Scaling Beyond the GPU Memory Limit for Large Mixture-of-Experts Model Training

Yechan Kim [*1] Hwijoon Lim [*2] Dongsu Han [1 2]

Abstract

Mixture-of-Experts (MoE) is a powerful technique for enhancing the performance of neural networks while decoupling computational complexity from the number of parameters. However, despite this, scaling the number of experts requires adding more GPUs. In addition, the load imbalance in token load across experts causes unnecessary computation or straggler problems. We present ES-MoE, a novel method for efficient scaling MoE training. It offloads expert parameters to host memory and leverages pipelined expert processing to overlap GPU-CPU communication with GPU computation. It dynamically balances token loads across GPUs, improving computational efficiency. ES-MoE accelerates MoE training on a limited number of GPUs without degradation in model performance. We validate our approach on GPT-based MoE models, demonstrating 67× better scalability and up to 17.5× better throughput over existing frameworks.[1]

1 Introduction

Mixture-of-Experts (MoE) (Shazeer et al., 2017; Du et al., 2022; Riquelme et al., 2021; Jiang et al., 2023) integrate multiple smaller sub-models or "experts" within a comprehensive ensemble model. The key enabling factor behind their success lies in the incorporation of learnable gating networks, which efficiently allocate input tokens to different experts. It improves the performance and scalability of models without increasing the computational complexity (Clark et al., 2022; Hwang et al., 2022), effectively *decoupling* the computational cost from the parameter size.

Parameters in MoE models can easily exceed billions, requiring a significant amount of memory for training. To train such models, expert parallelism (Fedus et al., 2022; Lepikhin et al., 2020) has become a de facto standard, wherein each GPU exclusively handles one or more experts. This assigns experts across GPUs while replicating non-expert parameters on each GPU, allowing multiple experts to run concurrently within a single GPU.

Recent research has been pushing MoE models beyond 128 experts to improve model accuracy (Puigcerver et al., 2023; Clark et al., 2022; Koishekenov et al., 2022; Rajbhandari et al., 2022; Fedus et al., 2022). However, scaling MoE models requires additional GPUs due to the increased memory footprint, even though the computational complexity remains the same. Thus, existing solutions for training large MoE models (e.g., FairSeq, DeepSpeed-MoE, Tutel) demand a large array of GPUs, limiting the ability of many researchers to train or even refine these models. For example, training MoE-L with 8 experts is feasible with four A100 GPUs, but scaling it to 128 experts demands 52 GPUs, which is a significant hurdle to most academic researchers.

To address the problem, ES-MoE improves the scalability of MoE models beyond the GPU memory limit and enhances the training efficiency of large MoE models. It is the first work to target the settings where the memory footprint of MoE expert models exceeds the GPU memory capacity. Our approach involves offloading expert parameters and optimizer states to the host, which reduces the GPU memory footprint, allowing it to process larger mini-batches within limited GPU memory for greater throughput. It facilitates the flexible assignment of experts to GPUs, providing the chance to balance the computational load across GPUs. However, realizing the idea involves handling new challenges:

- Unlike traditional systems in which an expert is statically assigned to a GPU, ES-MoE adopts dynamic scheduling wherein experts are placed on GPUs based on the output of the gating network. This prevents prefetching the experts onto GPUs, potentially causing the pipeline stalls.
- With optimizer states stored in CPU memory, transferring them back to GPU is expensive. Thus, model optimization must operate on the CPU, which takes more time.

[*]Equal contribution [1]Kim Jaechul Graduate School of AI, KAIST, Daejeon, Republic of Korea [2]School of Electrical Engineering, KAIST, Daejeon, Republic of Korea. Correspondence to: Dongsu Han <dhan.ee@kaist.ac.kr>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]The source code is available at `https://github.com/kaist-ina/es-moe`.

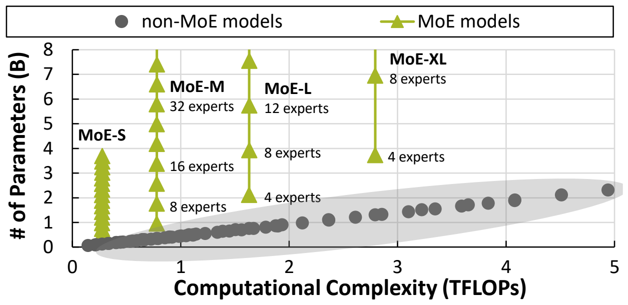


Figure 1: MoE models decouple the computational complexity from the number of parameters.

- Scaling the number of experts exacerbates token load imbalance. This disparity leads to the *straggler problem* (Harlap et al., 2016). Although techniques such as sparse matrix multiplication (Gale et al., 2022) have been proposed, existing designs do not scale well.

Our design resolves the challenges by introducing dynamic expert placements, and pipelined CPU optimizations, while offloading expert parameters for virtually unlimited scaling. ES-MoE employs expert-level scheduling that strategically manages expert parameter uploads to enable pipelined expert processing that overlaps the GPU-CPU communication with expert computation to maximize the GPU utilization. In contrast to generic pipelining methods that operate on a layer-by-layer basis and wait for the entire layer to finish, our expert-level pipelining starts optimization as soon as an individual expert finishes its computation. In addition, ES-MoE balances the load across GPUs by managing the placement of experts considering their load.

We provide a comprehensive evaluation of the scalability and throughput of ES-MoE with GPT-based (Radford et al., 2019) MoE models. ES-MoE accommodates up to 66.5 times more experts than existing frameworks and is able to fine-tune 15B parameter language models with just 4 GPUs. It completed the fine-tuning on 100 M tokens in 6.5 hours using datasets (Williams et al., 2017) containing 16 M tokens to significantly improve the accuracy. It achieves a notable increase in throughput of up to 11.6× over generic offloading methods, and up to 2.12× compared to the state-of-the-art method, Tutel (Hwang et al., 2022).

2 Related Work

Mixture-of-Experts (MoE) is a machine learning architecture that has gained traction for its high scalability. MoE models, leveraging a learnable gating network, intelligently route input tokens to one or more of the most appropriate sub-models, each termed an 'expert'. These models are typically built on top of transformer-based models, where the feed-forward network in each transformer is replaced by a combination of a gating network and multiple experts.

