tuning experiments along with HF Accelerate for distributed execution.

C. Models and Datasets

We use Pythia-160M, Pythia-410M, and Pythia-1B [24] to evaluate fine-tuning methods with different model sizes. The Pythia model architecture is built using GPT-NeoX transformer blocks. We fine-tune these models on a subset of the CodeParrot [39] dataset with 3320 samples and a max sequence length of 128.

D. Fine-tuning Method

We use LoRA [22], AdaLoRA [26], LoHa [27], and LoKr [28] as representative PEFT methods for evaluation. We use the full fine-tuning method as the baseline for comparision. Table I shows the number of DNN trainable parameters for the fine-tuning methods. In Table I, we show the number of trainable parameters for these different fine-tuning methods with the different model sizes. Finally, we use QLoRA [30] for quantized fine-tuning of LLMs.

TABLE I: The number of DNN trainable parameters for the fine-tuning methods.

	Pythia-160M	Pythia-410M	Pythia-1B
Full FT	162 Million	405 Million	1011 Million
LoRA	0.13 Million	0.36 Million	0.48 Million
AdaLoRA	0.54 Million	1.45 Million	1.93 Million
LoHa	0.07 Million	0.19 Million	0.26 Million
LoKr	0.01 Million	0.03 Million	0.03 Million

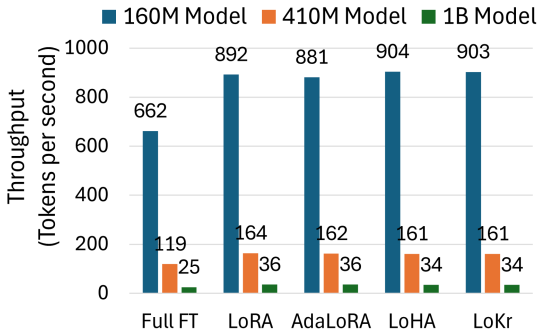
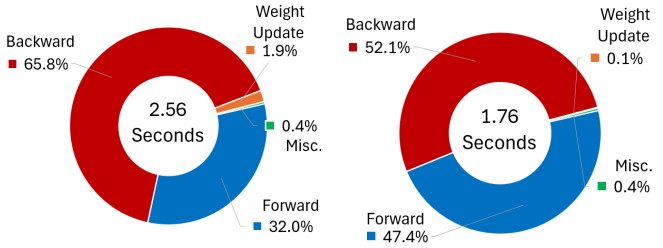


Fig. 4: Throughput of the full fine-tuning, LoRA, AdaLoRA, LoHa, and LoKr methods on a single GPU with the 160M, 410M, and 1B model sizes.



(a) Full FT (b) LoRA

Fig. 5: Breakdown of a single training iteration on 1 GPU using the full fine-tuning and LoRA methods on the 1B model size. The figures show percentages of computation, overlap, and miscellaneous operations.

V. Performance Characterization and Analysis

In this section, we aim to provide a comprehensive analysis of PEFT methods compared to full fine-tuning as a baseline in distributed environments. We conduct our experiments in the following order:

- In V-A, we start with a study of the performance of different fine-tuning methods (LoRA, AdaLoRA, LoHA, and LoKr) on 3 model sizes (160M, 410M, and 1B parameters) on a single NVIDIA A100 GPU. This is intended to establish a baseline for the scalability study.
- In V-B, we conduct an in-depth analysis of the communication, computation, and overlap, and their effect on the throughput of different fine-tuning methods on up to 32 NVIDIA A100 GPUs.
- In V-C, we look into the effect of using mixed precision and quantization training techniques in both single and multi-GPU settings.
- In V-D, we examine the peak memory requirements for different model sizes using the full fine-tuning and LoRA methods.

A. Throughput and Iteration time Breakdown on Single GPU

This section aims to establish a fundamental understanding of single GPU execution before conducting scalability studies.

