Communication-Efficient MoE Fine-Tuning with Locality-Aware Expert Placement

Chenghao Hu, Yufei Kang, Baochun Li
{ch.hu, yufei.kang}@mail.utoronto.ca, bli@ece.toronto.edu
Department of Electrical and Computer Engineering, University of Toronto

***Abstract*—With the prevailing Mixture-of-Experts (MoE) architecture pushing the performance of Large Language Models (LLMs) to new limits, fine-tuning MoE models presents a significant challenge due to their tremendous number of parameters and sparsely activated network structures. While expert parallelism has been proposed to train large-scale MoE models by distributing expert layers among multiple devices, it fails to exploit the unique communication patterns in fine-tuning pre-trained MoE models. In this paper, we demonstrate that expert layers are not uniformly accessed, but exhibit a stable locality, with some experts being accessed more frequently than others throughout the fine-tuning process. Based on this insight, we introduce VELA, a novel fine-tuning system for MoE architectures that leverages expert locality to reduce communication overhead. Specifically, VELA implements a novel training and communication framework that separates expert layers from the MoE model, and employs a locality-aware expert placement mechanism to minimize the communication overhead, thereby significantly improving the fine-tuning efficiency. Our extensive array of evaluations demonstrates that VELA decreases the communication overhead by up to 25%, consequently accelerating the fine-tuning process by up to 28% compared to conventional methods.**

I. INTRODUCTION

In recent years, large language models (LLMs) have revolutionized the field of natural language processing, becoming increasingly popular and prevalent across a wide array of applications [1]. These powerful models have demonstrated remarkable capabilities in tasks ranging from text generation to complex reasoning. Building upon their success, researchers have pushed the boundaries even further by combining LLMs with the Mixture-of-Experts (MoE) architecture [2], resulting in MoE models that achieve unprecedented levels of performance [3].

As LLMs continue to evolve, fine-tuning [4]–[6] has emerged as a common practice to harness their immense potential for specific domains or tasks. However, the integration of MoE architectures with LLMs presents a significant challenge for end users to perform fine-tuning. The number of parameters in MoE models can easily break into tens or even hundreds of billions due to the existence of parallel experts. For instance, while Mistral 7B [7] is already considered a large model with roughly seven billion parameters, its MoE variant, Mixtral 8×7B [3], easily reaches beyond fifty billion. With such a large number of model parameters, even the preparatory work for MoE models, such as loading the parameters, imposes harsh requirements on both hardware (*e.g.*, GPU memory) and software (*e.g.*, distributing parameters), not to mention the subsequent fine-tuning process.

To effectively train MoE models, expert parallelism [2], often combined with other parallelism techniques, has become the bedrock computation paradigm when MoE models are involved. As the name suggests, expert parallelism distributes the expert layers across multiple computing devices (*e.g.*, GPUs or TPUs) to alleviate the pressure of memory constraints. During computation, the intermediate results of expert layers must be synchronized through intra-node (when computing devices are located on the same machine) and cross-node (when computing devices are located on different machines) communication to maintain their computational integrity. This approach has been implemented in major LLM training frameworks, such as DeepSpeed [8] and Megatron-LM [9], and proven to be effective for *training an MoE model from scratch*.

However, fine-tuning a pre-trained MoE model is decidedly different from training it from scratch. In an MoE model, each "expert" is essentially a sub-network that can specialize in handling specific types of inputs or tasks. During the initial pre-training stage, the training framework must achieve a balanced load distribution among experts to ensure their parameters are all sufficiently trained from randomness [2], [10]. However, once an MoE model is fully trained, recent studies have shown that certain experts tend to become more active and will be accessed more frequently than others in specific tasks [11]–[13].

The nature of fine-tuning is adapting the model to a specific task, typically using a dataset that is too small to uniformly engage all experts. In this work, we discover that such imbalanced access pattern of experts, namely *expert locality*, not only exists in MoE fine-tuning, but also stays stable throughout the entire fine-tuning process. Since expert layers are distributed using expert parallelism, the communication overhead is inevitably affected by how experts are distributed and accessed, *e.g.,* more tokens will be sent to popular experts. Unfortunately, conventional expert parallelism didn't take this into account, potentially leading to inefficient communication and slowing down the fine-tuning process.

In this paper, we propose VELA, a fine-tuning system specifically designed for MoE models based on these insights. VELA implements a novel computation and communication

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Research Program.

framework, which strips the expert layers out of the model backbone, providing sufficient flexibility for users to manipulate the distribution of expert layers. A highlight of VELA is a new locality-aware expert placement mechanism, which reduces the communication overhead by strategically distributing expert layers based on expert locality.

Highlights of our original contributions in this paper are as follows. *First*, we present the phenomenon of *expert locality* and explore its impact on model fine-tuning. Through empirical measurements, we demonstrate that the probability of each expert being selected in a pre-trained MoE model is biased, and such biased probability distribution remains stable throughout the fine-tuning process. We also provide a theoretical analysis of this phenomenon, which not only explains the observed results, but also lays a solid theoretical foundation to exploit expert locality for efficient MoE fine-tuning.

Second, we implement a new fine-tuning and communication framework that separates the expert layers from the backbone of MoE models. The backbone model is hosted on a *Master* process, while experts are offloaded to multiple *Expert Manager* processes located on other available computing devices. This design provides users with a flexible environment to manipulate expert distribution at runtime.

Finally, based on expert locality and our framework design, we propose a new locality-aware expert placement mechanism that formulates the expert placement as an expectation minimization problem, in order to find the optimal expert arrangement and to achieve the best fine-tuning efficiency. Our new MoE fine-tuning system, VELA, integrates our framework design with our new expert placement mechanism. An extensive array of evaluation results demonstrates that VELA decreases the communication overhead by up to 25%, and accelerates the fine-tuning process by up to 28% compared to conventional expert parallelism.

II. PRELIMINARIES

Mixture-of-Experts. The Mixture-of-Experts model has emerged as an evolution of the mainstream large language model, which is built upon the transformer design [1]. Traditional transformers consist of two primary components: multi-head self-attention and a feed-forward network (FFN), interconnected through residual addition and normalization. MoE models retain most of this structure but enhance it by replacing the FFN with a specialized MoE block [2].

The workflow of the MoE block is depicted in Fig. 1. The input of the MoE block is typically a three-dimensional tensor with the shape of $[\text{batch size}, \text{sequence length}, \text{feature size}]$. During pre-processing, the input tensor is reshaped into a two-dimensional form $[\text{batch size} \times \text{sequence length}, \text{feature size}]$, because the tokens are processed individually in the MoE block, regardless of their sequence origin. Consequently, a conjugated post-processing step at the end of the MoE block restores the shape of the output tensor.

