Dense Backpropagation Improves Routing for Sparse Mixture-of-Experts

Anonymous Authors[1]

Abstract

Mixture of Experts (MoE) pretraining is more scalable than dense Transformer pretraining, because MoEs learn to route inputs to a sparse set of their feedforward parameters. However, this means that MoEs only receive a sparse backward update, leading to training instability and suboptimal performance. We present a lightweight approximation method that gives the MoE router a dense gradient update while continuing to sparsely activate its parameters. Our method, which we refer to as Default MoE, substitutes missing expert activations with default outputs consisting of an exponential moving average of previous expert outputs. This allows the router to receive signals from every expert for each token, leading to significant improvements in training performance. Our Default MoE outperforms standard TopK routing in a variety of settings without requiring significant computational overhead.

1. Introduction

Large-scale pretraining hinges on scaling up the number of parameters in a model, because models with more parameters are more sample-efficient and require less training to reach the same performance as smaller models (Kaplan et al., 2020; Hoffmann et al., 2022). The most common model architecture is a dense Transformer architecture (Vaswani et al., 2023) because its performance scales well with parameters and data. However, a sparsely activated Mixture-of-Experts (MoE) Transformer architecture (Shazeer et al., 2017) has been used by many industry deployments (DeepSeek-AI et al., 2024b; xAI, 2024; Databricks, 2024; Jiang et al., 2024; Snowflake, 2024; DeepSeek-AI et al., 2024a) because MoEs have been shown to scale even better than dense Transformers (Clark et al., 2022; Du et al., 2022; Lepikhin et al., 2020; Fedus et al., 2022). MoEs learn a *routing* function that selectively activates the Top-K subset of their modules, or *experts*, most relevant to a given input. This conditionally sparse activation (Jacobs et al., 1991; Jordan & Jacobs, 1994) allows us to multiplicatively increase the model parameter count without significantly increasing the cost of training or inference.

At train time, the sparse router enables very large MoEs to be trained with relatively little computation per token, but it also presents a challenge. The router does not receive a gradient update from experts that it does not activate. This may slow down learning, as the gradient update is unable to adjust the router to promote the optimal expert for each token. It may also cause load imbalance – where a few experts are over-utilized – leading to inefficient training and resource usage (Zoph et al., 2022; Zhou et al., 2022). This can be corrected by activating all experts during training, enabling all experts and routing directions to be optimized at once, although this requires extreme computational costs.

In this work we propose a new method that strikes a middle ground between dense and sparse routing; our proposed router can receive and differentiate contributions from all experts, while the computational cost remains virtually identical to a standard Top-K router. In our proposed method, the router receives contributions from all experts for every token. However, a forward pass is only computed on the Top-K experts for each token, while other experts contribute a "default vector" that represents the expected output of a typical token. Our method adds minimal computational overhead compared to a standard Top-K router while significantly improving model training speed and performance.

2. Background & Related Work

MoEs. The MoE layer replaces the feedforward networks (FFN) of Transformers and consists of two components : **1)** N FFNs (*experts*), $E_0(x), E_1(x), \dots E_N(x)$ and **2)** a router that assigns tokens to experts. Each input to the MoE layer is processed by K experts where $K < N$, and this is the source of sparsity in MoEs. The K experts are chosen by the router, which is a learnable component that maps each token to a set of weights over the experts. The router performs a linear transformation $\mathbb{R}^{d_{\text{token}}} \to \mathbb{R}^N$ which produces logits; these are normalized using softmax, resulting in a probability distribution over the experts. With the router's

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

linear transformation parameterized by a matrix W, we can represent the expert weights π in the following way:

$$\pi \in \mathbb{R}^N = \text{Softmax}(Wx) \tag{1}$$

Once we have these expert weights, we apply a routing function to decide which of K experts to route and process this token through. We consider Top-K routing because it is the most popular.

Top-K routing. A standard method to select K out of N experts given the expert weights is to select the experts corresponding to the K highest weights. Top-K routing (Fedus et al., 2022) passes the token to the K selected experts and averages the expert outputs using these weights to produce the final output. Experts not selected by the Top-K routing function do not process the token, and this introduces sparsity in MoEs. By representing the K chosen experts as the set $\mathcal{A}$, we can express the output of the MoE layer as an average of expert outputs weighted by the router scores:

$$y = \Sigma_{i \in \mathcal{A}} \pi_i E_i(x). \tag{2}$$

The expert weights serve two roles. They are used by the routing function to decide which of the K experts to process a token through, and also provide the weights for combining the expert outputs. Top-K routing makes the MoE layer desirable for training large, compute-efficient neural networks. It allows models to be scaled up, by way of increasing the total number of experts, while keeping the compute per token constant (as it is a function of K and not N).

The Router Gradient. Consider the gradient of the MoE layer's output y with respect to the router parameters W. We express y as a function of W by combining Equation (1) and Equation (2). With the chain rule, we can backpropagate through this function by considering the gradient at each respective step:

$$\frac{\partial y}{\partial W} = \frac{\partial y}{\partial \pi} \frac{\partial \pi}{\partial W} \tag{3}$$

The second term in Equation (3), $\frac{\partial \pi}{\partial W}$, is straightforward to compute because the steps in Equation (1) are easily differentiable, as they consist of linear operations and activations. But the first term, $\frac{\partial y}{\partial \pi}$, is not differentiable because in Equation (2) Top-K expert selection transforms the continuous router weights $\pi \in \mathbb{R}^N$ into a discrete set of selected experts $\mathcal{A}$ with $\binom{N}{K}$ possible values. One way to address backpropagation of nondifferentiable operations is to use the straight-through estimator (Bengio et al., 2013), which treats the operation as the identity function. With straight-through we bypass the Top-K routing function and Equation (2) becomes the dot product between π and the vector of all $E_i(x)$ with the following gradient:

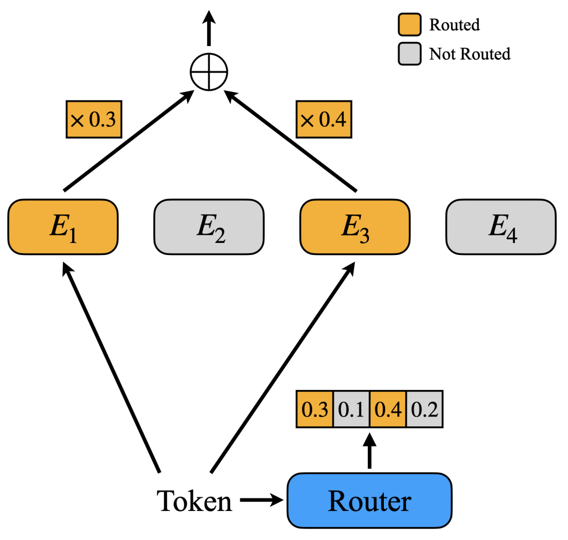
$$\frac{\partial y}{\partial \pi} = \begin{bmatrix} E_1(x), & E_2(x) & \cdots & E_N(x) \end{bmatrix}^T \tag{4}$$

This *dense* gradient requires the output of *all* of the experts for a given token. Passing a token through all the experts will destroy the sparsity of the MoE layer, thus impeding the scalability of this architecture. In this work, we develop a method for applying the straight-through estimator while maintaining the sparsity of the MoE layer by substituting the non-activated expert outputs in the dense gradient term with a *default* vector. This default vector is a running average of previously computed expert values, and it approximates the missing expert output without incurring the cost of a forward pass. Notably, this enables non-activated experts to contribute to the router's gradient update.

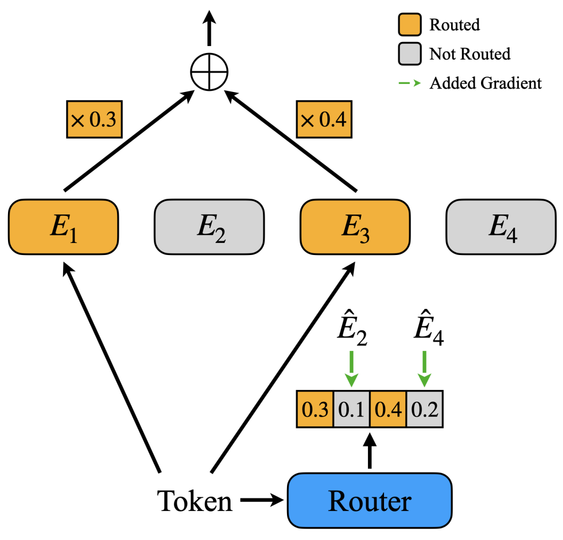
Related Works. Previous work has tried to address the issue of routing in MoEs. Separate from Top-K is the Sinkhorn routing method (Clark et al., 2022). Fedus et al. (2022) propose an auxiliary loss that encourages load balancing. Dai et al. (2024) propose multiple additional auxiliary loss terms. Wang et al. (2024) propose learning biases rather than an auxiliary load balancing loss, and this is used in DeepSeek-AI et al. (2024b). Phi-3.5-MoE (Abdin et al., 2024) uses SparseMixer (Liu et al., 2024; 2023), another estimator for $\partial y / \partial \pi$ not involving straight-through. We provide a comparison between our approach and SparseMixer later in Section 4.3. Our approach is to still use straight-through and auxiliary loss, but *approximate* these additional expert outputs. We now present our method.

3. Designing Dense Backpropagation

In this section we design a new MoE router that can receive a dense gradient without requiring additional forward passes through experts. Equation (4) describes the dense gradient of the router for an individual token. Each expert output $E_i(x)$ in the router gradient corresponds to updating the embedding for expert i in the routing layer (i.e. the ith row of the router parameters W). In a standard sparse MoE, the gradient only includes expert outputs E_i for the experts $i \in \mathcal{A}$ selected by the Top-K routing function. The other terms are essentially 0, which means that non-activated experts can not influence the router's update for a token. Our goal is to *approximate* the dense gradient in Equation (4) without directly computing the missing terms $E_{i'}(x)$ for $i' \notin \mathcal{A}$. This enables the router to receive a signal from all experts in its gradient update, without actually activating all of these experts. The router can then consider information from all experts when learning to route tokens instead of being limited to only receiving feedback from experts that were activated. As a result, we expect an improved router which learns to allocate tokens to different experts more effectively, leading to improved MoE training performance.



(a) Original Router



(b) Dense Approximation Router (ours)

Figure 1: **Overview of Routing with Dense Approximations**. The original mixture of experts router only receives gradients corresponding to experts the token is routed to, because there is no output from other experts. Our approach provides the router with a complete (dense) gradient by letting non-activated experts contribute a *default* vector that approximates its output without the cost of a forward pass. As indicated by the dashed green arrows, the approximated gradients are not actually connected to the token in the computation graph; instead, they are artificially applied in the backward pass.

Figure 1 presents an overview of our method for updating the router. Since we wish to avoid additional computational overhead, we fill in the missing terms in Equation (4) with estimates $\hat{E}_i$ for non-activated experts E_i. This simple addition ensures that the router updates its weights for all experts instead of only those selected by the Top-K routing.

3.1. Approximating Missing Expert Outputs

Backpropagating through the sparse MoE activation in Equation (2) does not align with the router's true dense gradient specified in Equation (4). Specifically, the gradient in TopK routing leads to an error based on the non-activated experts. Let us consider the gradient of the router parameters W with respect to the loss $\mathcal{L}$. For clarity, we examine the gradient term for a single input x; our approach also translates to practically training the router with a batch of inputs.