1) Varying the Model Size and Fine-tuning Method: We examine the performance of full fine-tuning and PEFT methods with different model sizes on a single GPU. Figure 4 shows the throughput of 160M, 410M, and 1B models training on a single GPU using Full FT, LoRA, AdaLoRA, LoHa, and LoKr. We have chosen the optimal batch size for all the different combinations to maximize throughput. As observed, PEFT methods provide higher throughput compared to full-fine tuning for all model sizes. With LoRA see an increase of 35%, 37%, and 44% for model sizes 160M, 410M, and 1B respectively.

2) Breakdown of a Single Training Iteration: To identify the source of this improvement, we have broken down the time of a single training iteration for full fine-tuning and LoRA in Figure 5. A single training iteration takes 2.56 and 1.76 seconds for full fine-tuning and LoRA respectively. While the forward passes times remain almost the same for both methods (around 825ms), the backward pass for the LoRA method is significantly shorter at 914ms compared to 1686ms for the full fine-tuning method. This improvement is attributed to the freezing of most of the model weights, leading to reduced gradient calculations. The backward pass requires propagating only the activations gradients back to the active weights, meaning that gradients for the frozen weights are not needed. In addition to reducing the backward pass time, PEFT methods spend less time on the model weights update step as the number of trainable parameters to be updated is only a small fraction compared to the full fine-tuning method.

B. Scalability and Communication Analysis in Multi-GPU and Multi-node Environments

This section presents a detailed analysis of the scalability and communication performance of the PEFT methods in comparison to full fine-tuning as a baseline.

1) Model Throughput and Scaling Speedup: Figure 6 illustrates the throughput in tokens per second of the full fine-tuning, LoRA, AdaLoRA, LoHa, and LoKr methods scaling on up to 32 GPUs with the 1B parameters model. We observe improved scalability for the PEFT methods compared to full fine-tuning. The speedup from 1 to 32 GPUs is 25.8× (81% efficiency) for full fine-tuning and up to 31.8× (99% efficiency) for the PEFT methods. While full fine-tuning demonstrates steady scaling on up to 4 GPUs, the rate of improvement drops after training across nodes (8 GPUs and up). On the other hand, PEFT methods exhibit near-linear scalability both intra- and inter-node.

2) Communication, Computation, and Overlap: To get a better understanding of the throughput and scaling trends, we profile a single training iteration for both full fine-tuning and LoRA in Figure 7. The single GPU baseline shows lower execution time for LoRA due to the reasons explained earlier in section V-A. As we increase the number of GPUs within the same node (up to 4 GPUs), the iteration time remains almost the same for both full fine-tuning and LoRA. While full fine-tuning spends more time on communication, it mostly overlaps with computation for single-node execution.

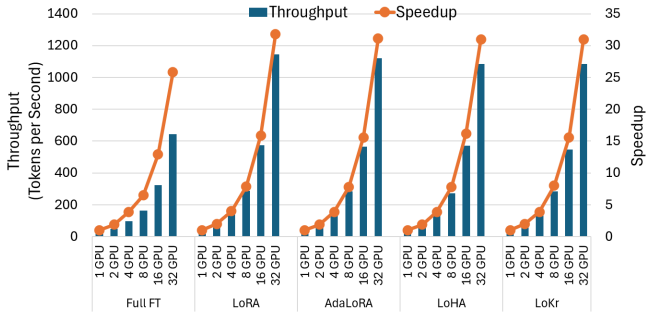


Fig. 6: Throughput of the full fine-tuning, LoRA, AdaLoRA, LoHa, and LoKr methods scaling on up to 32 GPUs with the 1B parameters model.

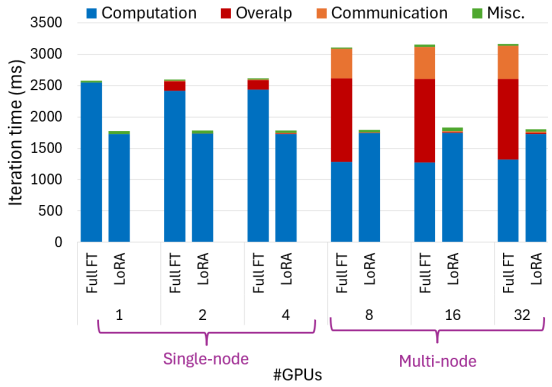
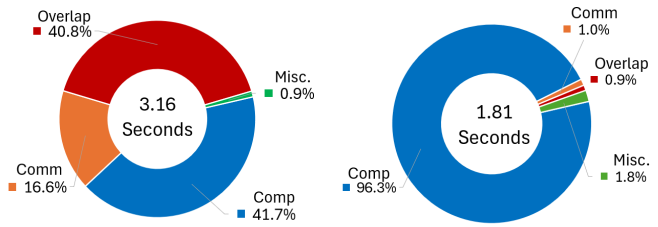