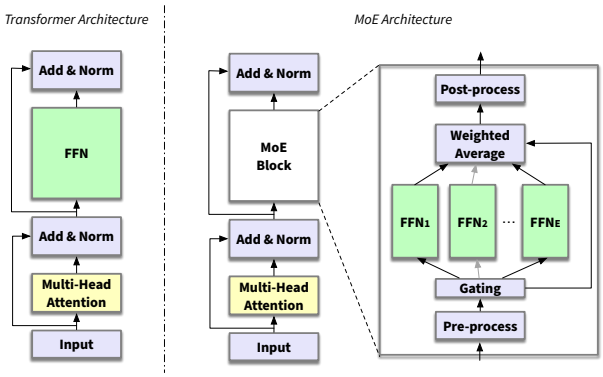


Fig. 1. The architecture of transformer and MoE models.

Each MoE block contains multiple "experts," which are essentially FFNs. Unlike traditional models, only a subset of the experts is selected for computation for each token. For instance, in the Mixtral 8×7B model, two out of eight experts are chosen per token, with different tokens potentially selecting different pairs of experts.

The selection of experts is determined by a gating mechanism, which functions as a softmax classifier predicting the probability of expert selection for each token. The tokens are then processed by the chosen experts, and the computation results for each token are weighted averaged to produce the final output. Specifically, the MoE output y for a token x is given by

$$y = \frac{\sum p_i y_i(x)}{\sum p_i}, \tag{1}$$

where y_i is the output of the selected expert i, and p_i is its softmax probability. With the MoE architecture, the large number of parameters allows MoE models to learn more complex feature representations. Meanwhile, the sparsely activated experts require significantly less resources for inference compared to dense models at the same scale. As a result, MoE models have become increasingly popular in both academic research and commercial products.

Fine-Tuning with MoE models. Due to the massive number of model parameters, it's typically impractical to fine-tune the entire MoE model. Instead, parameter-efficient fine-tuning techniques have been developed to reduce the scale of trainable parameters. One of the most commonly used techniques is Low-Rank Adaptation (LoRA) [14]. Given a pre-trained linear layer with parameter matrix $W \in R^{n\times m}$, LoRA injects two low rank matrices $A \in R^{n\times d}, B \in R^{d\times m}$ into this layer, and the output of this layer becomes

$$y = Wx + ABx.$$

During fine-tuning, only matrices A and B are optimized while W remains fixed. Since $d \ll \min(n, m)$, the number of parameters to optimize is significantly reduced.

While parameter-efficient fine-tuning techniques reduce the scale of trainable parameters, the pre-trained model, despite being frozen, still needs to be loaded into the computing

device, presenting significant hardware challenges. To alleviate this memory constraint, model parameters can be quantized before fine-tuning begins [15]. However, as MoE models continue to grow in size and complexity, it's impractical to assume all parameters can be quantized to fit into a single machine, not to mention the precision loss due to quantization. Therefore, fine-tuning MoE models in a distributed manner is a more reliable choice in practice.

Expert parallelism. To train MoE models, expert parallelism [2] is considered a foundational computation paradigm that distributes the MoE blocks, the largest components in an MoE model, across multiple computing devices. As shown in Fig. 2, experts numbered from 1 to E within the same MoE blocks are assigned to different devices, while all other layers are *replicated* on each device.

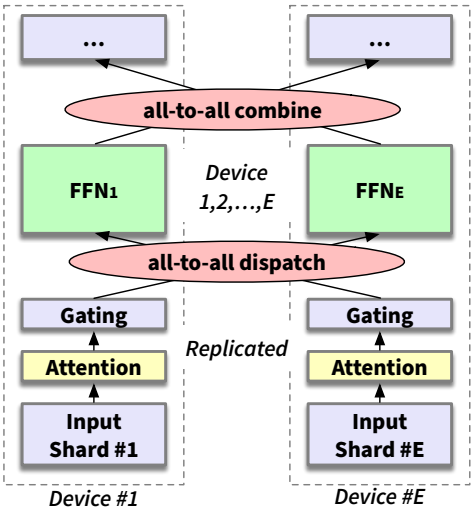


Fig. 2. An illustration of expert parallelism.

During runtime, the input data is sharded across devices, in a fashion similar to data parallelism, with each device processing only a portion of the inputs. After passing through the gating layer, tokens are dispatched to their selected experts for computation. The computation results are then gathered back to their original positions through another all-to-all communication. Finally, the weighted average shown in Eq. (1) is performed to obtain the final results.

However, expert parallelism and its follow-up improvements (*e.g.*, [16]) are designed primarily for training models from scratch, making some design choices, such as replicating all non-expert layers, unnecessary for fine-tuning scenarios. More importantly, the behavior of the gating mechanism differs significantly during fine-tuning. When training an MoE model from scratch, all expert layers are randomly initialized in the beginning. To ensure these experts receive sufficient training, the gating layer must achieve a certain degree of load balancing among the experts. In contrast, after the model is fully trained, different experts may specialize in handling different inputs or tasks, leading to an imbalanced access pattern when the model is applied in specific application scenarios [11]. This distinction between pre-training and fine-tuning suggests that conventional expert parallelism can be further optimized for fine-tuning tasks, which is a possibility we shall explore in this paper.

III. Expert Locality in Fine-Tuning

A. Expert Locality of MoE Models

Expert locality, the phenomenon where certain experts are accessed more frequently during MoE model training, is widely observed. To gain a clear insight into this phenomenon, we conducted a measurement study fine-tuning a TinyMistral-6×248M model[1], which is referred to as `TinyMistral`, on the Tiny-Shakespeare [17] dataset. Specifically, `TinyMistral` consists of 12 MoE blocks, each containing six experts, with two selected for each token.

Empirically, we passed all data through the model in the inference mode (*i.e.*, no fine-tuning has occurred yet) and counted the number of times each expert was accessed. We then calculated the frequency by determining the ratio of tokens processed by each expert compared to the total number of tokens and plotted the results in Fig. 3(a). Notably, there is a disparity in access frequency among experts within the same MoE block. For instance, experts 2 and 3 in the first block are accessed significantly more frequently than their counterparts.