Using the chain rule, we decompose the router gradient to isolate the nondifferentiable $\frac{\partial y}{\partial \pi}$, similar to Equation (3):

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial \pi}\frac{\partial \pi}{\partial W} \tag{5}$$

In standard Top-K routing, the gradient term effectively uses zero for non-activated experts:

$$\frac{\partial y}{\partial \pi_i} = \begin{cases} E_i(x) & \text{if } i \in \mathcal{A} \\ 0 & \text{if } i \notin \mathcal{A} \end{cases} \tag{6}$$

The error is introduced by the missing terms in $\frac{\partial y}{\partial \pi}$:

$$\epsilon_{\text{TopK}} = \frac{\partial \mathcal{L}}{\partial y} \sum_{i' \notin \mathcal{A}} E_{i'}(x) \frac{\partial \pi}{\partial W} \tag{7}$$

To address this error, we need a meaningful estimate for $E_{i'}(x)$ without activating the non-activated experts $E_{i'}$. Our approach to this involves maintaining a *default vector* for each expert $\hat{E}_i = \mathbb{E}_x[E_i(x)]$ that represents the expected value of the expert output. In practice, we will compute this estimate by taking the sample average of existing expert outputs $E_i(x')$ from other tokens x' for which E_i was actually activated. Considering this estimator, our gradient term for non-activated experts is now nonzero:

$$\frac{\partial y}{\partial \pi_i} = \begin{cases} E_i(x) & \text{if } i \in \mathcal{A} \\ \hat{E}_i & \text{if } i \notin \mathcal{A} \end{cases} \tag{8}$$

Note that $\hat{E}_i$ is not a function of x; it is universally applied to complete the router gradient for all such tokens in the batch. The gradient error term with this new default vector is as follows:

$$\epsilon_{\text{default}} = \frac{\partial \mathcal{L}}{\partial y} \sum_{i \notin \mathcal{A}} (E_i(x) - \mathbb{E}_x[E_i(x)]) \frac{\partial \pi}{\partial W} \tag{9}$$

The error term corresponding to $\frac{\partial y}{\partial \pi}$, $\sum_{i \notin \mathcal{A}}(E_i(x) - \mathbb{E}_x[E_i(x)])$, is 0 in expectation for all i. As a result, our

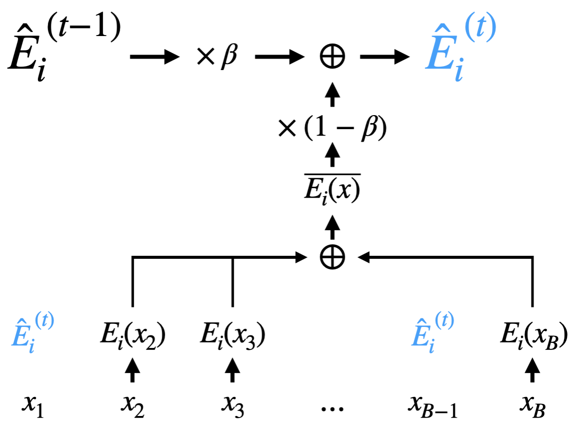


Figure 2: **Demonstration of Exponential Moving Average to Update Expert Approximations.** As described in Equation (11), our MoE training method involves substituting missing expert outputs with a default vector based on the average output. For tokens $x_1, \ldots, x_B$ in a batch of size B, we first compute $E_i(x_j)$ for all x_j for which expert i was activated. This is part of the standard MoE forward pass. Our method additionally averages these existing expert outputs to yield the sample mean $\overline{E_i(x)}$. We use this sample mean to update our exponential moving average based on the prior value $\hat{E}_i^{(t-1)}$ and the decay rate β. After computing the new EMA value $\hat{E}_i^{(t)}$ (highlighted in blue), we substitute it back in as the expert output for tokens that were not routed to expert i. At the end of this process, each token has some expert output E_i either computed directly or assigned to it.

default vector method corrects the gradient error by filling in missing expert activations with the mean expert output.

3.2. Default Expert Outputs with Exponential Moving Averages

To enable dense backpropagation without sacrificing the computational efficiency of sparse MoE forward passes, we propose **Default MoE**. The Default MoE provides a default value for the outputs of non-selected experts using exponential moving averages (EMAs). The EMA accounts for the fact that experts are being updated, and thus their outputs for inputs x will change over the course of training. For each expert E_i, we maintain an EMA of its average output:

$$\hat{E}_i^{(t)} = \beta \hat{E}_i^{(t-1)} + (1-\beta)\overline{E_i(x)} \tag{10}$$

where $\beta \in [0,1]$ is the decay rate and $\overline{E_i(x)}$ is the sample average of expert outputs E_i for all tokens x for which expert i was activated. We compute $\overline{E_i(x)}$ during the forward pass by collecting all such expert outputs $E_i(x)$ and taking the average. This operation is computationally trivial, since the outputs themselves are already computed as part of the standard MoE forward pass. Thus, the EMA serves as a lightweight proxy for the expert's expected output.

During the forward pass, for a given input x, and an expert i (where there are N total experts) we compute the output of the MoE as:

$$y = \sum_{i=1}^{N} \pi_i \cdot \begin{cases} E_i(x) & \text{if } i \in \text{TopK}(\pi) \\ \hat{E}_i^{(t)} & \text{otherwise} \end{cases} \tag{11}$$

Specifically, we first compute the forward pass for activated experts $E_i(x)$ for each token. Then, we perform the EMA update step using these outputs. After applying the EMA update, we use $\hat{E}_i^{(t)}$ to approximate missing outputs for non-activated experts. Figure 2 demonstrates an example of this EMA update step for an arbitrary expert E_i. Our method ensures that computed expert outputs at the current step are factored into the EMA update before applying the EMA as a substitute for other tokens that did not activate the expert.