Expert parallelism (Fedus et al., 2022; Lepikhin et al., 2020), a variant of model parallelism (Chilimbi et al., 2014; Shazeer et al., 2018; Huang et al., 2019), has become a standard approach in training MoE-based models.This approach distributes experts across multiple GPUs, assigning one or more experts to each GPU, while replicating non-expert parameters on every GPU. During training, input tokens are exchanged between GPUs to ensure alignment with the appropriate experts as determined by the gating network. This distribution not only improves computational efficiency but also significantly enhances the scalability of MoE models.

To mitigate the cost associated with token exchanges, recent works (He et al., 2022; Liu et al., 2023) propose a data-centric design in which tokens remain in GPUs and experts are exchanged instead, under the premise that moving experts is less costly than exchanging tokens in an all-to-all fashion. However, this premise is only valid in environments with large GPU clusters and abundant memory. With limited GPU memory, the trainable batch size is limited, making expert exchanges more costly than token exchanges.

Offloading experts to CPU memory. Several studies suggest offloading certain states to the CPU (Rhu et al., 2016; Huang et al., 2020; Ren et al., 2021; Wang et al., 2018) for generic models. Notable works (Ren et al., 2021; Rajbhandari et al., 2020) have proposed offloading of model parameters and optimizer states to CPU memory. Offloading optimizer states to CPU memory is particularly beneficial in reducing the GPU memory footprint, as optimizer states can occupy up to six times more memory than the model itself[2]. However, the downside is that CPU-side optimization is orders of magnitude slower, even in the state-of-the-art implementation (Rajbhandari et al., 2022)

Load imbalance. Load balancing across the experts is essential in ensuring optimal model accuracy and efficient GPU resource utilization. With expert parallelism, an uneven distribution of tokens across the experts results in an unbalanced workload per GPU, which causes some GPUs to become stragglers (Harlap et al., 2016). To mitigate imbalance, a common practice is to incorporate an auxiliary imbalance loss to promote balanced token distribution (Shazeer et al., 2017; Fedus et al., 2022). Despite the practice, the load imbalance is still common (Zhou et al., 2022).

Many state-of-the-art frameworks (Hwang et al., 2022; He et al., 2022; Lepikhin et al., 2020; Fedus et al., 2022) use batched matrix multiplication, necessitating uniform input batch sizes across all experts. To meet this requirement, batches are typically zero-padded. The greater the load imbalance, the more zero padding is required to equalize batch sizes. However, since these placeholders do not con-

[2]Adam optimizer (Kingma & Ba, 2014), one of the most common optimizer, requires 12 bytes for the optimizer state per parameter, while the model parameter itself only require 2 bytes (`fp16`).

tribute to the actual training, they lead to computational inefficiency. In addition, batched matrix multiplication obligates all experts to be loaded into GPU memory and uses a large dispatch mask for token reordering, further restricting the batch size and even causing out-of-memory (OOM).

An alternative strategy is token dropping (Fedus et al., 2022; Zoph et al., 2022), where a limit is set on the maximum number of tokens per expert. This approach, however, can lead to dropping a significant fraction of tokens (more than 40% as reported by (Zhou et al., 2022)), which even affects model convergence (Rajbhandari et al., 2022).

3 Problem and Motivation

MoE models can be easily scaled to attain enhanced model performance by adding more experts but without increasing the computational cost, as shown in Figure 1. GPT-L has 760 million parameters, but its MoE counterpart, MoE-L can reach 2.11 billion parameters with 4 experts and 14.8 billion parameters with 32 experts, while having the same computational cost as GPT-L.

However, scaling MoE-based models with existing frameworks, in practice, presents challenges for two key reasons. First, although the model inherently decouples computational complexity from the model size, the underlying system does not support decoupling computation from memory. As a result, increasing the number of experts requires adding more GPUs, which is not always a viable option for many researchers. For instance, training MoE-L with 8 experts is feasible with only four A100 GPUs, scaling the model to 128 experts requires 52 GPUs, substantially increasing the barrier. Second, scaling the number of experts exacerbates existing inefficiencies in computation, which further limits GPU utilization. We expose the issues in greater detail.

Large memory footprint. Existing systems demand large amounts of GPU memory for the following reasons: First, they load all experts on GPUs as they execute the expert computations simultaneously. Adding more experts increases the GPU memory requirements for model memory and optimizer states. Second, the use of batch matrix multiplication in expert computation requires creating a dispatch mask to reorder the tokens so that they can be sent to the correct expert according to the decision of the gating network. However, this dispatch mask is essentially a *huge* table that maps tokens to experts with a dimension of (number of tokens after zero-padding)$\times$(number of tokens) (Huang et al., 2023), occupying substantial memory. For example, training MoE-L with a batch size of 32 and 1024 tokens per batch requires *at least* 48 GiB for the mask.

GPU underutilization. Scaling up the number of experts results in GPU under-utilization for the following reasons: First, as the input batch (input tokens) gets distributed across experts, increasing the number of experts proportionally reduces the number of tokens per expert. This decrease in the token count per expert in MoE models leads to lower GPU utilization. Second, the demand for large memory creates memory pressure and limits the size of microbatches; e.g., training MoE-M with 4 experts and four A100 40GB GPUs allows the microbatch size of 8, but increasing to 32 experts drops the size to 2, reducing training throughput by 46.2%. Finally, increasing the number of experts exacerbates the token load imbalance, which can be quantified by the fraction of zero-padding required to evenly distribute the load across all experts relative to the total workload. Empirical data from training the MoE-L model show that with 8 experts, the token load imbalance in the initial training phase reaches up to 17% (i.e. 17% of computations are used for computing zero padding) but with 32 experts, it increases to 39%.

Limited GPU availability. Securing a large number of GPUs for model training is a significant challenge for many researchers, especially those in academic settings and small organizations (Holmes & Gardizy, 2023; Kuperman, 2023). Most cloud providers impose stringent resource quotas on individuals, due to the limited availability of GPUs (Google Cloud, 2024). The situation worsens during peak demand periods, making it difficult for individuals to obtain even a few GPUs. This barrier often prevents researchers from the opportunity to train or even fine-tune large MoE models.