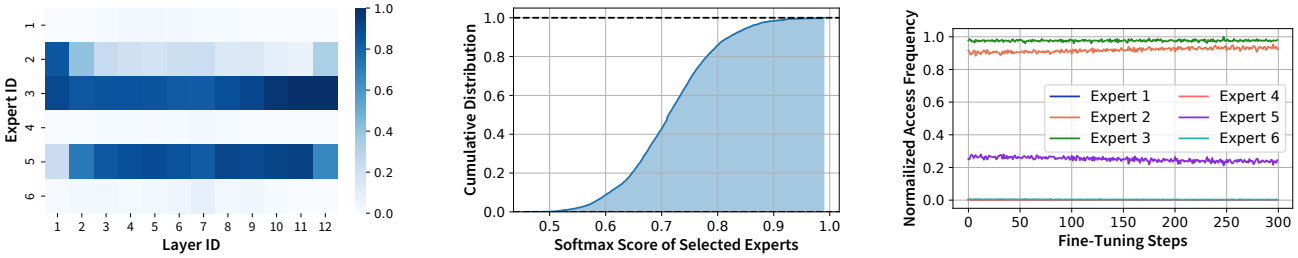
This imbalanced access pattern exists throughout the entire lifecycle of MoE models, from pre-training to deployment for inference. Such a phenomenon poses unique challenges and opportunities across different application scenarios, and how it is handled varies based on the specific use cases:

- *Pre-training stage*: Imbalanced expert access during pre-training leads to popular experts receiving more training than others [2]. Therefore, it is crucial to mitigate expert locality during the pre-training phase. For instance, one approach is to introduce auxiliary loss terms [10] that penalize this imbalance, enforcing a more equal utilization of experts to ensure that all of them receive sufficient training.
- *Inference stage*: Despite efforts to achieve load balance during pre-training, expert locality still exists after the model is fully trained and deployed in an inference environment. As a result, uniformly allocating computing resources across all experts leads to inefficient utilization, since some experts are barely used. Researchers have leveraged this insight to optimize the deployment of pre-trained MoE models [11]–[13] by strategically allocating more resources to frequently accessed experts.

Now let's return to model fine-tuning. Similar to inference deployment, fine-tuning also works with pre-trained MoE models, whose model weights are fully trained, and expert access patterns have already been established. Given this situation, it seems natural to capture the selection probability of each expert in advance to optimize the fine-tuning process.

However, there is an important distinction between MoE model deployment and fine-tuning. During deployment, model weights remain fixed in inference, while in fine-tuning, these

[1]https://huggingface.co/M4-ai/TinyMistral-6x248M



(a) Expert access frequency.

(b) Cumulative distribution of softmax score of selected experts.

(c) Expert access frequency during fine-tuning.

Fig. 3. Expert locality persists in fine-tuning `TinyMistral` model on Tiny-Shakespeare.

weights are continuously updated. This means that model behaviors, including expert selection patterns, could potentially change during the fine-tuning process. Therefore, before we can exploit the potential benefits of expert locality, we must first address a crucial question: *will expert selection patterns remain stable as the model behavior evolves during fine-tuning?*

B. Expert Selection in Fine-Tuning

To address this question, we theoretically study the stability of the expert selection process and confirm the reliability of expert locality as a predictor throughout the fine-tuning process.

As shown in Fig. 1, experts are selected by the softmax predictor in the gating mechanism. Thus, we analyze the softmax output when input perturbation exists.

Let's examine a simplified abstraction of a gating mechanism within the MoE model. Let x denote the input of the MoE model, and let $f(x; w_t)$ denote all the computations prior to the softmax predictor. The softmax prediction $P_t(x)$ at step t can be obtained by

$$P_t(x) = \text{softmax}\left(f(x; w_t)\right),$$

where w_t represents the model parameters to be fine-tuned.

Assume there are E experts. The e-th component $P_t(x)[e]$, where $e \in \{1, 2, \ldots, E\}$, denotes the softmax score for the e-th expert. The stability of expert selection can be quantified by

$$\Delta P_t(e) = \left|P_t(x)[e] - P_{t-1}(x)[e]\right|, \tag{2}$$

where $\Delta P_t(e)$ represents the difference in the softmax scores for expert e after one step of fine-tuning. Intuitively, a smaller value indicates that the gating mechanism consistently selects the same experts for the same token inputs during fine-tuning. Theoretically, the following theorem establishes a bound on this difference and provides critical insights into the expert selection process during fine-tuning.

Theorem 1. *Assume that the loss function $f(x; w)$ is L-Lipschitz continuous with bounded gradients, i.e., $||\nabla f(x; w)|| \leq L$. If the model weights w_t are optimized from w_{t-1} using the SGD optimizer with learning rate μ at step t, such that $w_t = w_{t-1} - \mu \nabla f(x; w_{t-1})$, we have*

$$\Delta P_t(e) \leq \mu E L^2 \underbrace{P_{t-1}(x)[e](1 - P_{t-1}(x)[e])}_{\textit{uncertainty term}},$$

Proof. Let $y_t = f(x; w_t)$. The softmax score for expert e is

$$P_t(x)[e] = \text{softmax}(y_t)[e] = \frac{\exp(y_t[e])}{\sum_{k=1}^{E} \exp(y_t[k])}$$

The partial derivatives with respect to each component of y_t are

$$\frac{\partial P_t(x)[e]}{\partial y_t[k]} = \begin{cases} P_t(x)[e](1 - P_t(x)[e]) & \text{if } k = e \\ -P_t(x)[e]P_t(x)[k] & \text{if } k \neq e \end{cases}$$

We approximate $P_t(x)[e]$ by the first-order Taylor expansion of the softmax function around y_{t-1}, yielding

$$P_t(x)[e] \approx P_{t-1}(x)[e] + \sum_{k=1}^{E} \frac{\partial P_{t-1}(x)[e]}{\partial y_{t-1}[k]} \left(y_t[k] - y_{t-1}[k]\right).$$

Substituting $P_t(x)[e]$ in (2) with the Taylor expansion, we have

$$\Delta P_t(e) \approx \left| \sum_{k=1}^{E} \frac{\partial P_{t-1}(x)[e]}{\partial y_{t-1}[k]} \left(y_t[k] - y_{t-1}[k]\right) \right|. \tag{3}$$

Given Lipschitz continuity, we have

$$\begin{aligned} |y_t[k] - y_{t-1}[k]| &\leq ||f(x; w_t) - f(x; w_{t-1})|| \\ &\leq L||w_t - w_{t-1}|| \\ &\leq L\mu||\nabla f(x; w_{t-1})|| \\ &\leq \mu L^2. \end{aligned}$$

Then (3) can be rewritten as

$$\Delta P_t(e) \leq \mu L^2 \left| \sum_{k=1}^{E} \frac{\partial P_{t-1}(x)[e]}{\partial y_{t-1}[k]} \right|. \tag{4}$$

Finally, we bound the partial derivatives as follows:

$$\left|\sum_{k=1}^{E}\frac{\partial P_{t-1}(x)[e]}{\partial y_{t-1}[k]}\right|$$
$$\leq\left|\frac{\partial P_{t-1}(x)[e]}{\partial y_{t-1}[e]}\right|+\left|\sum_{k=1,k\neq e}^{E}\frac{\partial P_{t-1}(x)[e]}{\partial y_{t-1}[k]}\right|$$
$$\leq P_{t-1}(x)[e](1-P_{t-1}(x)[e])+\sum_{k=1,k\neq e}^{E}P_{t-1}(x)[e]P_{t-1}(x)[k]$$
$$\leq EP_t(x)[e](1-P_{t-1}(x)[e]).$$