This formulation allows the router to receive meaningful gradients for all experts while maintaining the computational benefits of sparse activation. The EMA provides a reasonable approximation of what non-activated experts would have computed, based on their historical outputs for other tokens. Importantly, this approximation requires only $\mathcal{O}(1)$ additional memory per expert and requires minimal additional forward pass computation for the EMA update.

This approach enables dense backpropagation through the router while preserving the sparse computational pattern that makes MoE architectures efficient. The router receives gradient information about all possible routing decisions, not just the selected experts, potentially leading to more informed routing decisions.

4. Evaluation

We describe the experimental setup in Section 4.1, show that our Default MoE method speeds up training by 15% compared to the TopK baseline in Section 4.2, and then show how our method compares to the baseline as we ablate virtually every parameter in the experimental setup in Section 4.3. We finish by concluding that our method does not significantly impact throughput or the memory footprint of training, so the improvements we observe are -as far as we can tell- a free lunch.

4.1. Experimental Setup

We train a standard MoE with 8 experts. The model has a hidden dimension of 1024, with 2B total parameters. When

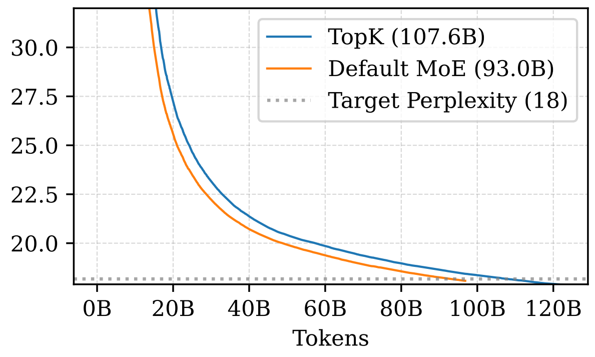


Figure 3: **Default MoE Beats TopK.** Our method reaches the target perplexity of 18 about 15% faster than the baseline TopK method, without introducing any additional overhead.

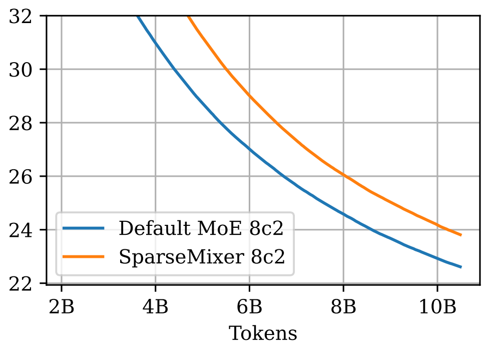


Figure 4: **Comparison of Default MoE and SparseMixer.** We compare Default MoE with SparseMixer, both configured with 8 experts and Top-K=2 active experts. The results report training perplexity throughout training, demonstrating that Default MoE consistently outperforms SparseMixer.

doing $K = 1$ top-K routing, only 1 expert at each layer is activated, and about $500M$ of the parameters are active. We ablate the model architecture, hidden dim, number of total experts, and number of active experts in Section 4.3. We train on FineWeb (Penedo et al., 2024) with the Llama3 tokenizer (Dubey et al., 2024). We use SwiGLU (Shazeer, 2020) MLPs following Llama (Touvron et al., 2023), 16 attention heads with dimension of 64, LayerNorm (Ba et al., 2016) and RoPE (Su et al., 2023).

Hyperparameters. We use the initialization from Wang et al. (2022) for residual branch merge layers and the initialization from Nguyen & Salazar (2019) for all other layers. We use the AdamW optimizer (Loshchilov & Hutter, 2019). We set the maximum learning rate to 7×10^{-4} following a hyperparameter sweep, the minimum learning rate to 7×10^{-5}, and use a standard cosine decay schedule. We use a sequence length of 2048 and a global batch size of 1024, resulting in a global token batch size of 2^{21}.

MoE-Specific Training Details. We set the auxiliary loss (Fedus et al., 2022) to 0.01. We do not use z-loss or jitter, because we find that at this scale they do not improve training for the baseline model. Following DeepSeek-AI et al. (2024b), we set the first layer to be dense. Following Liu et al. (2024), we compute the aux loss across nodes. We train dropless MoEs.

Implementation. We train with the gpt-neox library (Andonian et al., 2023) integrated with Megablocks (Gale et al., 2022) and augmented with a number of Triton kernels that speed up training. We have open-sourced our implementation already for reproducibility, and will include a link here upon publication.

4.2. Main Results

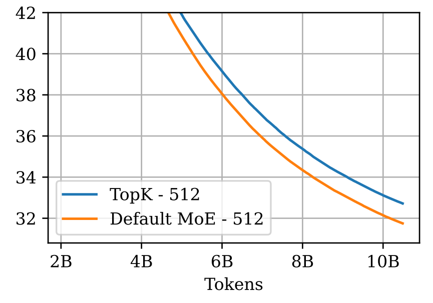
Our main result compares our Default MoE, which performs a dense update of the router weights by approximating the dense gradient, to baseline Top-K routing, both with $K = 1$ active experts. In Figure 3 we compare how fast each method trains our MoE to a target perplexity of 18. Without introducing significant overhead, our method reduces the tokens required to reach a target perplexity of 18 by 15%. As the model has just $500M$ active parameters, we are training for significantly more than the compute-optimal number of tokens Hoffmann et al. (2022). Therefore, we can be confident that we are not just seeing our method converge faster to a worse minima; this is a real improvement in terms of both speed of convergence and the quality of the converged model.