4 ES-MoE Design

Goals. We present a design of ES-MoE that tackles the challenges described in §3 in scaling training MoE-based models. Specifically, ES-MoE achieves the following goals:

- **Scalable w.r.t. the number of experts:** ES-MoE must be able to scale to a large number of experts without having to add more GPUs.
- **Improve efficiency in training**: It should improve the GPU utilization by supporting larger mini-batches, mitigating the token imbalance, and minimizing any overhead introduced from scaling.
- **Preserve model accuracy**: It must maintain the integrity of a model by maintaining mathematical equivalence to preserve the model accuracy.

Key approach. The key idea of ES-MoE is to offload expert parameters from the GPU to host memory and storage, which allows us to accommodate larger models than the GPU memory permits. With our careful pipelining of expert loading and computation, we effectively minimize the overhead of copying parameters to and from GPUs (§4.1).

The offloading of experts opens up two new opportunities for enhancing the training throughput. First, the offloaded experts alleviate the GPU memory pressure and free memory, which can be used to increase the batch size for training. This allows ES-MoE to fully utilize the parallelism in GPUs

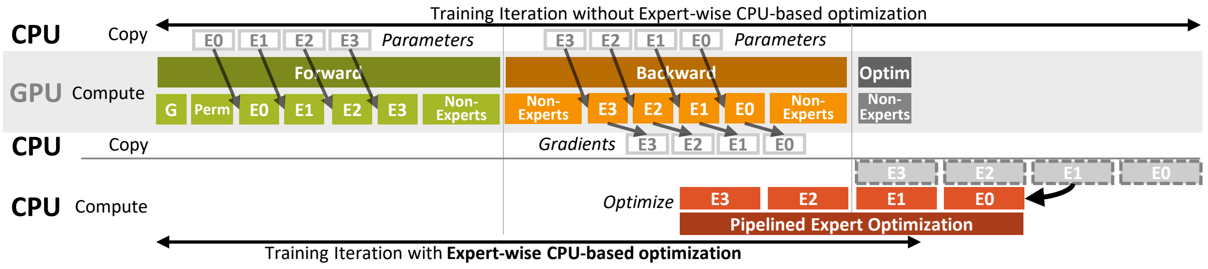

Training Iteration without Expert-wise CPU-based optimization
CPU
Copy
E0 E1 E2 E3 Parameters
E3 E2 E1 E0 Parameters
GPU Compute
Forward
Backward
Optim
G Perm E0 E1 E2 E3 Non-Experts
Non-Experts E3 E2 E1 E0 Non-Experts
Non-Experts
CPU Copy
Gradients E3 E2 E1 E0
CPU Compute
Optimize E3 E2 E1 E0
Pipelined Expert Optimization
Training Iteration with Expert-wise CPU-based optimization

Figure 2: ES-MoE overlaps expert's computation and communication and pipelines CPU optimization at the expert granularity to overlap with the backward pass of the layer. E0, ..., E3 indicate experts in the same layer. `G` and `Perm` respectively indicate the gating network and token permutation phase.

and thus increase the training speed. Second, the experts have to be dynamically loaded to GPUs for computation, which gives us an opportunity to place experts on the GPUs in a way that evenly balances the load across GPUs without having to use zero-padding (§4.2). Finally, ES-MoE adapts the degree of offloading based on the number of experts per GPU and the maximum number of experts a GPU can accommodate in its memory (§4.3).

4.1 Expert-wise Offload and Processing

ES-MoE offloads expert parameters and optimizer states, while efficiently scheduling the upload, download, and optimization of individual experts. It maintains only the followings states in the GPU memory: non-expert parameters, parameters of the expert being used at the moment, and their activations. The remaining offloaded state is kept on either the host memory or storage. The offloading allows ES-MoE to scale the number of experts beyond the GPU memory limit, unlike layer-wise offloading (Ren et al., 2021; Rajbhandari et al., 2021) which causes out-of-memory when the experts in a layer exceeds the GPU memory capacity.

Pipelined expert processing. A key challenge in offloading experts is that its upload must be carefully scheduled so that they minimize GPU stalls. Training an MoE block starts with tokens passing through the gating network (`G` in Figure 2). Based on the output of the gating network, ES-MoE places experts on GPUs to evenly distribute load across GPUs, calculated by the expert placement module (detailed in §4.2). After the decision, ES-MoE uploads the experts to the GPUs according to the decision. However, this poses a challenge because the expert placement on GPUs can only be determined after the gating network is executed, leaving too little time to upload all experts assigned to a GPU, leading to potential GPU stalls.

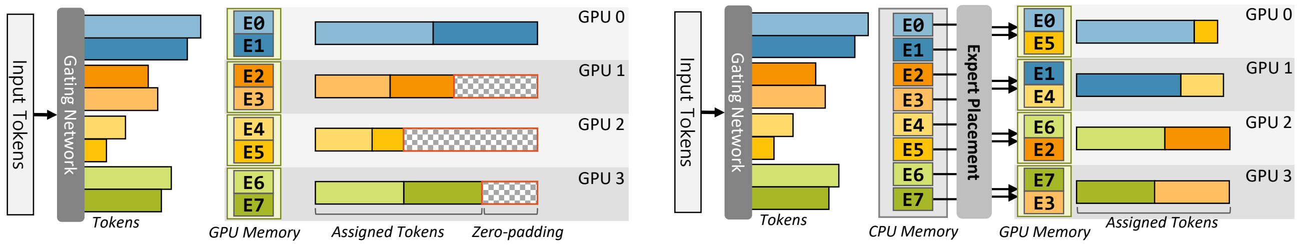
To address this, ES-MoE implements a careful pipeline of the tasks to be completed following the output of the gating network. The tasks include token permutation, expert upload, and expert processing. Before being fed to the experts, the tokens are reordered in the permutation phase for the token exchange across GPUs. Although the token permutation time is too short to complete the upload of multiple experts, it usually gives sufficient time for transferring a single expert. Thus, ES-MoE overlaps the permutation phase (`Perm` in Figure 2) with the time required to upload the first expert. Subsequent experts are then processed sequentially, ensuring concurrent expert computation and upload, reducing the perceived expert loading time.