The last inequality is derived from $1-P_{t-1}(x)[e]=\sum_{k=1,k\neq e}^{E}P_{t-1}(x)[k]$, implying that $P_{t-1}(x)[k]\leq(1-P_{t-1}(x)[e])$ for any single k. Putting the bound back into (4) completes the proof. □

C. Locality Persists in MoE Fine-Tuning

According to Theorem 1, the stability of expert selection during the fine-tuning process is determined by the uncertainty term $P_{t-1}(x)[e](1-P_{t-1}(x)[e])$. This uncertainty term is influenced by the softmax scores $P_{t-1}(x)[e]$ of the expert selection from the previous step. When $P_{t-1}(x)[e]$ approaches 0 or 1, the uncertainty term approaches 0, meaning the gating mechanism makes consistent selections throughout the process. This observation leads us to the following claim:

Claim 1. *When the gating mechanism exhibits a high preference in selecting certain experts initially, it maintains this selection pattern throughout the fine-tuning stage.*

To empirically validate this claim, we fine-tuned `TinyMistral` for 300 steps. Before fine-tuning, we recorded the sum of the softmax scores for the selected experts in the first MoE block, and plotted the cumulative distribution in Fig. 3(b). The results show that nearly all scores exceed 0.5, with over 60% of the softmax scores higher than 0.7. Note that the model has six experts and only two are selected, the remaining 0.3 scores are shared among the other four experts. Such a distribution indicates that the model exhibits high preference in its expert selections, suggesting a strong initial bias in the routing decisions.

Throughout the fine-tuning process, we monitored the behavior of the gating mechanism in the first MoE block. As presented in Fig. 3(c), the frequency with which each expert in this block is accessed remains very stable. Notably, popular experts become slightly more favored as fine-tuning progresses. This observation aligns perfectly with our theorem, which predicts that when a model demonstrates high preference in its expert selection, it maintains these choices throughout the fine-tuning steps.

As a brief summary, we first visualized the expert locality phenomenon in pre-trained MoE models (Fig. 3(a)). Our theoretical analysis demonstrated that expert selection remains stable during fine-tuning, which was corroborated by empirical observations (Fig. 3(c)). These findings provide a solid foundation for exploiting expert selection probability and developing a more efficient fine-tuning system.

IV. VELA: Fine-Tuning MoE Models with Expert Locality

To facilitate the fine-tuning process of MoE models, we present VELA, a new system framework designed to enhance the fine-tuning efficiency by leveraging expert locality. Specifically, there are two key components in VELA:

1) A novel distributed framework that detaches the expert layers from MoE model, allowing users to manipulate MoE model fine-tuning workloads with unprecedented flexibility.
2) A locality-aware expert placement mechanism that minimizes communication overhead by strategically distributing experts based on their estimated access probabilities.

Next, we will present VELA's architecture and functionality in detail.

A. Framework Design

VELA is built upon expert parallelism and aims to optimize expert placement to minimize the communication overhead. To achieve this, it introduces a novel distributed fine-tuning framework, which provides unprecedented flexibility in managing the expert distribution. There are two highlights in VELA's framework design.

Expert broker. As previously illustrated in Fig. 2, conventional expert parallelism tightly couples expert layers with other layers, making their fine-grained placement impractical. To address this, VELA first introduces an *Expert Broker* design that fully detaches the expert layers from the MoE model, facilitating flexible placement strategies.

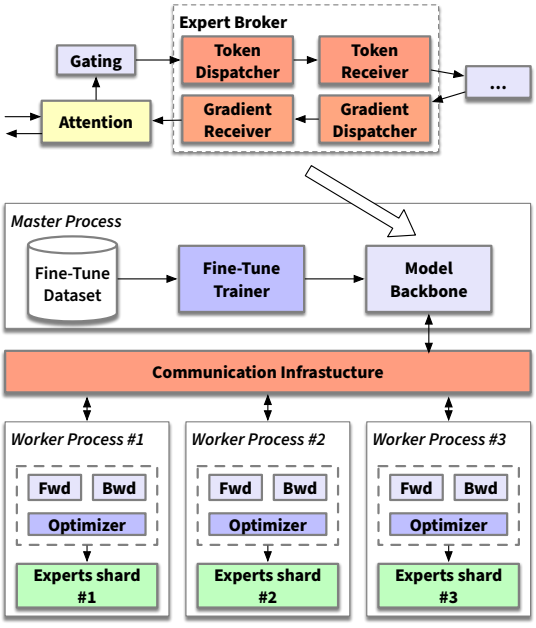


Fig. 4. The overview of VELA framework, where 'Fwd' and 'Bwd' means forward and backward computations respectively.

As shown in Fig. 4, VELA replaces the original MoE block with a special expert broker layer. As the name suggests, the

broker layer does not perform any computation; instead, they function as proxies, offloading data to expert layers which will be separated from the model body and possibly placed on other devices. In the forward pass, the broker layer dispatches tokens to each expert according to the gating output, and waits to receive the expert computation results. Similarly, during the backward pass, a conjugated set of gradient dispatcher and receiver will manage the exchange of gradients.

We refer the processed model as the *model backbone*, excluding the expert layers that account for the majority of the model size. This allows the expert layers to be flexibly hosted and computed on any device, as long as their locations are reported to the broker layers such that data flow of the intermediate results can be directed to the correct place.

From the trainer's perspective, the expert broker functions exactly like a real MoE block, processing inputs from previous layers and passing expert computation results to subsequent layers. This allows the model backbone to be used directly by trainers, making the expert broker design transparent to the fine-tuning process.

Master-worker architecture. As we have shown in Fig. 2, expert parallelism is typically combined with data parallelism to handle the huge amount of pre-training data. In this way, all non-expert layers are replicated across devices, allowing input data to be sharded and processed by different devices. However, we argue that this design is not suitable for end-user fine-tuning for several reasons:

1) Fine-tuning datasets are usually domain-specific and much smaller than pre-training data.
2) Most end-users do not have the computational resources to replicate the model, especially when it becomes unnecessary.
3) Data parallelism requires an expensive all-reduce operation at the end of each training step to combine gradients and update parameters.

In simple words, applying data parallelism to large MoE models when working with small fine-tuning datasets leads to unnecessary model replication, which significantly wastes computational resources especially for end users.

