Dense Backpropagation Improves Training for Sparse Mixture-of-Experts

Ashwinee Panda[1*] **Vatsal Baherwani**[1*] **Zain Sarwar**[2,3] **Benjamin Therien**[2,4]
Sambit Sahu[2] **Tom Goldstein**[1] **Supriyo Chakraborty**[2]
[1]University of Maryland [2]Capital One [3]University of Chicago [4] Mila – Quebec AI Institute

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 expert outputs previously seen over the course of training. 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

The sparsely activated Mixture-of-Experts (MoE) Transformer architecture [Shazeer et al., 2017] has been used by many industry deployments [DeepSeek-AI Team, 2024b, xAI, 2024, Databricks, 2024, Mistral Team, 2024, Snowflake, 2024, DeepSeek-AI Team, 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 and 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. This can be corrected by activating all experts during training, enabling all experts and routing directions to be optimized at once, although this requires much more compute.

In this work we propose **DefaultMoE**: 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 improving performance on a range of standard benchmarks for 2 billion total parameter MoEs trained on 160 billion tokens.

[*]* denotes equal contribution. Correspondence to: ashwinee@umd.edu

2 Background & Related Work

MoEs. The MoE layer replaces the feedforward networks (FFN) of Transformers and consists of two components : **1)** N FFNs (*experts*), $E_0(x), E_1(x), \ldots E_N(x)$ 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 $\mathbb{R}^{d_{\text{token}}} \rightarrow \mathbb{R}^N$ which produces logits; these are normalized using softmax, resulting in a probability distribution over the experts. With the router's 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).

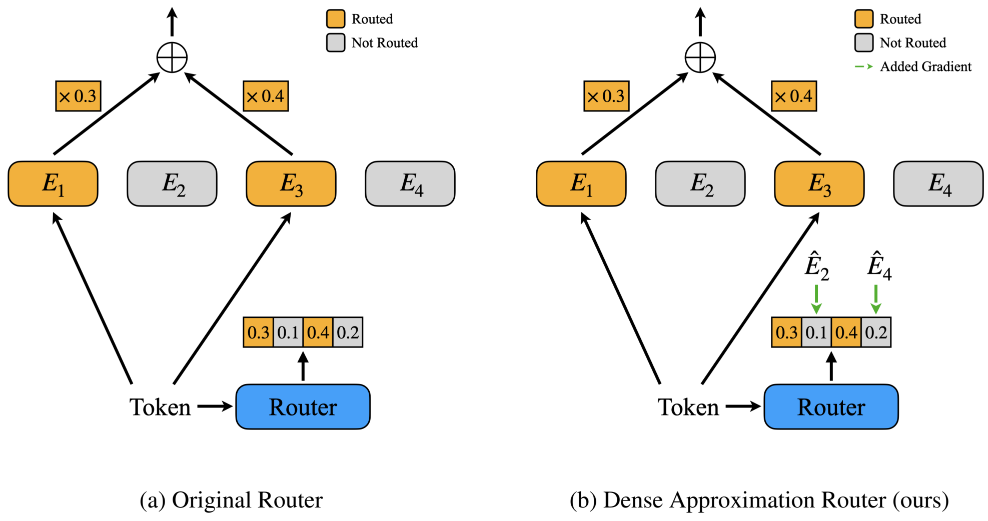
The Router Gradient. Consider the gradient of the MoE layer's output y with respect to the router parameters W. We express y as a function of W by combining Equation (1) and Equation (2). With the chain rule, we can backpropagate through this function by considering the gradient at each respective step:

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

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:

$$\frac{\partial y}{\partial \pi} = [E_1(x), \quad E_2(x) \quad \cdots \quad E_N(x)]^T \tag{4}$$

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.



(a) Original Router (b) Dense Approximation Router (ours)

Figure 1: **Overview of Routing with Dense Approximations**. The original MoE 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.

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.

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 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**. 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. 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. We now evaluate DefaultMoE.