Supporting larger batches. ES-MoE differs from other GPU-based schemes (e.g., Tutel) in that it does *not* use batched matrix multiplication. This is due to the incompatibility of expert-wise offloading with batched matrix multiplication, which requires all experts to be loaded into GPU memory. Interestingly, not using batched matrix multiplication brings a significant benefit in reducing the GPU memory footprint. Instead of creating a large dispatch mask required for the batched matrix multiplication, ES-MoE sequentially assigns tokens to the target expert based on the gating network decision, saving memory substantially.

The GPU memory saved by this sequential approach allows for larger batch sizes, which results in improved throughput. For example, when training MoE-L, this approach allows ES-MoE to handle $8\times$ larger microbatches, which translates into $3.1\times$ throughput improvements (Table 3).

Comparing memory-saving techniques, we can compare ES-MoE's sequential approach with the sparse batched matrix multiplication introduced by MegaBlocks (Gale et al., 2022). MegaBlocks reduces the memory required for the dispatch mask and improves throughput by eliminating zero padding. MegaBlocks particularly becomes efficient as the number of experts increases (i.e., the batched matrices become sparser). However, MegaBlocks is only useful when the GPU memory is abundant since it requires all experts to be loaded into GPU memory for batched matrix multiplication. In contrast, ES-MoE's sequential approach doesn't require all experts to be loaded into GPU memory at the same time, allowing training of larger batches.

Expert-wise CPU-based optimization. Due to the large memory footprint of optimizer states, the use of a CPU-based optimizer is inevitable. However, the CPU-based optimizer is extremely slower compared to a GPU-based



(a) Traditional MoE training (static expert placement)

(b) ES-MoE (dynamic expert placement)

Figure 3: Example of training a MoE model with 4 GPUs and 8 experts. E0 to E7 indicates separate experts in the same layer. Dynamic expert placement of ES-MoE eliminates the need for zero padding, achieving high efficiency.

optimizer (e.g., 31x slower with Adam). On top of this, existing frameworks apply optimizer at the granularity of the entire model (Hwang et al., 2022; Lepikhin et al., 2020) or the entire layer (Pudipeddi et al., 2020). As the number of experts in MoE-based models grows, the processing time of CPU-based optimization increases, resulting in GPU stalls.

To address the challenge, ES-MoE introduces expert-wise CPU-based optimization, which enables concurrent CPU optimization and GPU computation. ES-MoE runs the optimizer at the granularity of individual experts—ES-MoE initiates the optimizer for each expert as soon as each expert completes its backward pass (Figure 2), instead of waiting for the entire layer to complete the backward pass. This is especially useful for models with many layers and experts since the optimization of layers close to the output can be hidden by the GPU's processing of other layers.

Note this is different from delayed update used in ZeRO-Offload (Ren et al., 2021), where the parameter update is delayed to overlap CPU optimizations with the next iteration. Although delayed update hides the latency of CPU-based optimization, it introduces "staleness", affecting final model accuracy (Dai et al., 2018). In contrast, ES-MoE performs updates at the granularity of the expert without introducing staleness and maintains the original model accuracy.

Offloading experts to SSD. Although CPU RAM is expandable, this expandability does not apply to cloud environments, where most researchers train their models. Cloud providers offer only predetermined sets of GPU, CPU, and RAM configurations for each type of virtual machine instance. For example, AWS instance type `p3.4xlarge` offers four V100 GPUs, but only provides 244 GiB of CPU RAM. To access larger amounts of RAM, researchers must opt for more expensive higher-tier instances, resulting in increased costs. Instead, cloud providers offer highly scalable storage solutions, such as AWS Elastic File System (EFS), which can scale almost without limit.

To exploit highly scalable storage, ES-MoE extends its offloading strategy to include fast storage devices, such as SSDs, allowing it to scale beyond the CPU memory capacity. To enable this, ES-MoE uses a virtual memory (VM)-like method with prefetching; it maintains a limited set of experts in CPU memory, and evicts them using the Least Recently Used (LRU) cache policy. The key to this system is the prefetching of experts using the predictable sequence of forward and backward passes in training. This enables efficient expert handling between CPU memory and storage without bottlenecks and is superior to using naïve VM for two reasons: First, the naïve approach lacks application-level knowledge and can fetch the next expert only after a page fault, which may stall the training. Second, it allows more efficient data transfers, as ES-MoE prefetches experts onto DMA-able non-pageable (pinned) memory area. When using a naïve VM approach, experts must be copied from the pageable memory to the pinned memory.

4.2 Dynamic Expert Placement on GPUs

Existing work on training MoE suffers from load imbalance across GPUs that arise from the skewed distribution of tokens across experts (Figure 3(a)). This is because experts are fixed on the GPU memory, whereas the distribution of tokens changes over time. However, in ES-MoE, because experts are loaded on the GPUs on demand and each GPU processes multiple experts sequentially, the placement of experts can be adapted to the distribution of tokens on a per-batch basis such that the aggregate load on a GPU is balanced, as shown in Figure 3(b). Our dynamic expert placement effectively decouples the load-balancing decision from the token routing decision.

We now explain how ES-MoE decides the placement of n experts on k GPUs, where n is often much greater than k. Distributing experts across GPUs to balance the token load is similar to minimum makespan scheduling (Vazirani, 2001) whose goal is to minimize the finishing time of the last task. This is known as a strong NP-hard problem (Garey, 1997). To ensure fast expert upload and token transfer, we require an approximate solution. We adopt a greedy scheduling algorithm from (Graham, 1969) that gives a $\frac{4}{3}$-approximation for the problem. This algorithm sorts the experts by the expert processing times, modeled as the maximum of the expert upload time plus the expert processing time determined by the number of tokens assigned. It then assigns each expert to the group with the lowest accumulated processing times.

This algorithm runs efficiently on the CPU in a short time (< 2.69 us). Considering that expert computation and upload take a few milliseconds, running the expert placement algorithm does not block the training process. The complexity of the algorithm is $O(m * logn + m * logm)$, where m is the number of experts and n is the number of GPUs. However, in most cases, $n << m$ and m is at most hundreds, so the actual runtime of this algorithm is trivial.