Default MoE Beats Sparsemixer. Liu et al. (2023; 2024) use Sparsemixer, which estimates the true router gradient without straight-through. Liu et al. (2024) note that Sparsemixer lags behind Top-K (which our method always outperforms) for the first $0.5T$ tokens, likely due to the noise that Sparsemixer adds. It is somewhat outside of our computational budget to run a baseline for $0.5T$ tokens. We see in Figure 4 that our method significantly outperforms SparseMixer for at least the first $10B$ tokens.

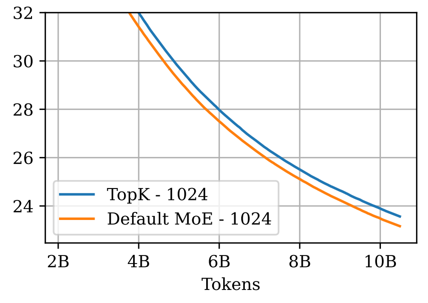
4.3. Ablations

We now ablate every setting in our experimental setup. , from the model architecture to the learning rate.

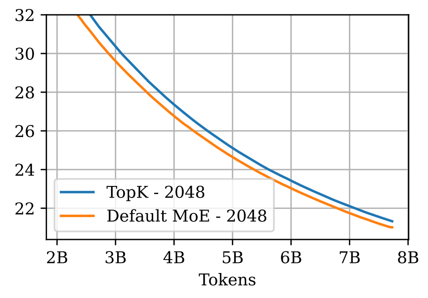
Default MoE Remains Superior As We Increase the Model Size. Figure 5 compares the two methods as we increase the size of the model from $557M$ total parameters at a hidden dimension of 512, to $7.33B$ total parameters when the hidden dimension is 2048. Our method outperforms the baseline across all model sizes. All other results in the paper use a hidden dimension of 1024 because this model is fairly easy to train with just data parallelism.



(a) Hidden Dimension = 512



(b) Hidden Dimension = 1024



(c) Hidden Dimension = 2048

Figure 5: **Ablation over the size of the hidden dimension.** We perform an ablation study across varying hidden dimensions for both the Default MoE and a Top-K baseline approach. We plot the perplexity across 3 hidden dimensions: 512, 1024 and 2048. Notably, the Default MoE consistently outperforms the Top-K baseline across all model sizes, demonstrating lower perplexity at every scale. We compare the training efficiency of Default MoE and Top-K by measuring the number of tokens required to reach a target perplexity. Since Default MoE uses an Exponential Moving Average (EMA) of previous expert outputs to supplement the router during training, it requires fewer tokens to reach the same perplexity as Top-K.

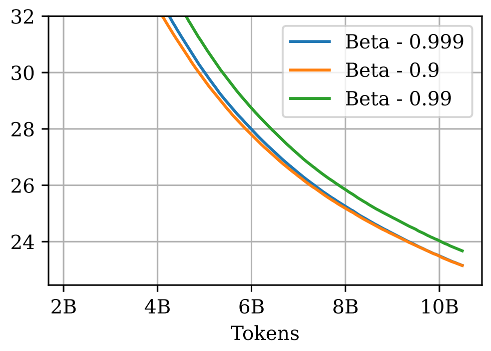


Figure 6: **Selecting the Best EMA Beta Parameter.** We compare different values of the exponential moving average (EMA) smoothing factor β for Default MoE, evaluating $\beta = 0.999$, $\beta = 0.99$, and $\beta = 0.9$. The results show that $\beta = 0.9$ provides the most stable and effective training behavior, leading to lower perplexity. Based on these findings, we use $\beta = 0.9$ for the remainder of our experiments.

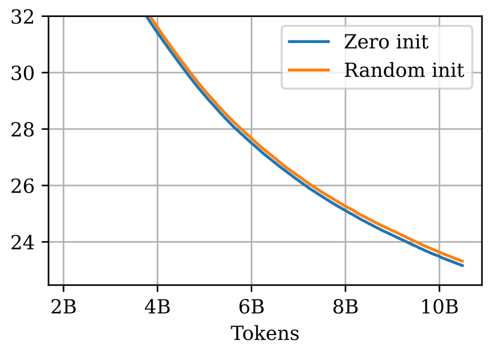


Figure 7: **EMA Buffer Initialization Strategies for Default MoE.** We compare two approaches for initializing the exponential moving average (EMA) buffer for Default MoE: zero initialization and random initialization from a Gaussian distribution. The results indicate that zero initialization is more effective, leading to lower perplexity. Based on this finding, we adopt zero initialization for our experiments.

Varying the Default MoE β Hyperparameter. The lone hyperparameter introduced by our method is the β parameter of the EMA. We vary the β used in the Default MoE in Figure 6 and find that both $\beta = 0.9$ and $\beta = 0.999$ perform equally well. Intuitively, β parametrizes how fast our EMA adjusts its model of the sample mean as we train the model. In principle the optimal value of β might depend on the learning rate, which defines how fast the model is changing, the batch size, which controls how many tokens are actually incorporated into the EMA, and even the data distribution. However, we don't take these factors into consideration and simply set $\beta = 0.9$ in all other results.

Initializing the EMA. We experiment with initializing default vector EMA in two ways: zero initialization and random (Gaussian) initialization. Zero initialization leads to the default vector starting off with zero signal in the earliest training steps. Random initialization, on the other hand, immediately provides some signal to fill in missing expert outputs but this signal starts as pure noise. Given these tradeoffs, we try both approaches and outline our results in Figure 7. Initializing our EMA with zeroes demonstrates slight improvement compared to random initialization. We believe this is due to the adverse effects from providing a random signal to the router early in training. However, the impact is minor, even without the use of any explicit bias correction term in our EMA.