4 Evaluation

We describe the experimental setup in Section 4.1, benchmark our Default MoE in Section 4.2, show that our improvements hold across multiple configurations in Section 4.3, discuss prior work in Section 4.4, analyze our Default MoE in Section 4.5, and find in Section 4.6 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. We conclude with a cohesive explanation of why Default MoE beats TopKMoE in Section 5.

4.1 Experimental Setup

Model Architectures. We use `NcK` to refer to a model with N total experts and K active experts. We train both standard and finegrained [DeepSeek-AI Team, 2024a] MoEs. All of our models have 1.96 billion total parameters, 366 million of which are non-MoE parameters. This leaves 1.6B MoE parameters, and the number of active parameters depends on sparsity. All parameter counts are specified in Table 1. We ablate the model architecture, hidden dim, number of total experts, and number of active experts in Section 4.3.

MoE Config.	8c1	8c2	32c1	32c2	32c4
Active Params.	565M	764M	416M	466M	565M

Table 1: Active parameter counts for each MoE configuration; the total parameter count is 1.96B.

Dataset. We train on FineWeb-Edu [Lozhkov et al., 2024] (for Table 2 and Figure 11) and FineWeb [Penedo et al., 2024] (for all other plots and results) with the Llama3 tokenizer [Llama 3 Team, 2024]. For our main results, we train for 160B tokens, which is a tokens-per-parameter ratio of ≈ 283; at this level of overtraining, many spurious improvements should wash out, so we can be confident that any improvements we observe are real gains.

We have open-sourced our training code. All hyperparameters for the models trained in Table 2 can be found in our config file, and all hyperparameters are the same between the baseline and our DefaultMoE. All further experimental details can be found in Section A.2.

4.2 Main Results

Pretraining Benchmarks. In Table 2 we compare Default MoEs to TopK MoEs. All models are trained for 160B tokens on FineWeb-Edu and have 1.96B params. For both finegrained MoEs (32c4) and standard MoEs (8c1), our Default MoEs outperform TopK MoEs on standard benchmarks. We conduct all evaluations with the lm-eval harness [Gao et al., 2024].

Due to space constraints we defer pretraining curves to the Appendix, but discuss them here. In Figure 11 we compare each MoE's loss curves. Without introducing significant overhead, our method reduces the tokens required to reach a target perplexity of 12.18 by 9%. 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.

[2]We use the ‘Random word from passage" as a baseline for LAMBADA [Paperno et al., 2016].