To address these issues, VELA deviates from the conventional design of expert parallelism frameworks and adopts a master-worker architecture, which eliminates redundancy and improves communication efficiency in fine-tuning. As shown in Fig. 4, VELA operates with two different roles: a *master process* and a series of *worker processes*. The model backbone is hosted on the master process because it is much smaller than the experts (*e.g.,* the model backbone of Mixtral 8×7B is only 3 GB while the entire model is over 87 GB in half precision), typically we just need to launch one master process. Meanwhile, the expert layers are distributed among worker processes. To avoid unnecessary cross-GPU data transfer during computation, we follow common practices in distributed training and launch worker processes on each available GPUs.

Flexibility of VELA. The expert broker design and master-worker architecture in VELA decouple the physical connection between expert layers and other layers, providing extraordinary flexibility in arranging the model-to-device mapping. This flexibility allows users to freely configure the specific deployment location for each expert and the number of experts to be deployed on each device.

The workflow of VELA is presented in Fig. 4. The master process initializes the fine-tuning tasks, while worker processes wait for intermediate results or gradients from the model backbone. Upon receiving the data, worker processes perform forward and backward computations on the requested experts. When a fine-tuning iteration is complete, an optimization step is executed on the trainable parameters.

B. Locality-Aware Expert Placement

With our distributed fine-tuning framework in place, the remaining challenge is to optimally allocate experts across available devices to minimize communication overhead between the master and worker processes. To address this, we formulate expert placement as an optimization problem, aiming to minimize the communication time required for each fine-tuning step.

Decision variables. Consider a distributed setting with N available computing devices, such as GPUs, spreading across multiple computing nodes. The MoE model to be fine-tuned consists of L MoE blocks, each containing E experts. We represent an expert distribution scheme as a three-dimensional binary tensor $X \in \mathbb{R}^{N\times L\times E}$, where

$$X_{n,l,e} \in \{0, 1\},$$

which means tensor elements $X_{n,l,e}$ takes the value 1 if the e-th expert of block l is assigned to worker process n, and 0 otherwise. For X to be a valid distribution scheme, the following requirement

$$\sum_{n=1}^{N} X_{n,l,e} = 1$$

should be satisfied to ensure that each expert can only be assigned to a single work process.

Optimization objective. Next, we estimate the communication time given the expert distribution scheme X. During fine-tuning, a total of K tokens are fed into the gating layer for processing where

$$K = \text{batch size} \times \text{sequence size}.$$

Since experts are distributed among different workers, we denote $K_{n,l}$ as the number of tokens sent to worker n for computation in MoE block l. The data transmitted between the master process and worker n for MoE block l is given by

$$D_{n,l} = \frac{bHK_{n,l}}{8}$$

bytes, where H is the feature size of this model, and b is the bit-depth for the feature.

Let B_n denote the bandwidth between the master process and worker n, which can be measured in advance. The value of B_n varies significantly depending on the location of the worker process. For instance, if worker n and the master process

are spawned on the same node but different GPUs, they can communicate with high-speed PCIe or NVLink connections with bandwidth up to tens even hundreds of Gbps. Conversely, if the worker process is located on a different machine, the bandwidth will be constrained by the network connection.

Besides dispatching the tokens to each worker, the master process receives output of the same size from each worker. Thus, the total communication overhead is doubled to $2D_{n,l}$. Thus, the communication time for worker n processing MoE block l is:

$$T_{n,l} = \frac{2D_{n,l}}{B_n} = \frac{bHK_{n,l}}{4B_n} \tag{5}$$

Note that prior to fine-tuning, we pass the dataset through the model to generate a probability matrix $P \in \mathbb{R}^{L\times E}$, where each element $P_{l,e}$ represents the probability of expert e being accessed in MoE block l. Due to expert locality, the probability matrix remains stable throughout the process. Therefore, we safely estimate the communication time as follows:

$$\mathbb{E}(T_{n,l}) = \frac{bH}{4B_n}\mathbb{E}(K_{n,l}) = \frac{bH}{4B_n}\sum_{i=1}^{E} X_{n,l,e}P_{l,e}K \tag{6}$$

During fine-tuning, the master process waits for all workers to finish their communication before proceeding to subsequent operations. Therefore, the expected total communication time for MoE block l is determined by the maximum of the individual worker communication times. Consequently, the total communication time for one fine-tuning step consisting of multiple MoE blocks is:

$$\sum_{l=1}^{L} \max\left(\mathbb{E}(T_{1,l}), \mathbb{E}(T_{2,l}), \ldots, \mathbb{E}(T_{N,l})\right), \tag{7}$$

which forms the optimization target for the expert placement problem.

Problem formulation. We are now ready to present the complete problem formulation as follows:

$$\min_{X} \quad \sum_{l=1}^{L} \max\left(\mathbb{E}(T_{1,l}), \mathbb{E}(T_{2,l}), \ldots, \mathbb{E}(T_{N,l})\right) \tag{8}$$

$$s.t. \quad X_{n,l,e} \in \{0,1\}, \tag{9}$$

$$\sum_{n=1}^{N} X_{n,l,e} = 1, \tag{10}$$

$$\sum_{l=1}^{L}\sum_{e=1}^{E} X_{n,l,e} \le C_n \tag{11}$$

Constraints (9) and (10) ensure that the results are binary indicators, such that each expert can be assigned to only one worker process. Constraint (11) implies that the number of the expert that worker n can host is limited by a constant number C_n to make sure the assigned experts won't exceed the GPU memory limitations in the runtime.

Due to the homogeneous nature of expert weights (*i.e.,* each expert has consistent input/output feature sizes and uniform weight matrices), the GPU memory consumption for each expert during fine-tuning remains constant. Consequently, the GPU memory required by a worker scales linearly with the number of experts it hosts. The capacity C_n of worker n can be obtained by dividing the total available GPU memory of worker n by the memory required for a single expert.

LP transformation. To efficiently solve the expert placement problem, we transform the original problem into a linear programming (LP) problem. Notably, constraints (11) and (10) are already linear. Therefore we just need to address the binary constraint (9) and the $\max$ function in the optimization target.

1) For the optimization target (8), referring to (6), we note that each element in the $\max$ function is linear. We can equivalently linearize it by replacing the $\max$ function with an auxiliary variable λ_l, and adding constraints to ensure each of the original linear expressions in the $\max$ function is less than or equal to λ_l.
2) For (9), we can relax the binary constraint and allow each element to float between 0 and 1. After solving the problem, we can round the results back to either 0 and 1.

In this way, the original problem becomes

$$\begin{aligned}
\min_{X} \quad & \sum_{l=1}^{L} \lambda_l \\
s.t. \quad & 0 \le X_{n,l,e} \le 1, \\
& \sum_{n=1}^{N} X_{n,l,e} = 1, \\
& \sum_{l=1}^{L}\sum_{e=1}^{E} X_{n,l,e} \le C_n, \\
& \frac{bH}{4B_n}\sum_{e=1}^{E} X_{n,l,e}P_{l,e}K \le \lambda_l
\end{aligned}$$

where the optimization target and all the constraints are linear. The original optimization problem is, therefore, transformed into a linear programming problem, and can be efficiently solved by off-the-shelf LP solvers.