Fig. 7: Breakdown of a single training iteration on 1-32 GPUs comparing the full fine-tuning and LoRA methods on the Pythia-1B model. The figure shows the breakdown of computation, communication, overlap, and miscellaneous operations.

However, as we increase the number of GPUs beyond the one node (8-32 GPUs), we observe an increased iteration time for full fine-tuning but the same iteration time for LoRA. Although there is a significant portion of communication overlapped with computation for full fine-tuning, there is an extra communication overhead of 473ms, 509ms, and 523ms for 8, 16, and 32 GPUs respectively. This is compared to only 8.5ms, 17.7ms, and 18.6ms for the LoRA method.

3) Training Iteration Breakdown: In Figure 8, we further break down the iteration time in terms of percentages for the 32 GPUs run. The full fine-tuning iteration takes 3.16 seconds with 41.7%, 16.6%, and 40.8% spent on computation, communication, and overlap respectively. Whereas the LoRA iteration takes 1.81 seconds with 96.6% spent on computation. This shows that the LoRA iteration is 75% faster compared to the full fine-tuning iteration. The lower iteration time is partially due to a significant decrease in the communication latency and volume.

4) Total Allreduce Latency per Training Iteration: Figure 9 shows the total latency of the Allreduce operation for a single training iteration on 2-32 GPUs. As expected, the total time spent increases significantly from 151ms (single node) to



(a) Full FT (b) LoRA

Fig. 8: Breakdown of a single training iteration on 32 GPUs using the full fine-tuning and LoRA methods on the Pythia-1B model. The figures show percentages of computation, communication, overlap, and miscellaneous operations.

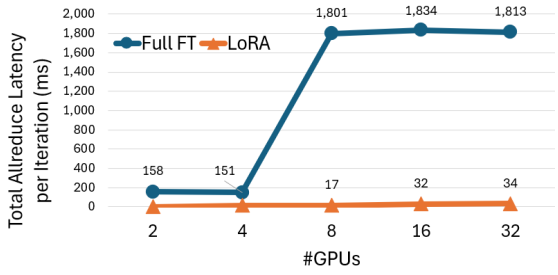


Fig. 9: Total latency of the Allreduce operation for a single training iteration on 2-32 GPUs using the full fine-tuning and LoRA methods with the Pythia-1B model.

1800ms (inter-node) for full fine-tuning. On the other hand, the Allreduce latencies for the LoRA runs only increase from 14ms to 34ms when going from 2 GPUs to 32 GPUs.

5) Communication Volume, Number of Calls, and Average Message Size: The decrease in latency for the PEFT methods is attributed to the lower communication volume due to communicating the gradients for the active parameters only. Table II shows the Allreduce communication volume, number of calls, and average message size for a single training iteration. The full fine-tuning method requires communicating 3.9GB for the 1B model divided into 50 calls with an average message size of 77MB. The PEFT methods, on the other hand, only require communicating 1.88MB, 7.38MB, 1MB, and 0.13MB for LoRA, AdaLoRA, LoHa, and LoKr respectively. These empirical results match the expected theoretical communication volume calculated by equation 1.

TABLE II: Allreduce communication volume, number of calls, and average message size for a single training iteration with different fine-tuning methods.