4.3 Adaptive Offloading

This section introduces additional optimizations regarding expert offloading. While the CPU offloading and pipelined expert processing are useful in scaling MoE models, they do not provide performance benefits when training smaller models that fit within the aggregate GPU memory and/or when the number of tokens allocated to each expert is so small that its computation time is too short to hide the delay of uploading another expert. To automatically attain the best performance in any setting, ES-MoE introduces adaptive offloading, in which the degree of offloading is determined based on the number of experts per GPU and the maximum number of experts a GPU can accommodate in its memory.

GPU only. In the limited scenario where all expert parameters and optimizer states fit within the GPU memory, ES-MoE operates with all experts kept within the GPU memory, achieving training throughput gain from the zero-padding elimination. As ES-MoE does not offload experts, we cannot obtain benefits coming from the dynamic expert placement, thus the load across GPUs may vary as in other baselines. However, ES-MoE still outperforms other baselines as it saves GPU memory by avoiding creating large dispatch masks and allows training with larger batches, which contributes to higher GPU utilization.

Offload with expert pinning. As the number of experts increases, expert loading time relative to the processing time also increases, potentially causing GPU stalls. To mitigate this, ES-MoE pins a few heavily used experts on each GPU. Pinning experts allows greater time to dynamically load other experts and reduces the number of expert I/O, thus improving the GPU utilization. The token load of an expert does not vary much from one iteration to the next in the training phase. Thus, ES-MoE pins the top n_p experts to each GPU from the previous iteration and use dynamic placement for remaining experts. We empirically set n_p as 25% of the number of experts in each GPU.

5 Evaluation

We evaluate ES-MoE against several state-of-the-art training frameworks, including a generic CPU offloading framework and and those optimized for training MoE-based models. Our main findings are as follows:

- ES-MoE shows excellent scalability with an increasing number of experts and model size, allowing training of 64-expert MoE-L with only 4 GPUs, while all other frameworks suffer from OOM.
- ES-MoE enhances training throughput up to **17.5**× compared to the framework that supports offloading and **2.13**× compared to existing frameworks optimized for training MoE models, all while preserving mathematical equivalence to original training semantics.

Implementation. ES-MoE is implemented on top of the Fairseq framework (Ott et al., 2019). For CPU-based optimization, we adopt the efficient CPU Adam optimizer by DeepSpeed (Microsoft, 2023). We implement ES-MoE with 3.3k lines of Python and 3.0k lines of C++ code.

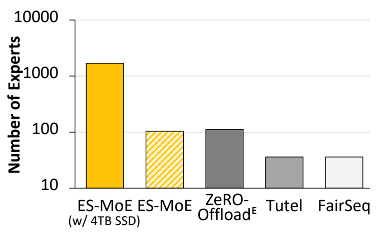
Setup. We conduct our experiment on a GPU node with four NVIDIA A100 with 40 GB of GPU memory and an AMD EPYC 7543 processor (32 cores) and 512 GiB DDR4 CPU memory. The node uses PCIe 4.0 for CPU-GPU communication and NVLink (600 GB/s) for GPU-GPU communication, enabling efficient token exchange between GPUs.

Baselines. We compare ES-MoE with frameworks optimized for training MoE-based models, including Fairseq's Gshard (Lepikhin et al., 2020) and Tutel (Hwang et al., 2022). In addition, to compare with a CPU offloading scheme, we use a modified version of ZeRO-Offload (Ren et al., 2021). The original ZeRO-Offload, which offloads parameters layer by layer, fails to handle a large number of experts causing OOM. Thus, we extend it to support expert-wise offloading and name this version Zero-Offload^E. For all frameworks, we enable activation checkpointing (Griewank & Walther, 2000; Chen et al., 2016).

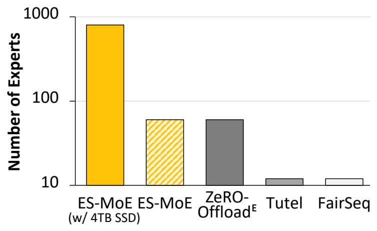
Models. We evaluate ES-MoE using GPT-derived Mixture-of-Experts (MoE) language models, MoE-S, MoE-M, MoE-L, and MoE-XL introduced in Gale et al. (2022). We provide details of the models, including their hyperparameters in Appendix A.1. We train the models using the WikiText-103 dataset with a vocabulary size of 51,200. We employ the top-1 gating mechanism that directs tokens to the top-ranked expert. We incorporate the imbalance loss technique from Fedus et al. (2022), with a coefficient of 0.01, to align with previous research (Fedus et al., 2022; Rajbhandari et al., 2022; Lepikhin et al., 2020). In line with standard practices, we apply mixed precision training (Micikevicius et al., 2017), using 16-bit (fp16) for parameters and 32-bit (fp32) for the optimizer state, enhancing numerical stability. We maintain a per-device batch size of 32; for frameworks other than ES-MoE, we employ smaller microbatches to prevent OOMs during training and use gradient accumulation.

# Experts	Model	Param.	Training Throughput (words/s)				
			Zero-Offload^E	FairSeq	Tutel	ES-MoE	
8	MoE-S	521 M	46321	82631	123152	**163217**	**(3.52x)**
	MoE-M	1.76 B	18784	27772	57605	**65352**	**(3.48x)**
	MoE-L	3.93 B	8677.3	21542	25526	**38173**	**(4.40x)**
16	MoE-S	974 M	24469	60142	96314	**158904**	**(6.49x)**
	MoE-M	3.37 B	6987.7	23705	43480	**63150**	**(9.04x)**
	MoE-L	7.56 B	4674.4	OOM	OOM	**20247**	**(4.33x)**
32	MoE-S	1.88 B	12847	47088	76776	**148673**	**(11.6x)**
	MoE-M	6.60 B	3987.3	17252	21587	**42946**	**(10.8x)**
	MoE-L	14.8 B	2166.9	OOM	OOM	**10217**	**(4.72x)**
64	MoE-S	3.70 B	6702.8	31644	55124	**117150**	**(17.5x)**
	MoE-M	13.0 B	2225.7	OOM	OOM	**12623**	**(5.67x)**
	MoE-L	29.3 B	OOM	OOM	OOM	**1240.8**	**(NaN)**

Table 1: Training throughput (words/s) evaluated on multiple MoE models. Numbers in parentheses show improvement over Zero-Offload^E. ES-MoE enables training large MoE models and a large number of experts increases without OOM, while GPU-only baselines (FairSeq and Tutel) suffer from OOM at scale. Bold indicates the best result for each configuration.