The solution of the above problem yields a tensor with floating values between 0 and 1, which we need to convert back to binary values while respecting GPU memory constraints. This conversion can be accomplished through the following steps:

1) Round each relaxed value to binary by applying a threshold of 0.5, where any value above 0.5 becomes 1 and any value below or equal to 0.5 becomes 0.
2) Examine each worker process n and identify those where the number of assigned experts exceeds its capacity limit C_n. For these overloaded devices, remove expert assignments with the lowest relaxed values until the capacity constraint is satisfied.
3) Check for any experts that were left unassigned during the previous step. For each such expert, find a device that still has available capacity and showed the strongest affinity (*i.e.,* the highest relaxed value) for that expert in the original solution, then assign the component to that device.

This process ensures we obtain a feasible binary solution that closely approximates the optimal relaxed solution while satisfying all capacity constraints.

V. Evaluation

A. Evaluation Setup

Experimental environment. We implemented and deployed VELA in a real distributed environment, including three computation nodes, each equipped with two NVIDIA V100 GPUs, for a total of six GPUs. Each computation node has 72 GB of physical main memory, while the GPU has 32 GB of dedicated GPU memory. According to the measurement results, the GPUs within the same node communicate using internal links, offering a measured bandwidth of up to 18.3 GB/s. On the other hand, different nodes are linked using Ethernet connection with a bandwidth of 1.17 GB/s measured by iperf.

Throughout our experiments, we observed that over 18 TB of intermediate data, like features and gradients, is transmitted across different nodes. This substantial volume of data exchange emphasizes the critical importance of communication efficiency in distributed fine-tuning of MoE models.

Models and datasets. We evaluated VELA using two MoE models: `Mixtral`-8×7B [3] and `GritLM`-8×7B [18]. Mixtral-8×7B is widely considered an optimal balance between performance and model scale, making it the most commonly used MoE model. GritLM-8×7B is an enhanced version of Mixtral, fine-tuned on a generative representational instruction tuning dataset [18]. Even though using half precision, we still need to load over 80 GB of pre-trained model parameters into GPU memory to work with industry-level MoE models.

To demonstrate VELA's generalizability, we fine-tuned these MoE models using two different datasets, each corresponding to a distinct NLP task: 1) WikiText [19], which is pure text content used for language modeling. 2) Alpaca [4], where the data is organized into dialogues, representing the prevailing instruction tuning task [6].

Fine-tune settings. In our experiments, we employed LoRA [20], the most prevalent fine-tuning technique, to fine-tune the MoE models. These models were fine-tuned using mixed precision: pre-trained parameters were loaded in half precision, while other variables remained in full precision to maintain model performance.

Shen et al. [5] demonstrated that fine-tuning the gating mechanism leads to performance degradation. Therefore, in our experiments, we fine-tuned all the linear layers except for the gating mechanism, using a LoRA configuration of $r = 8$ and $\alpha = 16$. The batch size was set to 8, and all models were fine-tuned for 500 steps using the AdamW optimizer with the following hyperparameters: learning rate $3e-5$, beta values $[0.8, 0.999]$, epsilon $1e-8$, and weight decay $3e-7$.

Our evaluation excludes convergence results from fine-tuning since Vela maintains identical computation logic to single-device fine-tuning, despite the expert layers being distributed across different devices. Because the intermediate data is exchanged without precision loss, fine-tuning MoE models with Vela produces the same convergence results as traditional fine-tuning approaches.

Baselines. In our experiments, we primarily compared VELA with conventional expert parallelism. To ensure a fair comparison and exclude the effects of hardware acceleration, we implemented conventional expert parallelism strictly following Fig. 2. In this implementation, the experts of each MoE block were sequentially placed on GPUs, with the e-th expert of any MoE block assigned to the $e\%N$-th GPU while the other layers were replicated among all devices.

Beyond expert parallelism, we also validated the superiority of VELA by comparing it with different placement schemes within the same framework. To the best of our knowledge, VELA is the first work to study the expert placement problem in MoE fine-tuning. Consequently, we can only compare it with two baseline approaches: 1) sequential placement, which sequentially assigns experts to devices as in expert parallelism, but runs within VELA's framework, and 2) random placement, where all experts from all MoE blocks are randomly shuffled and assigned to different worker processes.

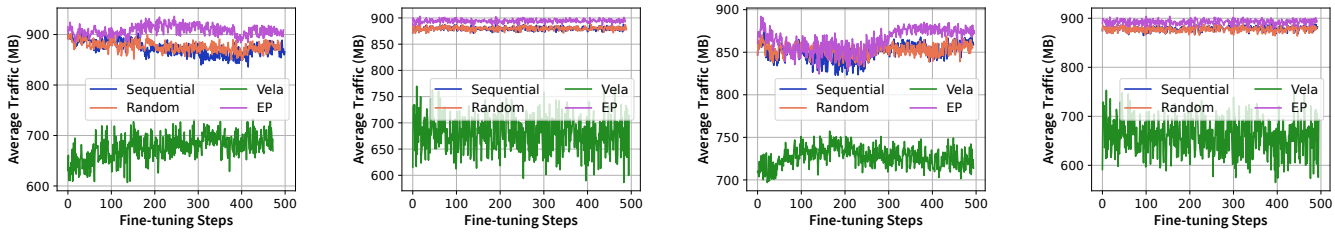
B. Evaluation Results

Communication efficiency. We first evaluate communication efficiency by examining the average cross-node communication traffic per node (referred to as external traffic). We track external traffic at each step to capture the dynamic characteristics of expert selection.

In our experiments, communication overhead primarily arises from token exchange. For baseline methods, approximately more than 2600 tokens are sent to external devices for each MoE block in one fine-tuning step. Each token has a feature size of 4096 and a bit depth of 16 bits, totally 16.4 MB. This data exchange occurs four times across all 32 layers in the `Mixtral` and the `GritLM` models: sending and gathering features in the forward pass, and sending and gathering gradients in the backward pass. Consequently, there is around $(16.4 \times 4 \times 32)/3 \approx 866$ MB of external traffic for token exchange per node for each fine-tuning step.