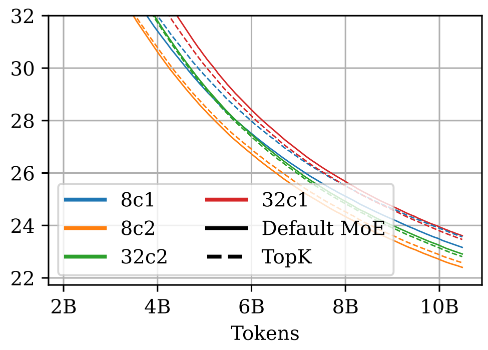


Figure 8: **Comparison of Default MoE and Top-K Across Different Configurations.** We compare Default MoE and Top-K across four configurations: 8c1, 8c2, 32c1, and 32c2. The results show that Default MoE outperforms Top-K in the standard MoE configurations (8c1 and 8c2), but underperforms in the finegrained MoE setups (32c1 and 32c2). This can be attributed to the increased number of EMA buffers in larger models, which require more training time to stabilize. Since each EMA must accurately approximate the sample mean, the presence of more EMA buffers in 32c1 and 32c2 delays convergence, particularly when fewer tokens are available per expert.

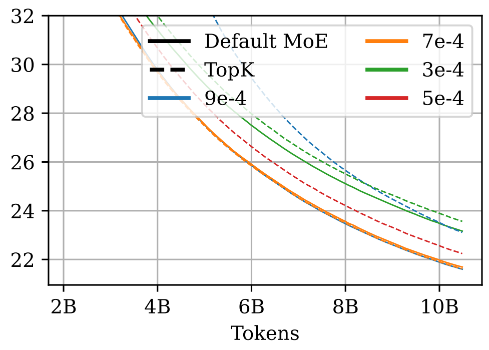


Figure 9: **Learning Rate Sweep for Top-K and Default MoE.** We perform a learning rate sweep for the Top-K approach, evaluating lr $= 9e$-4, $7e$-4, $5e$-4, and $3e$-4 to determine the largest stable learning rate, which is then used for Default MoE. The results show that while Top-K struggles with higher learning rates—evidenced by poor performance at $9e$-4—Default MoE remains stable and can effectively utilize larger learning rates. This is demonstrated by Default MoE achieving comparable performance at $9e$-4 to Top-K at $7e$-4, highlighting its greater training stability.

MoE Architecture Ablations. We apply Default MoE with different expert configurations corresponding to various levels of sparsity, using `NcK` to refer to a model with N total experts and K active experts. When increasing the number of experts past 8, we fine-grain experts (Dai et al., 2024) so that the total number of model parameters remains constant. Specifically, the intermediate size is now 704 rather than 2816 because we have $4\times$ the number of experts). In Figure 8, we train both a standard Top-K model and Default MoE with the configurations 8c1, 8c2, 32c1, and 32c2, which correspond to sparsity factors of $1/8$, $1/4$, $1/32$, and $1/16$ respectively.

Interestingly, while we outperform the non-finegrained MoEs, our Default MoE is slower to train for finegrained MoEs. This is because each expert's default vector is updated with fewer tokens, as each expert is only receiving a quarter of the tokens compared to a vanilla MoE if we assume stable routing. Our estimation error of the true expert activation mean is therefore higher, which in turn leads to not approximating the dense router gradient well. We plan to train these finegrained MoEs for more tokens and see whether the trend remains the same.

Tuning the Learning Rate. We want to ensure that we are comparing against a tuned baseline, so we tune the learning rates in Figure 9. We use the learning rate that achieves the best performance for the TopK baseline, $7e-4$, for our main results in Figure 3. However, this is actually not the best learning rate for the Default MoE; the larger learning rate of $9e-4$ is slightly better. By contrast, $9e-4$ is much too large for the baseline. This indicates that our Default MoE is more stable to train, likely because we are updating the entire router. If we train the baseline with such a large learning rate, we see that a single iteration with a very imbalanced load will lead to a large loss spike; this iteration is very noticeable because, since we train dropless MoEs, it is also slower. We never observe this for the Default MoE. It is unsurprising that our Default MoE's best learning rates may be different from the baseline, but importantly, our method outperforms the baseline at *all* learning rates we consider.

Passing the Default Vector Forward. Our goal is to provide a default expert activation for unactivated experts so that the router can receive a gradient for all experts. We do this by passing the default vector forward through the network, and the router gradient is automatically computed. However, we can in principle do this without passing the default vector forward, and just manually writing the gradient update for the backward pass. One reason why we pass the default vector forward is because our error in estimating the true dense gradient, as written in Eq. 9, is scaled by the loss. Therefore if the default vector activations actually improve the model's output, our error is smaller. We validate this in Figure 10, where we find that passing the default vector forward does improve over only updating the router gradients in the backward pass with the default vector.

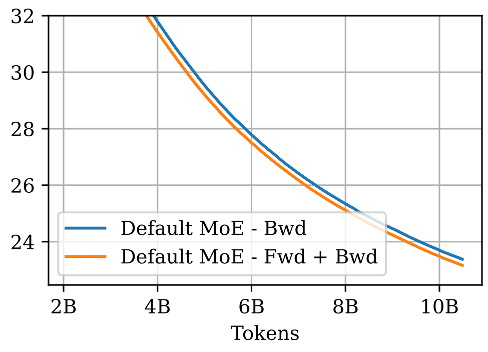


Figure 10: **The importance of applying the EMA during the forward pass.** We perform an ablation study to examine the effect of applying the exponential moving average (EMA) during both the forward and backward passes versus only the backward pass. The results show that using the EMA in both passes leads to consistently lower training perplexity, whereas applying the EMA only in the backward pass—which affects only the gradient update—results in inferior performance. This highlights the importance of incorporating EMA throughout training to maximize its stabilizing effect.

4.4. Efficiency

Table 1: **Throughput Comparison.** We compare the throughput between our method and the TopK baseline for different model sizes. Throughput is measured in tokens per second (sequence length 2048) on a single GPU.