(a) MoE-M



(b) MoE-L

Figure 4: ES-MoE is highly scalable, accommodating up to 5× (67× with SSD) more experts compared to other frameworks.

5.1 Scalability and Performance

Table 1 shows the training throughput of each framework as we increase the number of experts for MoE-based models. For each framework, we use the microbatch size that maximizes its own throughput. Frameworks that rely solely on GPU memory, such as FairSeq, Tutel, and MegaBlocks, fail to complete due to out-of-memory (OOM) as we increase the model size. FairSeq and Tutel struggle to train MoE-L models with 16 or more experts, even with the smallest microbatch of one. Zero-Offload^E encounters OOM when the memory usage exceeds the CPU memory capacity; the 512 GiB RAM is insufficient for training MoE-L models with 64 or more experts. In contrast, ES-MoE efficiently scales to accommodate large models by offloading experts to host CPU memory and storage (SSD), successfully training 29 B-parameter MoE-L with only 4 GPUs.

The result demonstrates that ES-MoE delivers a superior training throughput in all cases, from models of 0.5B to 58B parameters. It outperforms Zero-Offload^E by up to 11.6× and MoE-specialized frameworks by up to 3.16×. The significant performance benefit comes from its ability to handle larger batch sizes, the pipelined offloading design, and the elimination of extra computation from zero-padding, resulting in the highest training throughput across a wide-range of scenarios, as detailed in §5.2.

LLM fine-tuning with 4 GPUs. Table 2 shows the fine-tuned result of a pre-trained Fairseq-MoE-15B model on three different datasets, SST-2 (Socher et al., 2013), MNLI (Williams et al., 2017), and BoolQ (Williams et al., 2017), trained for 100 M tokens without freezing layers. Fine-tuning the model on existing systems requires at least 400 GB of GPU memory and 64 GPUs (Ott et al., 2019). ES-MoE, on the other hand, allows fine-tuning the same model using only 4 GPUs in about 6.5 hours, without compromising model accuracy, unlike low-rank approximation (e.g., LoRA (Hu et al., 2021)) that reduces memory footprint at the expense of accuracy.

Dataset	**SST-2**	**MNLI**	**BoolQ**
Zero-shot accuracy	51.6%	49.3%	60.9%
Fine-tuned accuracy	**88%**	**78.2%**	**68.5%**

Table 2: Fine-tuned results of pre-trained Fairseq-MoE-15B model achieved in only **6.5 hours** with 4 GPUs.

Maximum supported model size. We evaluate the scalability of each framework by comparing the maximum number of experts each can handle with 4 GPUs. Figure 4 shows the result for MoE-M and MoE-L models. ES-MoE demonstrates exceptional scalability, surpassing all baselines by supporting up to 5× more experts and 4.78× larger model with host memory. The result shows that ES-MoE's scalability is not limited to the GPU memory, but can accommodate larger MoE models as much as host memory and storage permits. With 4 TB of SSD, ES-MoE can scale up to 67× more experts (63× larger number of parameters) compared to the baselines. In contrast, FairSeq and Tutel, are constrained by GPU memory and they can train up to only 12

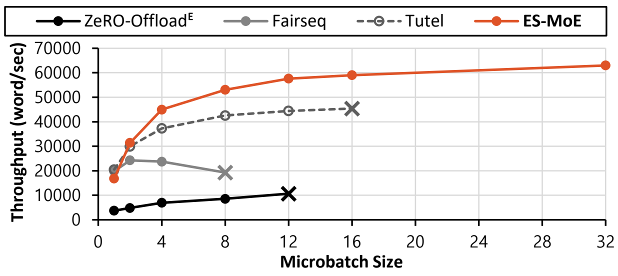


Figure 5: Training throughput of MoE-M with 16 experts while varying the microbatch size. ES-MoE achieves the best training throughput by supporting larger microbatches.

experts (5.7 M params) with MoE-L. They rely on batched matrix multiplication, which requires creating large dispatch masks. The existence of zero-padding even increases the size of the mask, exacerbating the problem.

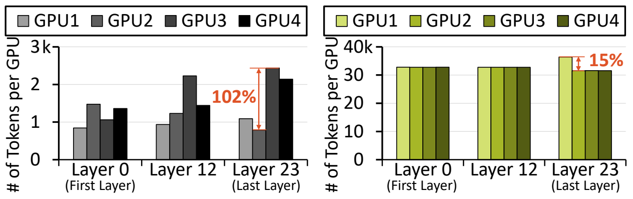
5.2 Component-wise Benefit

All three design components considerably benefit performance. We analyze the benefit of each.

Benefit of expert-wise processing. The expert-wise processing effectively reduces GPU memory usage, allowing ES-MoE to handle larger microbatches and makes the GPU run more efficiently. As shown in Figure 5, ES-MoE accommodates 2.67× larger batches than those manageable by other frameworks, allowing it to perform up to 5.91× faster compared to Zero-Offload^E. Because Zero-Offload^E constantly offloads experts to CPU memory, it introduces significant overhead and suffers from limited training throughput. It also struggles to handle larger microbatches due to inefficiencies associated with zero padding, which can take up to 24%. In contrast, ES-MoE stands out by enabling the training of much larger microbatches, up to 32, achieving superior training throughput. Note that adjusting the size of the microbatch does *not* impact model accuracy.

Next, we quantify the benefit of our pipelined expert processing, which enables concurrent CPU optimization and GPU computation. This leads to shorter iteration times in all cases where the experts are offloaded. Compared to ES-MoE without pipelined expert optimization, ES-MoE achieves up to 63.0% higher throughput on MoE-M with 32 experts, resulting from 61.1% higher GPU utilization.