Benchmark	8c1						32c4						Random Baseline (Score)
	Score			Improvement (%)			Score			Improvement (%)			
	TopK	Default	Diff (%)	TopK	Default	Diff (%)	TopK	Default	Diff (%)	TopK	Default	Diff (%)	
LogiQA	$26.0_{1.7}$	$26.9_{1.7}$	+3.6%	3.8	7.5	+96.0%	$29.0_{1.8}$	$28.3_{1.8}$	-2.6%	16.1	13.1	-19.0%	25.0
MathQA	$26.2_{0.8}$	$25.8_{0.8}$	-1.8%	4.9	3.0	-38.1%	$25.0_{0.8}$	$26.1_{0.8}$	+4.3%	0.1	4.4	+4266.7%	25.0
MMLU	$31.8_{0.4}$	$32.3_{0.4}$	+1.6%	27.3	29.4	+7.5%	$33.7_{0.4}$	$32.5_{0.4}$	-3.6%	34.7	29.9	-14.0%	25.0
OpenBookQA	$37.2_{2.2}$	$38.2_{2.2}$	+2.7%	48.8	52.8	+8.2%	$39.6_{2.2}$	$39.0_{2.2}$	-1.5%	58.4	56.0	-4.1%	25.0
Lambada	$38.6_{0.7}$	$41.2_{0.7}$	+6.6%	3761.8	4016.0	+6.8%	$44.1_{0.7}$	$43.8_{0.7}$	-0.6%	4307.1	4281.9	-0.6%	1.0[2]
SocialIQA	$39.7_{1.1}$	$41.0_{1.1}$	+3.2%	20.3	24.2	+19.1%	$41.1_{1.1}$	$40.7_{1.1}$	-1.1%	24.7	23.3	-5.7%	33.0
HellaSwag	$40.4_{0.5}$	$41.2_{0.5}$	+2.0%	61.6	64.9	+5.4%	$42.9_{0.5}$	$43.3_{0.5}$	+0.9%	71.4	73.0	+2.2%	25.0
ARC	$45.7_{0.9}$	$47.4_{0.9}$	+3.7%	82.8	89.6	+8.3%	$52.0_{0.9}$	$49.9_{0.9}$	-4.2%	108.2	99.5	-8.1%	25.0
Winogrande	$53.6_{1.4}$	$54.8_{1.4}$	+2.2%	7.2	9.6	+33.0%	$55.5_{1.4}$	$56.3_{1.4}$	+1.4%	11.0	12.5	+14.4%	50.0
PubMedQA	$55.2_{2.2}$	$52.4_{2.2}$	-5.1%	67.3	58.8	-12.6%	$54.4_{2.2}$	$57.4_{2.2}$	+5.5%	64.8	73.9	+14.0%	33.0
BoolQ	$58.5_{0.9}$	$62.0_{0.8}$	+6.1%	17.0	24.1	+41.7%	$62.8_{0.8}$	$63.1_{0.8}$	+0.4%	25.6	26.2	+2.1%	50.0
PIQA	$70.8_{1.1}$	$71.9_{1.0}$	+1.5%	41.7	43.7	+5.0%	$72.3_{1.0}$	$73.1_{1.0}$	+1.1%	44.6	46.1	+3.4%	50.0
SciQ	$86.2_{1.1}$	$87.8_{1.0}$	+1.9%	244.8	251.2	+2.6%	$89.1_{1.0}$	$90.3_{0.9}$	+1.3%	256.4	261.2	+1.9%	25.0
Average	$46.9_{0.4}$	$47.9_{0.4}$	+2.1%	52.3	54.9	+5.0%	$49.4_{0.4}$	$49.5_{0.4}$	+0.3%	59.7	59.9	+0.4%	-

Table 2: **Comparison of Pretraining Benchmark Scores for TopK MoE and Default MoE.** After training for 160 billion tokens, Default MoE outperforms TopK MoE across a range of benchmarks for both a standard MoE (right) and a finegrained MoE (left).

Pretraining on FineWeb. In Table 2 and Figure 11 we pretrain on FineWeb-Edu [Lozhkov et al., 2024]. We also pretrain models on FineWeb [Penedo et al., 2024], which is less curated. The rest of the plots in this paper (our ablations) are on FineWeb, in order to enforce that on some level we are not simply overfitting to the pretraining dataset that we use to collect benchmark scores. In Figure 12 we show that when using the learning rate of 7×10^{-4} that we sweep in Figure 2, our DefaultMoE outperforms the baseline TopK. Therefore, our improvements are not due to any unfair hyperparameter tuning when compared to the baseline.

4.3 Ablations

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

Tuning β. We initially tuned β for every configuration independently, and the results of this can be found in Section A.4. We found that more sparse configurations, such as 32c1, required a lower $\beta = 0.65$ whereas a much higher beta of $\beta = 0.999$ was necessary for 32c4 to achieve its best performance. However, we can actually automatically account for the impact of sparsity and granularity on the default vector update by weighting the updates to the default vector by the router logits. When we do this, different values of β, that would otherwise be suboptimal, all converge to the same good performance. Therefore, in our final method, the value of β does not seem to matter as shown in Figure 8.

MoE Architecture Ablations. In Figure 2(a), we train both a standard TopK model and Default MoE with the configurations 8c1, 8c2, 32c1, 32c2, and 32c4, which correspond to sparsity factors of $1/8$, $1/4$, $1/32$, $1/16$ and $1/8$ respectively. Default MoE outperforms a standard TopK MoE at all sparsity configurations. Our improvement is more pronounced for the lowest sparsity (8c2) but still significant for higher sparsity MoEs such as 32c1. Notably, Default MoE takes longer to "warm up" for sparser models; for example, we notice in Figure 13 that the 32c1 Default MoE rapidly catches up and surpasses the TopK MoE around the 10 billion token mark, while the 32c2 MoE is clearly ahead of TopK even at 2 billion tokens.