Fine-tuning method	Communication Volume (MB)	#Calls	Average Message Size (MB)
Full-FT	3859.64	50	77.19
LoRA	1.88	2	0.94
AdaLoRa	7.38	2	3.69
LoHa	1.00	1	1.00
LoKr	0.13	1	0.13

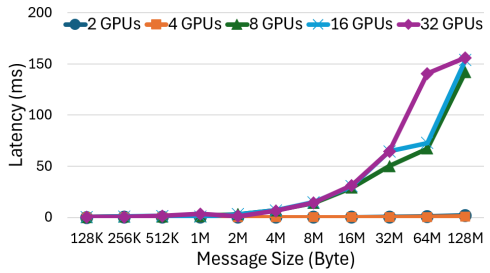


Fig. 10: GPU Allreduce latency for message sizes 128K-128MB on 1-8 GPU nodes. Each node is equipped with 4 NVIDIA A100 GPUs (32 GPUs for 8 nodes).

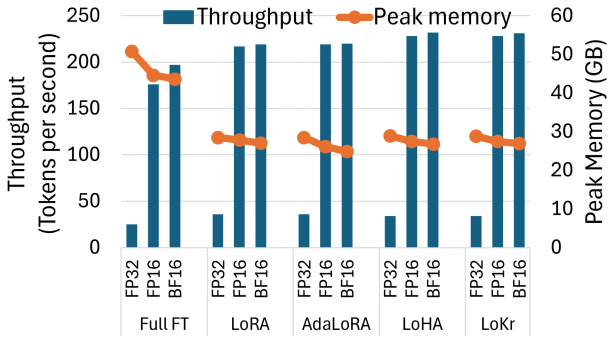


Fig. 11: Throughput with FP32, FP16, and BF16 training on one GPU using the full fine-tuning, LoRA, AdaLoRA, LoHa, and LoKr methods on the 1B parameters model.

6) Pure Communication Performance for Intra- and Inter-node: In Figure 10, we run pure Allreduce communication benchmarks to measure the latency with message sizes between 128KB and 128MB and 2-32 GPUs. The latency for intra-node communication (4 GPUs and below) remains as low as 2ms for the largest message size. However, for inter-node communication (8-32 GPUs), we see a significant increase in latency as we increase the message size beyond 2MB. These findings are consistent with the previous results that show the minimal communication overhead for PEFT methods due to the small communication volume, message size, and number of calls. They also explain the increased total Allreduce latency, increased iteration time, and decreased throughput for the full fine-tuning method.

C. Impact of Mixed Precision and Quantization Techniques

1) Mixed Precision Distributed Training with FP16 and BF16: In Figure 11, we analyze the throughput on a single GPU using FP32, FP16, and BF16 data types for full fine-tuning and PEFT methods on a 1B parameter model. We observe improvements of up to 6× using FP16 and BF16 compared to FP32. We also notice a slight decrease in peak memory usage.

Figure 12 shows the throughput and speedup with FP32, FP16, and BF16 training on 1-32 GPUs using the full fine-tuning and LoRA methods on the 1B parameter model. As observed, full fine-tuning exhibits poor scalability with FP16

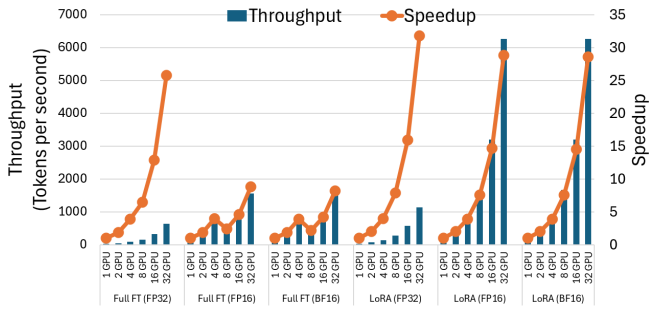


Fig. 12: Throughput with FP32, FP16, and BF16 training on 1-32 GPUs using the full fine-tuning and LoRA methods on the 1B parameters model.

and BF16 training achieving only 8.8× (28% efficiency) and 8.2× (26% efficiency) speedup, respectively, when scaled on 32 GPUs. On the other hand, the LoRA method achieves 28.8× (90% efficiency) and 28.6× (89% efficiency) with FP16 and BF16, respectively. While this is a reasonable speedup, especially with high throughput of the one GPU baseline, the mixed precision scaling efficiency does not match the one for FP32.

Although computations may be performed in FP16 or BF16 for speed and memory efficiency, gradients are converted to FP32 before communication. This is necessary to prevent significant precision loss during gradient accumulation and ensure accurate updates to model parameters. Most deep learning frameworks follow this practice.