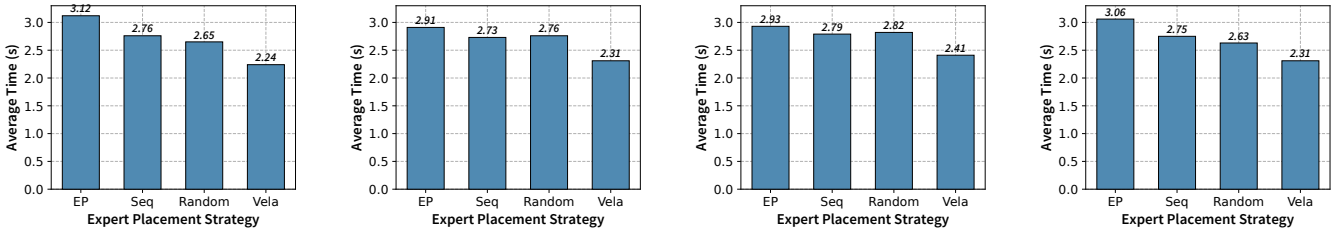
The communication traffic depends on the location of the experts. Since there is no optimization in baseline methods, expert parallelism, sequential and random expert placements have no advantage against each other. Thus, the communication overhead of these baseline methods is roughly the same as shown Fig. 5, while expert parallelism is slightly higher due to the gradient synchronization.

On the other hand, VELA achieves the lowest external traffic compared to all other methods across all experiment settings. This demonstrates the effectiveness of the locality-aware placement method. Compared with original expert parallelism, the optimized placement of VELA significantly reduces the communication overhead, and the performance gain varies depending on the dataset and model used. For



(a) `Mixtral` with WikiText. (b) `Mixtral` with Alpaca. (c) `GritLM` with WikiText. (d) `GritLM` with Alpaca.

Fig. 5. Cross-node traffic comparison with expert parallelism (EP) and other expert placement strategy.



(a) `Mixtral` with WikiText. (b) `Mixtral` with Alpaca. (c) `GritLM` with WikiText. (d) `GritLM` with Alpaca.

Fig. 6. Average time to complete one fine-tuning step. "EP" indicates expert parallelism and "Seq" represents the sequential placement strategy.

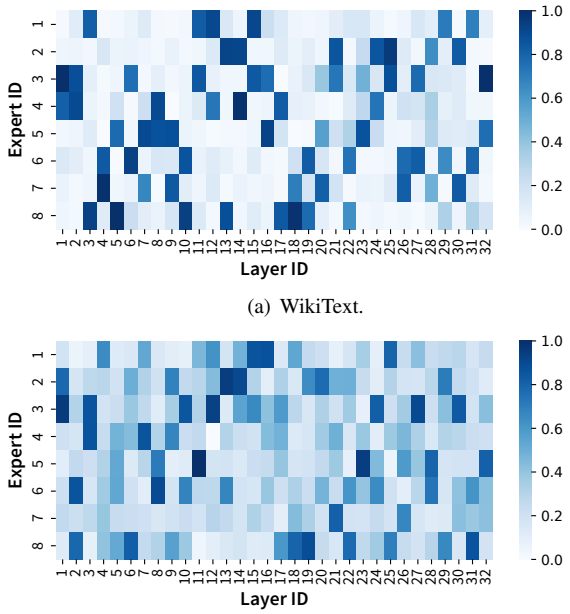
the WikiText dataset, VELA reduces communication overhead by an average of 18.1% to 25.3% compared to conventional expert parallelism. In contrast, the benefit on Alpaca dataset is relatively smaller, with communication reduction ranging from 17.3% to 20.1%.

It is worth noting that the benefit of VELA remains consistent throughout the entire fine-tuning process. Even though the external traffic of VELA increased slightly as shown in Fig. 5(a), it remains significantly lower than that of comparisons. These results indicate that the expert selection pattern remain stable throughout the fine-tuning process, further verifying our theoretical analysis of expert selection stability in fine-tuning.

Fine-tuning acceleration. The communication overhead directly affects fine-tuning speed, so we further evaluate efficiency by examining the average time spent on each fine-tuning step. The results are presented in Fig. 6.

Although the conventional expert parallelism has a similar communication overhead to sequential and random placements, their communication time is significantly distinct, due to different fundamental communication patterns. In expert parallelism, all-to-all communication is required to dispatch tokens to different devices. Consequently, all devices need to determine how many tokens they should receive from each other before performing the data transfer. In other words, token exchange in conventional expert parallelism is interrupted by a status synchronization process.

In contrast, VELA avoids this problem due to its master-worker architecture. Instead of requiring all-to-all synchronized communication, VELA uses a one-to-all pattern where the master process directly initiates data transfer with worker processes. Without such synchronization overhead,



(a) WikiText.

(b) Alpaca.

Fig. 7. Expert access frequency of `Mixtral` on different datasets.

our framework outperforms conventional expert parallelism in terms of speed. Especially, with the optimized expert placement, VELA accelerates each fine-tuning step from 20.6% in Fig. 6(b) to 28.2% in Fig. 6(a) which is even greater than the reduction ratio in the communication size due to the architectural difference.

Performance analysis. The benefit of VELA comes from leveraging the locality phenomenon of MoE models. To better

understand VELA's performance improvement, we visualized the expert access pattern in our experiments and showed it in Fig. 7.

To avoid redundancy, we only show the access frequency results for `Mixtral` as the other model exhibits a similar pattern. On both datasets, there are multiple popular experts spread across different layers. Interestingly, different datasets show different preference for expert selection. For example, the last expert in the third MoE block is extremely popular in WikiText, but rarely selected in Alpaca. This aligns with the common understanding that different experts specialize in processing tokens related to different domains.

From previous experiments, we observed that VELA performs better with the WikiText dataset than with Alpaca, and Fig. 7 reveals the underlying causes.

In WikiText, expert access is more concentrated on popular experts, as indicated by the large white areas in the heatmap. By grouping these popular experts together, VELA significantly reduces communication overhead in fine-tuning.

In contrast, the expert access is more uniformly distributed with Alpaca, as indicated by numerous light blue blocks in the heatmap. In this case, although VELA groups the popular experts, many tokens still need to be processed by experts located on other devices. This degrades the performance and was reflected in both the overall traffic reduction and the stability, as shown in Fig. 5.

VI. Related Work

Parameter-efficient fine-tuning. Model fine-tuning enables users to quickly adapt powerful LLMs to their specific tasks [21], [22]. However, due to the formidable size of LLM parameters, fine-tuning the entire parameter set is impractical. Consequently, parameter-efficient fine-tuning techniques have become widely adopted in the LLM era, such as freezing most parameters while retraining selected layers [23], or introducing auxiliary adapters [14], [24]–[26]. Though these methods reduce the number of trainable parameters, the large scale of pre-trained parameters still poses significant challenges. As a result, quantization techniques are typically applied to pre-trained parameters in fine-tuning tasks, compressing them from 32 bits to as low as 4 bits [15], [27]–[29] at the cost of performance degradation.

However, as LLMs especially MoE models continue to grow in scale, it becomes impossible to host the entire model on a single machine, even with parameter-efficient techniques and quantization. As a result, distributed training is required, even for fine-tuning tasks.