Tuning the Learning Rate. We want to ensure that we are comparing against a tuned baseline, so we tune the learning rates in Figure 2(b). We use the learning rate that achieves the best performance for the TopK baseline, 7×10^{-4}, for our results in Figure 12. We observe that while it performs better than Default MoE earlier in training (across our learning rate sweep experiments), the results in Figure 12 show that even in this case, Default MoE eventually surpasses TopK. Moreover, we note that the best learning rate for the Default MoE is the larger learning rate of 9×10^{-4}. By contrast, 9×10^{-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.

Default MoE Remains Superior As We Increase the Model Size. Figure 2c 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.

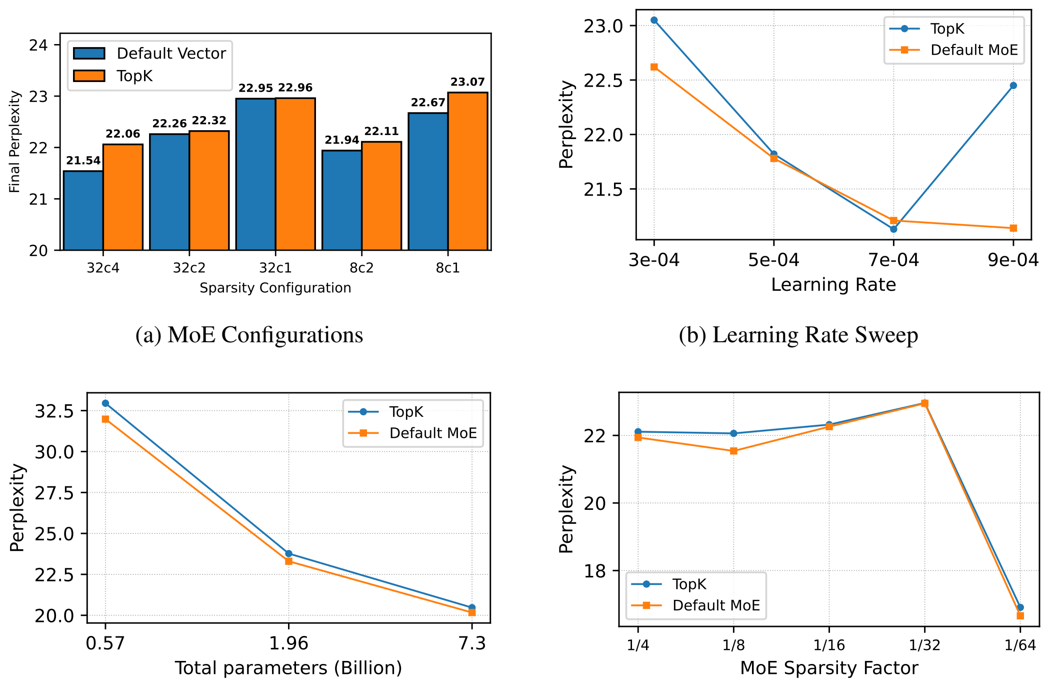


Figure 2: **Comprehensive Ablations.** We report the perplexity after training for 10B tokens for Default MoE and Top-K while varying: **(a)** MoE configuration across five MoE setups. **(b)** Learning rate while keeping the MoE configuration at 8c1. **(c)** Total model size while keeping the MoE configuration at 8c1. **(d)** Sparsity, where the final 1/64 sparsity is for a 7B model, therefore the perplexity is lower than the other sparsity factors that are for 2B models.

We present additional ablations regarding the EMA, including the hyperparameter β, the EMA initialization, and passing the EMA value forward, in Section A.4.