Figure 13 shows the breakdown of a single training iteration on 32 GPUs with FP32, FP16, and BF16 for full fine-tuning and LoRA. We observed that the communication latency remains constant for different data types since the gradients are always communicated in FP32. Therefore, as the computation time decreases when using FP16 or BF16, the overall communication percentage increases in the training iteration. This explains the difference in speedup from 31.8× to 28.8× when using mixed precision with LoRA as the small communication overhead becomes more critical. This also explains the poor scalability with full fine-tuning when using mixed precision training.

2) Quantized PEFT Methods: Figure 14 depicts a comparison of throughput between LoRA with FP32, FP16, and BF16 methods and its quantized version (QLoRA) using a 1B parameters model. The experiment using a single GPU shows a 19% improvement for QLoRA over LoRA, along with a decrease in peak memory consumption.

Figure 15 compares the scaling performance of LoRA and QLoRA on up to 32 GPUs with the 1B parameters model. The best throughput for QLoRA reaches 7031 tokens per second on 32 GPUs, while LoRA achieves 6264 tokens per second. However, the speedup for QLoRA is 26.9× (84% efficiency), compared to 28.8× for LoRA with FP16. This decrease in speedup efficiency is attributed to the reasons explained earlier in section V-C1.

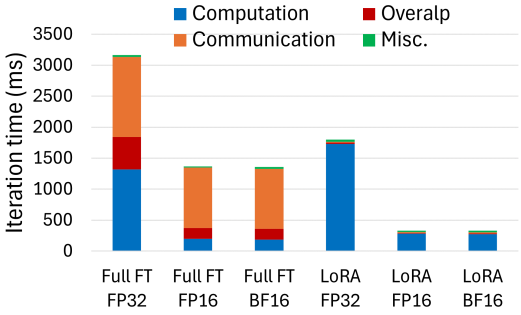


Fig. 13: Breakdown of a single training iteration on 32 GPUs using the full fine-tuning and LoRA methods on the Pythia-1B model with FP32, FP16, and BF16 training. The figure shows the stacked latencies of computation, overlap, communication, and miscellaneous operations.

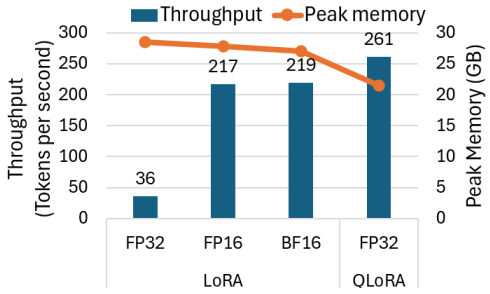


Fig. 14: Throughput of LoRA using the mixed precision methods and QLoRA on a single GPU with the 1B parameters model.

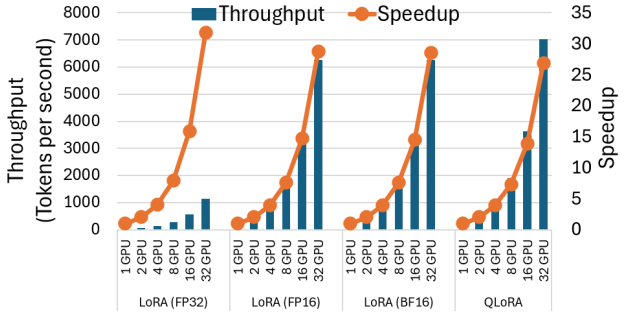


Fig. 15: Throughput of LoRA using the mixed precision methods and QLoRA on 1-32 GPUs with the Pythia-1B model.

D. Peak Memory Analysis

Figure 16 illustrates the peak GPU memory usage for different batch sizes and model sizes, comparing the full fine-tuning and LoRA methods. We observed that LoRA requires less memory across all batch and model sizes. For a batch size of one, full fine-tuning consumes 4GB, 8.6GB, and 20.1GB, while LoRA only consumes 1.3GB, 2.4GB, and 4.7GB for model sizes 160M, 410M, and 1B respectively. When the batch size is 128, full fine-tuning consumes 31GB, 56.9GB, and 73.8GB, whereas LoRA consumes 24.9GB, 42.4GB, and 52.4GB for model sizes 160M, 410M, and 1B respectively. This significant reduction in peak GPU memory, especially for small batch sizes, gives PEFT methods an advantage when attempting to fit the entire model within the GPU's memory.