Hidden Dim	Model Size	Tokens per second TopK	Tokens per second Ours	Overhead vs TopK
512	557M	55,968	54,157	-3.34%
1024	1.96B	26,393	25,913	-1.85%
2048	7.33B	1,393	1,391	-0.18%

To ensure a fair comparison, it is crucial that our method does not significantly reduce throughput or increase the memory footprint of training.

Our Method Does Not Significantly Reduce Throughput. In Table 1 we report the throughput in samples per second and of TopK and our method while training the standard 8-expert MoE on a single GPU node. Our method is quite lightweight, because we just need to update an EMA in the forward pass and then update the router gradients in the backward pass. However, for the 2B model we train, this is still nearly a 2% overhead; less than the 15% speedup we observe over the TopK baseline in Figure 3, but still non-zero. We find that the overhead of our method decreases as we increase the hidden size. The $2B$ model we train has 1024 hidden size, 2048 hidden size is an $8B$ model, etc. As the hidden size increases, the proportion of time spent in the MLP matmuls increases, and the overhead of our model is no longer significant compared to the total MoE layer runtime. For even just an 8B model, our overhead is near-zero; and of course, this is almost $100\times$ smaller than production MoEs DeepSeek-AI et al. (2024b).

Our Method Does Not Significantly Increase the Memory Footprint. The EMA buffers are not updated during the backward pass, so we don't need to store any additional activations. The only additional memory required by our method is the EMA buffers themselves. For each expert in each layer, we store a buffer of the same size as the model's hidden dimension. The total number of parameters in an expert is $hidden\_size \times intermediate\_size$; for our MoE this is 1024×2816. And we are just increasing this by 1024, which is a $1/2816 = 0.03\%$ increase in the number of parameters. The MoE parameters are not the entirety of the model (they are about $3/4$) so the additional memory footprint of our method is negligible.

5. Discussion

We propose a training method to improve language modeling performance for MoEs. By approximating the signal of a dense mixture-of-experts layer, the MoE router is able to receive information from all experts for each token, regardless of how many were activated. This approximated dense signal unlocks the possibility for training more sparse MoEs. With this method, our Default MoE outperforms the standard Top-K baseline in a variety of settings. Moreover, the Default MoE requires minimal added computational overhead which decreases as the model size grows larger, showcasing the potential of our method for large-scale MoE pretraining.

Limitations. All of the evaluations in our paper are based on perplexity rather than zero-shot language model benchmarks, because the models we train are not really large enough, or trained for long enough, to have nontrivial performance on benchmarks such as MMLU. In a future version of the paper we plan to train larger models for longer so that we can gauge how our Default MoE's improvement in perplexity translates to better performance on language model benchmarks.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abdin, M., Aneja, J., Awadalla, H., Awadallah, A., Awan, A. A., Bach, N., Bahree, A., Bakhtiari, A., Bao, J., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, M., Cai, Q., Chaudhary, V., Chen, D., Chen, D., Chen, W., Chen, Y.-C., Chen, Y.-L., Cheng, H., Chopra, P., Dai, X., Dixon, M., Eldan, R., Fragoso, V., Gao, J., Gao, M., Gao, M., Garg, A., Giorno, A. D., Goswami, A., Gunasekar, S., Haider, E., Hao, J., Hewett, R. J., Hu, W., Huynh, J., Iter, D., Jacobs, S. A., Javaheripi, M., Jin, X., Karampatziakis, N., Kauffmann, P., Khademi, M., Kim, D., Kim, Y. J., Kurilenko, L., Lee, J. R., Lee, Y. T., Li, Y., Li, Y., Liang, C., Liden, L., Lin, X., Lin, Z., Liu, C., Liu, L., Liu, M., Liu, W., Liu, X., Luo, C., Madan, P., Mahmoudzadeh, A., Majercak, D., Mazzola, M., Mendes, C. C. T., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Ren, L., de Rosa, G., Rosset, C., Roy, S., Ruwase, O., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Shen, Y., Shukla, S., Song, X., Tanaka, M., Tupini, A., Vaddamanu, P., Wang, C., Wang, G., Wang, L., Wang, S., Wang, X., Wang, Y., Ward, R., Wen, W., Witte, P., Wu, H., Wu, X., Wyatt, M., Xiao, B., Xu, C., Xu, J., Xu, W., Xue, J., Yadav, S., Yang, F., Yang, J., Yang, Y., Yang, Z., Yu, D., Yuan, L., Zhang, C., Zhang, C., Zhang, J., Zhang, L. L., Zhang, Y., Zhang, Y., Zhang, Y., and Zhou, X. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL https://arxiv.org/abs/2404.14219.

Andonian, A., Anthony, Q., Biderman, S., Black, S., Gali, P., Gao, L., Hallahan, E., Levy-Kramer, J., Leahy, C., Nestler, L., Parker, K., Pieler, M., Phang, J., Purohit, S., Schoelkopf, H., Stander, D., Songz, T., Tigges, C., Thérien, B., Wang, P., and Weinbach, S. GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch, 9 2023. URL https://www.github.com/eleutherai/gpt-neox.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Clark, A., de las Casas, D., Guy, A., Mensch, A., Paganini, M., Hoffmann, J., Damoc, B., Hechtman, B., Cai, T., Borgeaud, S., van den Driessche, G., Rutherford, E., Hennigan, T., Johnson, M., Millican, K., Cassirer, A., Jones, C., Buchatskaya, E., Budden, D., Sifre, L., Osindero, S., Vinyals, O., Rae, J., Elsen, E., Kavukcuoglu, K., and Simonyan, K. Unified scaling laws for routed language models, 2022. URL https://arxiv.org/abs/2202.01169.