Benefit of dynamic expert placement. ES-MoE's dynamic expert placement effectively distributes the workload across GPUs, significantly reducing token imbalance. To demonstrate this, we compare the number of tokens assigned to each GPU when training the MoE-M model with 64 experts. As shown in Figure 6(a), FairSeq (and other GPU-based baselines as well) shows a significant discrepancy reaching 102% difference in the number of tokens assigned between the most and least burdened GPUs. In contrast, ES-MoE's dynamic expert placement enables balancing the load across GPUs, reducing the gap down to 15%, as shown in Figure 6(b). Note that the number of tokens differs in two figures because FairSeq requires the use of the smaller microbatch size due to memory constraints, while we use a large microbatch size of 32 for ES-MoE.



(a) FairSeq (E=64, b=2) (b) ES-MoE (E=64, b=32)

Figure 6: The number of tokens assigned to each GPU, evaluated on MoE-M. E and b indicate the number of experts and the microbatch size respectively. Only the first, middle, and last layers are shown out of 24 layers in the model.

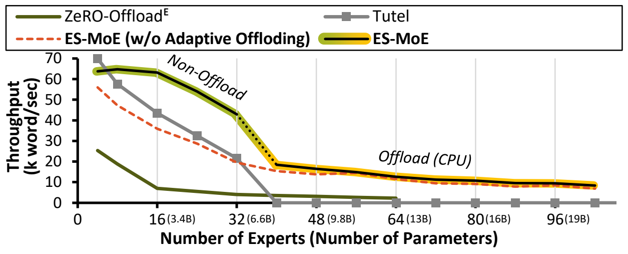


Figure 7: Throughput of ES-MoE measured with MoE-M while varying the number of experts.

Benefit of adaptive offloading. ES-MoE offloads experts only when the aggregate GPU memory does not allow it to load the entire model. It has three modes of operation: 1) non-offload, when the GPUs aggregate capacity allows it to load the entire model; 2) offload to CPU memory; 3) offload to CPU memory and SSD. Figure 7 shows the throughput comparison as the number of experts increases. We compare the performance of ES-MoE, Tutel, and ES-MoE without adaptive offload. The model we use is MoE-M. Up to 32 experts (6.6 B parameters), ES-MoE trains the model without offloading. As ES-MoE is able to use larger microbatch size and eliminate zero padding, its performance is better than Tutel. It is also better than ES-MoE without adaptive offload because it does not unnecessarily offload the experts to the CPU and incurs communication overhead. The other two baselines do not scale beyond 32 experts due to the memory limit. In contrast, ES-MoE scales beyond the aggregate GPU memory capacity. Additionally, the strategy of expert pinning proves to be effective; pinning 25% of experts in an MoE-M model with 32 experts resulted in a 22.8% improvement in throughput, compared to ES-MoE without expert pinning (red dotted line). However,

Scheme	**Thpt.** (Tokens/s)	
ES-MoE	20,247	
– Expert pinning (§4.3)	19,501	(-3.8%)
– Optimizer overlapping (§4.1)	17,943	(-8.7%)
– Larger batch size (§4.1)	5,959	(-301%)
– Zero-padding elimination (§4.2) (=ZeRO-Offload^E)	4,674	(-27.4%)

Table 3: Ablation study with MoE-L with 16 experts. Results are cumulative across rows.

as the number of tokens for an expert decreases, it reduces computational efficiency. When the number of experts is above 104, ES-MoE starts to use the SSD offloading experts.

Ablation study. We report the results of an ablation study on the MoE-L model with 16 experts in Table 3. We evaluate the impact of four techniques: larger batch size, optimizer overlap, zero-padding elimination, and expert pinning. By eliminating each technique sequentially, we evaluate their individual contributions to training throughput. Note that our ES-MoE and ZeRO-Offload^E variant includes upload overlapping. The results show that all design components significantly benefit performance, and increasing the batch size has the most significant effect, achieving 3.01× improvement.

6 Conclusion

This paper addresses the challenges of training large Mixture-of-Experts models, under the constraints of limited GPU memory. By offloading expert parameters and optimizer states to the host, ES-MoE supports scaling MoE models without additional GPUs. Its dynamic expert placement ensures that the load is spread uniformly across GPUs without introducing zero-padding, solving the straggler problem and further saving the memory usage.

Our extensive evaluation demonstrates ES-MoE's superior scalability and throughput. It successfully accommodates up to 67× more experts than conventional methods and achieves remarkable throughput improvements. ES-MoE outperforms existing offloading frameworks by up to 17.5× and shows up to 2.13× gains over Tutel.

Acknowledgements

We thank the anonymous reviewers for providing helpful feedback and suggestions to improve our work. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00398157) and Samsung Electronics.

Impact Statement

This work prioritizes batch-level parallelism over expert-level parallelism and leverages CPU offload to achieve scalable training of large MoE-based models. We believe this work will empower researchers from academia and small organizations with the ability to train MoE-based LLMs with a larger number of experts.

References

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 571–582, 2014.

Clark, A., De Las Casas, D., Guy, A., Mensch, A., Paganini, M., Hoffmann, J., Damoc, B., Hechtman, B., Cai, T., Borgeaud, S., et al. Unified scaling laws for routed language models. In *International Conference on Machine Learning*, pp. 4057–4086. PMLR, 2022.

Dai, W., Zhou, Y., Dong, N., Zhang, H., and Xing, E. P. Toward understanding the impact of staleness in distributed machine learning. *arXiv preprint arXiv:1810.03264*, 2018.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

Gale, T., Narayanan, D., Young, C., and Zaharia, M. Megablocks: Efficient sparse training with mixture-of-experts. *arXiv preprint arXiv:2211.15841*, 2022.

Garey, M. R. Computers and intractability: A guide to the theory of NP-Completeness. *Fundamental*, 1997.

Google Cloud. Understand quota, values, and system limits. https://cloud.google.com/docs/quota/understand-limits, 2024. [Accessed 17-01-2024].

Graham, R. L. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

Griewank, A. and Walther, A. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):19–45, 2000.

Harlap, A., Cui, H., Dai, W., Wei, J., Ganger, G. R., Gibbons, P. B., Gibson, G. A., and Xing, E. P. Addressing the straggler problem for iterative convergent parallel ml. In *Proceedings of the seventh ACM symposium on cloud computing*, pp. 98–111, 2016.