Training with MoE. Expert parallelism [2] was introduced as a method to distribute MoE computations and quickly became the foundational computational paradigm for MoE models. This approach has been integrated into most mainstream training frameworks, including Megatron-LM [9], DeepSpeed [8], and ColossalAI [30]. However, these frameworks are primarily designed for building and pre-training models from scratch, making them less suitable for fine-tuning tasks. First, these frameworks have their own specific model definitions and MoE implementations, requiring significant engineering effort from end-users to convert pre-trained weights to fit these frameworks. Second, and more critically, these frameworks only allow users to specify the degree of expert parallelism, but do not offer the flexibility for manual control over expert distribution that VELA provides.

Some dedicated fine-tuning frameworks, such as LLama-Factory [31], do offer interfaces for fine-tuning MoE models. However, these solutions typically assume that the model can fit, or be quantized to fit, on a single machine, which is impractical for most users.

Locality in MoE models. The expert locality phenomenon was first observed in the pre-training stage [2]. Since the model was initialized from scratch, all expert parameters are waiting to be trained. To facilitate effective pre-train the MoE model, load balancing terms [2], [10] and mechanisms [16] are typically introduced to regulate the behavior of the expert selection, ensuring that all expert layers are sufficiently trained. However, once the model is fully pre-trained, fine-tuning the gating mechanism could potentially damage the learned representations, leading to a performance degradation [5]. This fundamental difference between pre-training and fine-tuning stages means that methods developed for pre-training cannot be adopted during fine-tuning. Instead of avoiding expert locality, VELA aims to leverage it to improve fine-tuning efficiency.

Expert locality is also considered in the inference systems of MoE models in a way similar to VELA. For instance, Lina [11] estimates the expert popularity to dynamic schedule the resources during inference. Fiddler [12] and MoE-Infinity [13] utilize the expert access frequencies to prefetch and offloads some experts for better inference latency. In these systems, MoE models remain fixed during inference, making expert selection a static process. In contrast, the model behavior evolves over time during fine-tuning. To the best of our knowledge, VELA is the first work that provides a theoretical analysis on the dynamic features of expert selection during fine-tuning.

VII. Concluding Remarks

With the increasing popularity of LLMs, fine-tuning these models to adapt to user application scenarios becomes a common yet challenging practice, especially when dealing with MoE models at an unprecedented scale. In this paper, we systematically study the expert locality phenomenon in MoE models, especially focusing on the analysis of its dynamic features during fine-tuning. Our analysis shows that pre-trained MoE models tend to maintain their high-confidence choices of experts even when the model parameters are further optimized. Based on these insights, we introduce VELA, a novel locality-aware fine-tuning framework for MoE models. VELA strategically optimizes expert placement in a distributed environment, leveraging expert locality to minimize communication overhead. Finally, we validate both the accuracy of our analysis and the efficacy of the VELA framework with extensive experiments.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[2] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=qrwe7XHTmYb

[3] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of Experts," *arXiv preprint arXiv:2401.04088*, 2024.

[4] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford Alpaca: An Instruction-following LLaMA Model," 2023.

[5] S. Shen, L. Hou, Y. Zhou, N. Du, S. Longpre, J. Wei, H. W. Chung, B. Zoph, W. Fedus, X. Chen, T. Vu, Y. Wu, W. Chen, A. Webson, Y. Li, V. Y. Zhao, H. Yu, K. Keutzer, T. Darrell, and D. Zhou, "Mixture-of-Experts Meets Instruction Tuning: A Winning Combination for Large Language Models," in *International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=6mLjDwYte5

[6] S. Zhang, L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu *et al.*, "Instruction Tuning for Large Language Models: A Survey," *arXiv preprint arXiv:2308.10792*, 2023.

[7] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7B," *arXiv preprint arXiv:2310.06825*, 2023.

[8] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale," in *International Conference on Machine Learning*. PMLR, 2022, pp. 18 332–18 346.

[9] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro *et al.*, "Efficient Large-Scale Language Model Training on GPU Clusters using Megatron-LM," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[10] W. Fedus, B. Zoph, and N. Shazeer, "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[11] J. Li, Y. Jiang, Y. Zhu, C. Wang, and H. Xu, "Accelerating Distributed MoE Training and Inference with Lina," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 945–959.

[12] K. Kamahori, Y. Gu, K. Zhu, and B. Kasikci, "Fiddler: CPU-GPU Orchestration for Fast Inference of Mixture-of-Experts Models," *arXiv preprint arXiv:2402.07033*, 2024.

[13] L. Xue, Y. Fu, Z. Lu, L. Mai, and M. Marina, "Moe-Infinity: Activation-aware Expert Offloading for Efficient MoE Serving," *arXiv preprint arXiv:2401.14361*, 2024.

[14] E. J. Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "LoRA: Low-Rank Adaptation of Large Language Models," in *International Conference on Learning Representations*, 2021.

[15] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," *arXiv preprint arXiv:2305.14314*, 2023.

[16] M. Zhai, J. He, Z. Ma, Z. Zong, R. Zhang, and J. Zhai, "SmartMoE: Efficiently Training {Sparsely-Activated} Models through Combining Offline and Online Parallelization," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 961–975.

[17] A. Karpathy, "Char-RNN," https://github.com/karpathy/char-rnn, 2015.

[18] N. Muennighoff, H. Su, L. Wang, N. Yang, F. Wei, T. Yu, A. Singh, and D. Kiela, "Generative Representational Instruction Tuning," 2024.

[19] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer Sentinel Mixture Models," 2016.

[20] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, "PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods," https://github.com/huggingface/peft, 2022.

[21] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving Language Understanding by Generative Pre-training," 2018.

[22] J. D. M.-W. C. Kenton and L. K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[23] E. B. Zaken, Y. Goldberg, and S. Ravfogel, "BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022, pp. 1–9.

[24] Z. Hu, Y. Lan, L. Wang, W. Xu, E.-P. Lim, R. K.-W. Lee, L. Bing, and S. Poria, "LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models," *arXiv preprint arXiv:2304.01933*, 2023.

[25] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for NLP," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2790–2799.

[26] X. L. Li and P. Liang, "Prefix-Tuning: Optimizing Continuous Prompts for Generation," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, 2021, pp. 4582–4597.

[27] 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," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=r1gs9JgRZ

[28] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=dXiGWqBoxaD

[29] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit Optimizers via Block-wise Quantization," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=shpkpVXzo3h

[30] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You, "Colossal-ai: A Unified Deep Learning System for Large-Scale Parallel Training," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 766–775.

[31] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, Z. Luo, Z. Feng, and Y. Ma, "LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Bangkok, Thailand: Association for Computational Linguistics, 2024. [Online]. Available: http://arxiv.org/abs/2403.13372