Dai, D., Deng, C., Zhao, C., Xu, R. X., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., Xie, Z., Li, Y. K., Huang, P., Luo, F., Ruan, C., Sui, Z., and Liang, W. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2401.06066.

Databricks. Dbrx, 2024. URL https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm.

DeepSeek-AI, Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Dengr, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Xu, H., Yang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Chen, J., Yuan, J., Qiu, J., Song, J., Dong, K., Gao, K., Guan, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Zhu, Q., Chen, Q., Du, Q., Chen, R. J., Jin, R. L., Ge, R., Pan, R., Xu, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Zheng, S., Wang, T., Pei, T., Yuan, T., Sun, T., Xiao, W. L., Zeng, W., An, W., Liu, W., Liang, W., Gao, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Chen, X., Nie, X., Sun, X., Wang, X., Liu, X., Xie, X., Yu, X., Song, X., Zhou, X., Yang, X., Lu, X., Su, X., Wu, Y., Li, Y. K., Wei, Y. X., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Zheng, Y., Zhang, Y., Xiong, Y., Zhao, Y., He, Y., Tang, Y., Piao, Y., Dong, Y., Tan, Y., Liu, Y., Wang, Y., Guo, Y., Zhu, Y., Wang, Y., Zou, Y., Zha, Y., Ma, Y., Yan, Y., You, Y., Liu, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Huang, Z., Zhang, Z., Xie, Z., Hao, Z., Shao, Z., Wen, Z., Xu, Z., Zhang, Z., Li, Z., Wang, Z., Gu, Z., Li, Z., and Xie, Z. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024a. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Wang, J., Chen, J., Chen, J., Yuan, J., Qiu, J., Li, J., Song, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Wang, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Wang, Q., Zhu, Q., Chen, Q., Du, Q., Chen,

R. J., Jin, R. L., Ge, R., Zhang, R., Pan, R., Wang, R., Xu, R., Zhang, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Pan, S., Wang, T., Yun, T., Pei, T., Sun, T., Xiao, W. L., Zeng, W., Zhao, W., An, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Zhang, X., Chen, X., Nie, X., Sun, X., Wang, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Song, X., Shan, X., Zhou, X., Yang, X., Li, X., Su, X., Lin, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhu, Y. X., Zhang, Y., Xu, Y., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Yu, Y., Zheng, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Tang, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Wu, Y., Ou, Y., Zhu, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Zha, Y., Xiong, Y., Ma, Y., Yan, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Huang, Z., Zhang, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Xu, Z., Wu, Z., Zhang, Z., Li, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Gao, Z., and Pan, Z. Deepseek-v3 technical report, 2024b. URL https://arxiv.org/abs/2412.19437.

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.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Tan, X. E., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Grattafiori, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Vaughan, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Franco, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Wyatt, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Ozgenel, F., Caggioni, F., Guzmán, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Thattai, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Damlaj, I., Molybog, I., Tufanov, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Prasad, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Huang, K., Chawla, K., Lakhotia, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Tsimpoukelli, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Laptev, N. P., Dong, N., Zhang, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Li, R., Hogan, R., Battey, R., Wang, R., Maheswari, R., Howes, R., Rinott, R., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Kohler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Albiero, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wang, X., Wu, X., Wang, X., Xia, X., Wu, X., Gao, X., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Hao, Y., Qian, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., and Zhao, Z. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022. URL https://arxiv.org/abs/2101.03961.

Gale, T., Narayanan, D., Young, C., and Zaharia, M. Megablocks: Efficient sparse training with mixture-of-experts, 2022. URL https://arxiv.org/abs/2211.15841.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 03 1991. ISSN 0899-7667. doi: 10.1162/neco.1991.3.1.79. URL https://doi.org/10.1162/neco.1991.3.1.79.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the em algorithm. In Marinaro, M. and Morasso, P. G. (eds.), *ICANN '94*, pp. 479–486, London, 1994. Springer London. ISBN 978-1-4471-2097-1.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

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, L., Gao, J., and Chen, W. Sparse backpropagation for moe training, 2023. URL https://arxiv.org/abs/2310.00811.

Liu, L., Kim, Y. J., Wang, S., Liang, C., Shen, Y., Cheng, H., Liu, X., Tanaka, M., Wu, X., Hu, W., Chaudhary, V., Lin, Z., Zhang, C., Xue, J., Awadalla, H., Gao, J., and Chen, W. Grin: Gradient-informed moe, 2024. URL https://arxiv.org/abs/2409.12136.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Nguyen, T. Q. and Salazar, J. Transformers without tears: Improving the normalization of self-attention. 2019. doi: 10.5281/ZENODO.3525484. URL https://zenodo.org/record/3525484.

Penedo, G., Kydlíček, H., allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C., Werra, L. V., and Wolf, T. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Shazeer, N. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

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, 2017. URL https://arxiv.org/abs/1701.06538.

Snowflake. Arctic, 2024. URL https://www.snowflake.com/en/blog/arctic-open-efficient-foundation-language-models-snowflake/.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Wang, H., Ma, S., Dong, L., Huang, S., Zhang, D., and Wei, F. Deepnet: Scaling transformers to 1,000 layers, 2022. URL https://arxiv.org/abs/2203.00555.

Wang, L., Gao, H., Zhao, C., Sun, X., and Dai, D. Auxiliary-loss-free load balancing strategy for mixture-of-experts, 2024. URL https://arxiv.org/abs/2408.15664.

xAI. Grok-1, 2024. URL https://github.com/xai-org/grok-1?tab=readme-ov-file.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A., Chen, Z., Le, Q., and Laudon, J. Mixture-of-experts with expert choice routing, 2022. URL https://arxiv.org/abs/2202.09368.

Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N., and Fedus, W. St-moe: Designing stable and transferable sparse expert models, 2022. URL https://arxiv.org/abs/2202.08906.