4.4 Comparison to Other Routing Methods

We consider three alternatives to TopK routing for comparison: Sparesemixer [Liu et al., 2023, 2024], ReMoE [Wang et al., 2025], and Loss-Free Balancing [Wang et al., 2024].

Default MoE Beats Sparsemixer. Liu et al. [2023] proposes Sparsemixer, which estimates the true router gradient without straight-through. Liu et al. [2024] note that Sparsemixer lags behind TopK (which our method always outperforms) for the first $0.5T$ tokens, likely due to the noise that Sparsemixer adds. After 24B tokens, Sparsemixer obtains a validation perplexity of 19.77, worse than both TopK and our method. We present the training curve in Figure 5.

ReMoE Fails to Converge. We cannot compare our zero-shot benchmark scores to ReMoE because they train on The Pile, which is unavailable. We train ReMoEs on FineWeb-Edu. ReMoE reports that the first stage of their method is expected to run slowly while their method starts to work, as it is essentially acting as a dense model for the first few hundred steps. Following that, they report that ReMoE exhibits a significant loss spike around 2 billion tokens, after which it recovers and starts to outperform the baseline. In our setting, we indeed were able to replicate the first two results, where ReMoE runs very slowly while acting as a dense model, but the sparsity coefficients start to converge, and the loss spikes at 2 billion tokens. However, the loss never recovered. We invested over 1000 GPU hours into trying to get the ReMoE model to train, but did not have any success.

Loss-Free Balancing Underperforms. We first sweep the bias update coefficient, which is the hyperparameter in the loss-free balancing method used in Wang et al. [2024], and find that 0.0001 provides the best results. However, after training the 8c1 MoE for 10B tokens, Loss-Free Balancing obtains a validation perplexity of 24.22, whereas our Default MoE obtains 23.48.

Why do Prior Methods Underperform? Prior work claims to improve upon standard dMoEs, but there is one difference in our experimental setup: we use globally reduced load balancing loss [Qiu et al., 2025] in all experiments. Corroborating their conclusions, we find that globally reducing the auxiliary loss significantly improves the load balancing, and subsequently the performance, of the baseline, therefore making the improvement of prior methods less noticeable. This is because the "free lunch" of slightly improving routing is more or less eaten by the global auxiliary loss reduction, which is trivial to implement and uses little compute, and prior work frequently targets this same free lunch as their source of improvement. Qiu et al. [2025] directly evaluate Loss-Free Balancing and report that it underperforms a proper globally reduced auxiliary loss, supporting our results.

4.5 Empirical Analysis

In Section B we validate that the load balance, expert coactivation, and domain specialization of the trained MoEs do not deteriorate when we implement our DefaultMoE. Our results corroborate those of Qiu et al. [2025], that even a baseline model trained with globally reduced auxiliary loss will have good load balance and domain specialization.

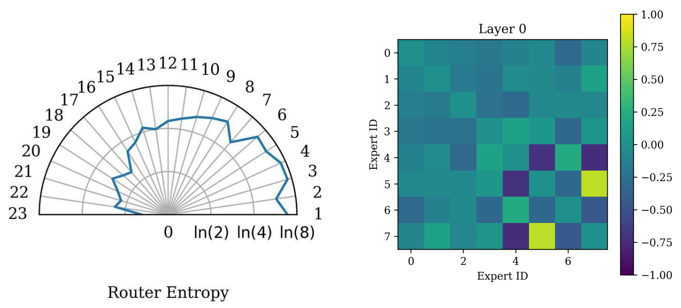
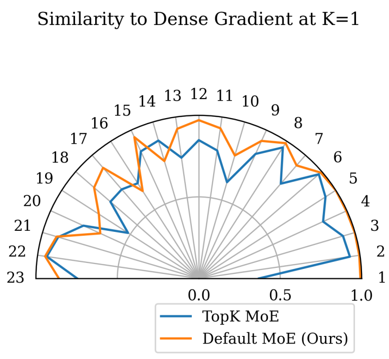


Figure 3: **Analysis.** Left: Router Entropy after training for 10B tokens. Middle, Right: Heatmap of the cosine similarities between default vectors for the 8c1 DefaultMoE at layers 0 and 16.