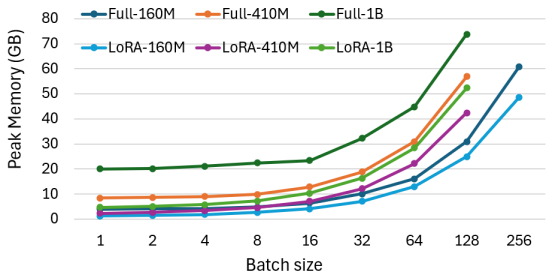


Fig. 16: Peak GPU memory for different batch sizes using the full fine-tuning and LoRA methods with the 160M, 410M, and 1B parameters models.

VI. Related Work

Although several studies have examined the convergence and memory savings of different PEFT methods, the performance characteristics of these methods in distributed environments remain largely unexplored. Han et al. [19] provide a comprehensive survey on PEFT methods and examine the additional computational overhead for PEFT methods. Hu et al. [18] evaluate the accuracy of adapter-based PEFT methods on 14 datasets from arithmetic reasoning and commonsense reasoning. Lialin et al. [20] compared storage, memory, and computational efficiency, and provided a summary of the model size and the number of DNN trainable parameters. Xu et al. [21] explain that PEFT methods reduce the number of DNN trainable parameters and memory usage compared to full fine-tuning. They conduct evaluations with different number of trainable parameters and analyze the accuracy and peak memory using full fine-tuning and different PEFT methods. Xin et al. [40] provide an analysis of PEFT methods with vision models. Particularly, they study the performance of vision transformers on visual downstream tasks.

VII. Conclusion and Future Work

In this paper, we thoroughly examined the communication dynamics involved in distributed parameter-efficient fine-tuning (PEFT) for large language models (LLMs). Our study emphasizes the crucial role of communication efficiency in the performance and scalability of PEFT methods when utilized across GPU clusters. Through systematic analysis of various PEFT techniques, we have illustrated how communication overhead can significantly impact throughput, training time, and overall model performance. Our findings indicate that PEFT methods inherently reduce communication and computational burdens compared to full fine-tuning. While this study mostly focused on data parallelism, in future research, we aim to explore more advanced parallelization strategies, such as Fully Sharded Data Parallel (FSDP) [41] training and 3D parallelism [42], for out-of-core models that do not fit in the memory of a single GPU. Overall, this study sheds light on the intricate interplay between communication and computation in PEFT, laying the groundwork for more efficient and scalable approaches to fine-tuning large-scale models in various application domains.

REFERENCES

[1] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," *ACM Trans. Intell. Syst. Technol.*, vol. 15, mar 2024.

[2] H. Zhao, H. Chen, F. Yang, N. Liu, H. Deng, H. Cai, S. Wang, D. Yin, and M. Du, "Explainability for large language models: A survey," *ACM Trans. Intell. Syst. Technol.*, vol. 15, feb 2024.

[3] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, "A survey of large language models," 2023.

[4] N. Patwardhan, S. Marrone, and C. Sansone, "Transformers in the real world: A survey on nlp applications," *Information*, vol. 14, no. 4, 2023.

[5] Y. Mo, H. Qin, Y. Dong, Z. Zhu, and Z. Li, "Large language model (llm) ai text generation detection based on transformer deep learning algorithm," 2024.

[6] J. Li, T. Tang, W. X. Zhao, J.-Y. Nie, and J.-R. Wen, "Pre-trained language models for text generation: A survey," *ACM Comput. Surv.*, vol. 56, apr 2024.

[7] B. Yao, M. Jiang, D. Yang, and J. Hu, "Benchmarking llm-based machine translation on cultural awareness," 2024.

[8] Z. Feng, Y. Zhang, H. Li, W. Liu, J. Lang, Y. Feng, J. Wu, and Z. Liu, "Improving llm-based machine translation with systematic self-correction," 2024.