He, J., Zhai, J., Antunes, T., Wang, H., Luo, F., Shi, S., and Li, Q. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 120–134, 2022.

Holmes, A. and Gardizy, A. AI Developers Stymied by Server Shortage at AWS, Microsoft, Google. *The Information*, 2023. URL https://www.theinformation.com/articles/ai-developers-stymied-by-server-shortage-at-aws-microsoft-google.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Huang, C.-C., Jin, G., and Li, J. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1341–1355, 2020.

Huang, H., Ardalani, N., Sun, A., Ke, L., Lee, H.-H. S., Sridhar, A., Bhosale, S., Wu, C.-J., and Lee, B. Towards moe deployment: Mitigating inefficiencies in mixture-of-expert (moe) inference. *arXiv preprint arXiv:2303.06182*, 2023.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Hwang, C., Cui, W., Xiong, Y., Yang, Z., Liu, Z., Hu, H., Wang, Z., Salas, R., Jose, J., Ram, P., et al. Tutel: Adaptive mixture-of-experts at scale. *arXiv preprint arXiv:2206.03382*, 2022.

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Koishekenov, Y., Berard, A., and Nikoulina, V. Memory-efficient nllb-200: Language-specific expert pruning of a massively multilingual machine translation model. *arXiv preprint arXiv:2212.09811*, 2022.

Kuperman, L. How to Solve the GPU Shortage Problem With Automation. https://devops.com/how-to-solve-the-gpu-shortage-problem-with-automation/, 2023. [Accessed 17-01-2024].

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Liu, J., Wang, J. H., and Jiang, Y. Janus: A unified distributed training framework for sparse mixture-of-experts models. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pp. 486–498, 2023.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Microsoft. Deepspeed, 2023. https://github.com/microsoft/DeepSpeed.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

Pudipeddi, B., Mesmakhosroshahi, M., Xi, J., and Bharadwaj, S. Training large neural networks with constant memory using a new execution algorithm. *arXiv preprint arXiv:2002.05645*, 2020.

Puigcerver, J., Riquelme, C., Mustafa, B., and Houlsby, N. From sparse to soft mixtures of experts. *arXiv preprint arXiv:2308.00951*, 2023.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Rajbhandari, S., Ruwase, O., Rasley, J., Smith, S., and He, Y. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2021.

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International Conference on Machine Learning*, pp. 18332–18346. PMLR, 2022.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. Zero-offload: Democratizing billion-scale model training. In *USENIX Annual Technical Conference*, pp. 551–564, 2021.

Rhu, M., Gimelshein, N., Clemons, J., Zulfiqar, A., and Keckler, S. W. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–13. IEEE, 2016.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Susano Pinto, A., Keysers, D., and Houlsby, N. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34: 8583–8595, 2021.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., et al. Mesh-tensorflow: Deep learning for supercomputers. *arXiv preprint arXiv:1811.02084*, 2018.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Vazirani, V. V. *Approximation algorithms*, volume 1. Springer, 2001.

Wang, L., Ye, J., Zhao, Y., Wu, W., Li, A., Song, S. L., Xu, Z., and Kraska, T. Superneurons: Dynamic gpu memory management for training deep neural networks. In *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, pp. 41–53, 2018.

Williams, A., Nangia, N., and Bowman, S. R. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A. M., Le, Q. V., Laudon, J., et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N., and Fedus, W. Designing effective sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

A Evaluation Details

A.1 Models

Model Name	n_{layers}	d_{model}	d_{head}	n_{heads}
MoE-S	12	768	3072	12
MoE-M	24	1024	4096	16
MoE-L	24	1536	6144	16
MoE-XL	24	1536	8192	24

Table 4: Model configurations for GPT-S, M, L, and XL

To create an MoE model based on GPT models, we follow the approach used by Switch (Fedus et al., 2022), which converts the Feed-Forward Network (FFN) in the Transformer architecture into experts and creates multiple copies as the number of experts increases.

To build MoE models of various sizes, we adopt hyperparameter configurations from GPT-3 models. Table 4 shows the details, where n_{layers}, d_{model}, d_{head}, n_{heads} respectively indicate the number of decoder layers, embedding dimension, feedforward layer embedding dimension, and the number of attention heads.

A.2 Additional Evaluations

Metrics	**Fairseq**	**ES-MoE**
Processed Tokens	1B tokens	1B tokens
Training Loss	3.353	3.344
Valid Loss	5.149	5.144
Training Duration (hrs)	**9.47**	**6.90**

Table 5: ES-MoE accelerates the training of the MoE-L model, achieving the same loss 37% more quickly.

# Exp.	**Params.**	**Zero-Offload^{E}**	**ES-MoE**	**# GPUs**
16	13.3B	41%	59%	8
24	19.8B	40%	57%	16
32	23.1B	32%	39%	16
40	32.7B	28%	33%	16
48	39.1B	18%	31%	32

Table 6: GPU utilization while training a MoE-L model with varying numbers of experts.

Case Study: Pretraining. In Table 5, we show a comparative analysis of end-to-end training time and training loss for MoE-L with 8 experts and a batch size of 128. Although both implementations are mathematically equivalent and demonstrate almost identical training and validation losses, a notable difference is observed in training efficiency. When compared to Fairseq (Lepikhin et al., 2020), ES-MoE completes the training process 37% more quickly, highlighting its enhanced efficiency in model training.

Communication and computation overhead. In Table 6, we present an analysis of ES-MoE's offloading overhead by examining the effective GPU utilization percentage. Across various configurations, ES-MoE outperforms Zero-Offload$^{\text{E}}$ in effective GPU utilization, coming from pipelined expert scheduling and dynamic expert placement. The effective GPU utilization decreases with the growth in the number of experts for both Zero-Offload$^{\text{E}}$ and ES-MoE. This trend results from the fixed batch size, causing a reduction in assigned tokens per expert with an increased expert count. Consequently, the overlapping of computation and communication is diminished, and the required CPU computation escalates. Despite this limitation, ES-MoE demonstrates a significant capability to train large MoE models that conventionally require up to 32 GPUs, using even a single GPU. This flexibility in resource utilization may have broader implications, overshadowing the effects of offloading overhead.