In Figure 3(a) we observe that the router entropy is highest for the early layers and consistently goes down, with the final layer's router entropy being very low. Intuitively, the router shouldn't quite know where to send a token at the very start of the model, but the routing decisions should become more and more fixed as we proceed through the model.

Similarity of Default Vectors. In Figure 3(a) we plot heatmaps of the cosine similarities between default vectors for experts in different layers of the 8c1 DefaultMoE. The default vectors at layer 0 are not very similar, indicating that the default vectors are learning something nontrivial. However, at layer 16 the default vectors are actually very similar. This follows the trend of the decline in router entropy of Section 4.5; if the router is very confident where a token should go, all the default vectors may learn similarly trivial things, but if the router has high entropy, the default vectors should learn faithful nontrivial approximations of the expert activations.

Similarity to Dense Gradient. Our entire design of DefaultMoE is based on the premise that we can better approximate the dense gradient corresponding to simultaneous activation of all the 8 experts. In Figure 4, we validate that our DefaultMoE's router gradient is more similar to the dense gradient than the TopK MoE's router gradient is. We take the 8c1 MoE pretrained on 160B tokens and compute the dense router gradient, by activating all the 8 experts. We then vary K between 1 and 7 (the full range can be found in Section B) and report the similarity of the sparse router gradient to the dense gradient, for both DefaultMoE and TopK MoE. At K=1, DefaultMoE is clearly much more similar to the dense gradient for the first few layers, while TopK is not close at all. This is because the routing distribution has high entropy at the early layers, so in DefaultMoE, the early



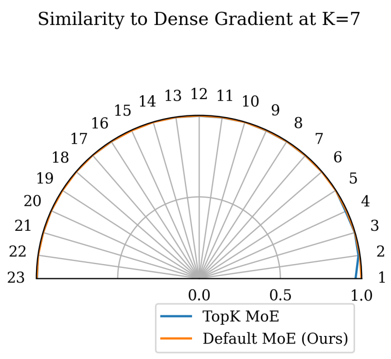


Figure 4: **Dense Router Gradient Similarity.** We plot similarity to the dense router gradient (K=8) for our 8ck Top-K and DefaultMoEs pretrained on 160B tokens. The DefaultMoE's router gradient is much more similar to the dense router gradient.

layers will make significant use of the default vectors. This increases the gradient flow to the router which is reflected in the closer approximation of the dense gradient. Whereas even if the router is allocating near-equal mass on all experts and TopK MoE only selects one, its router gradient will be quite dissimilar from the dense gradient. Even at K=7, TopK MoE is still not quite approximating the dense gradient, indicating importance of the missing information from the least activated expert.

4.6 Efficiency

Our Method Does Not Significantly Reduce Throughput. When we train the 7.33B 8-expert MoE, the per-node GPU throughput is 1,393 tokens per second for TopKMoE and 1,391 tokens per second for DefaultMoE, with a variation of about 1-2 tokens per second on each of these numbers; this is a throughput decrease of just 0.18% which is just variance. For larger models, more of the time will be spent in matmuls and the overhead of our method will be insignificant.

Our Method Does Not Significantly Increase the Memory Footprint. 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. We increase this by 1024, which is a negligible $1/2816 = 0.03\%$ increase in the number of MoE parameters.

Table 3: **Throughput Comparison.** We compare the throughput between DefaultMoE and TopKMoE for different model sizes, measured in tokens per second (sequence length 2048) on 1 GPU.