[9] S. Kumawat, I. Yadav, N. Pahal, and D. Goel, "Sentiment analysis using language models: A study," in *2021 11th International Conference on Cloud Computing, Data Science and Engineering (Confluence)*, pp. 984–988, 2021.

[10] D. Su, Y. Xu, G. I. Winata, P. Xu, H. Kim, Z. Liu, and P. Fung, "Generalizing question answering system with pre-trained language model fine-tuning," in *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, (Hong Kong, China), pp. 203–211, Association for Computational Linguistics, Nov. 2019.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[13] G. Yenduri, R. M, C. S. G, S. Y, G. Srivastava, P. K. R. Maddikunta, D. R. G, R. H. Jhaveri, P. B, W. Wang, A. V. Vasilakos, and T. R. Gadekallu, "Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions," 2023.

[14] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[15] H. Wang, J. Li, H. Wu, E. Hovy, and Y. Sun, "Pre-trained language models and their applications," *Engineering*, vol. 25, pp. 51–65, 2023.

[16] H. Dong, Z. Cheng, X. He, M. Zhou, A. Zhou, F. Zhou, A. Liu, S. Han, and D. Zhang, "Table pre-training: A survey on model architectures, pre-training objectives, and downstream tasks," 2022.

[17] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021.

[18] Z. Hu, L. Wang, Y. Lan, W. Xu, E.-P. Lim, L. Bing, X. Xu, S. Poria, and R. K.-W. Lee, "Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models," 2023.

[19] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," 2024.

[20] V. Lialin, V. Deshpande, and A. Rumshisky, "Scaling down to scale up: A guide to parameter-efficient fine-tuning," 2023.

[21] L. Xu, H. Xie, S.-Z. J. Qin, X. Tao, and F. L. Wang, "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment," 2023.

[22] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.

[23] NVIDIA, "Nvidia a100 tensor core gpu." https://www.nvidia.com/en-us/data-center/a100/.

[24] S. Biderman, H. Schoelkopf, Q. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. Sai Prashanth, E. Raff, A. Skowron, L. Sutawika, and O. van der Wal, "Pythia: A suite for analyzing large language models across training and scaling," *arXiv:2304.01373*, 2023.

[25] O. S. Center, "Ascend supercomputer," 2022.

[26] Q. Zhang, M. Chen, A. Bukharin, N. Karampatziakis, P. He, Y. Cheng, W. Chen, and T. Zhao, "Adalora: Adaptive budget allocation for parameter-efficient fine-tuning," 2023.

[27] N. Hyeon-Woo, M. Ye-Bin, and T.-H. Oh, "Fedpara: Low-rank hadamard product for communication-efficient federated learning," 2023.

[28] S.-Y. Yeh, Y.-G. Hsieh, Z. Gao, B. B. W. Yang, G. Oh, and Y. Gong, "Navigating text-to-image customization: From lycoris fine-tuning to model evaluation," 2024.

[29] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," *arXiv:1710.03740*, 2018.

[30] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *arXiv:2305.14314*, 2023.

[31] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," *arXiv:1902.00751*, 2019.

[32] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[33] M. Gheini, X. Ren, and J. May, "Cross-attention is all you need: Adapting pretrained transformers for machine translation," in *Conference on Empirical Methods in Natural Language Processing*, 2021.

[34] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," 2022.

[35] NVIDIA, P. Vingelmann, and F. H. Fitzek, "Cuda, release: 10.2.89," 2020.

[36] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," 2014.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *arXiv:1912.01703*, 2019.

[38] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv:1910.03771*, 2020.

[39] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. Horace, D. Song, and J. Steinhardt, "Measuring coding challenge competence with apps," *Advances in Neural Information Processing Systems*, 2021.

[40] Y. Xin, S. Luo, H. Zhou, J. Du, X. Liu, Y. Fan, Q. Li, and Y. Du, "Parameter-efficient fine-tuning for pre-trained vision models: A survey," 2024.

[41] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 3505–3506, 2020.

[42] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, E. Zhang, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro, "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model," 2022.