Hidden Dim	Model Size	Tokens per second TopK	Ours	Overhead vs TopK
1024	1.96B	26,393	25,913	-1.85%
2048	7.33B	1,393	1,391	-0.18%

5 Discussion

We have validated that DefaultMoEs beat tuned TopKMoEs across multiple settings, but *why?* Our explanation is as follows. Our default vectors are serving as nontrivial (Figure 3) approximations of their respective expert activations. Our gradient error should be corrected, which is validated by DefaultMoE's router gradient being closer to the dense router gradient than that of TopKMoE (Figure 4). Less gradient error means that DefaultMoE can tolerate a higher learning rate than TopKMoE (Figure 2), or simply converge faster at TopKMoE's best learning rate (Figure 11) and score higher on benchmarks (Table 2). This improvement is efficient (Table 3), and does not require additional hyperparameter tuning (Figure 8). We hope that our work will help the community train better MoEs.

References

Alex Andonian, Quentin Anthony, Stella Biderman, Sid Black, Preetham Gali, Leo Gao, Eric Hallahan, Josh Levy-Kramer, Connor Leahy, Lucas Nestler, Kip Parker, Michael Pieler, Jason Phang, Shivanshu Purohit, Hailey Schoelkopf, Dashiell Stander, Tri Songz, Curt Tigges, Benjamin Thérien, Phil Wang, and Samuel Weinbach. GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch, 9 2023. URL https://www.github.com/eleutherai/gpt-neox.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

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

Aidan Clark, Diego de las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Katie Millican, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models, 2022. URL https://arxiv.org/abs/2202.01169.

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

DeepSeek-AI Team. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024a. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI Team. Deepseek-v3 technical report, 2024b. URL https://arxiv.org/abs/2412.19437.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR, 2022.

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

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts, 2022. URL https://arxiv.org/abs/2211.15841.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024. URL https://arxiv.org/abs/2403.05530.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. Liger kernel: Efficient triton kernels for llm training. *arXiv preprint arXiv:2410.10989*, 2024. URL https://arxiv.org/abs/2410.10989.

Hunyuan Team. Hunyuan-large: An open-source moe model with 52 billion activated parameters by tencent, 2024. URL https://arxiv.org/abs/2411.02265.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 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.

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

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Liyuan Liu, Jianfeng Gao, and Weizhu Chen. Sparse backpropagation for moe training, 2023. URL https://arxiv.org/abs/2310.00811.

Liyuan Liu, Young Jin Kim, Shuohang Wang, Chen Liang, Yelong Shen, Hao Cheng, Xiaodong Liu, Masahiro Tanaka, Xiaoxia Wu, Wenxiang Hu, Vishrav Chaudhary, Zeqi Lin, Chenruidong Zhang, Jilong Xue, Hany Awadalla, Jianfeng Gao, and Weizhu Chen. Grin: Gradient-informed moe, 2024. URL https://arxiv.org/abs/2409.12136.

Llama 3 Team. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

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

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Mistral Team. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

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

OpenAI Team. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context, 2016. URL https://arxiv.org/abs/1606.06031.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Phi Team. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL https://arxiv.org/abs/2404.14219.

Zihan Qiu, Zeyu Huang, Bo Zheng, Kaiyue Wen, Zekun Wang, Rui Men, Ivan Titov, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Demons in the detail: On implementing load balancing loss for training specialized mixture-of-expert models. *arXiv preprint arXiv:2501.11873*, 2025.

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

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 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/.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers, 2022. URL https://arxiv.org/abs/2203.00555.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts, 2024. URL https://arxiv.org/abs/2408.15664.

Ziteng Wang, Jun Zhu, and Jianfei Chen. Remoe: Fully differentiable mixture-of-experts with relu routing, 2025. URL https://arxiv.org/abs/2412.14711.

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

A Appendix

A.1 Limitations

Throughout our work we only train dropless MoEs. These are ideal for the academic research setting, because they train efficiently in data-parallel and so do not require cumbersome implementations of model or pipeline parallelism. However, dropless MoEs may not be well-suited for industry, because reshaping the tensors to provide consistent shapes to the MoE kernels incurs many host-device syncs. This is a limitation that we share with prior work, but an important future direction will be extending our results to token-dropping MoEs.

We report results on a range of standard language modeling benchmarks, but we have not invested enough compute to see performance on LLM benchmarks such as MMLU-Pro, and our data is not sufficiently domain-specific to report scores on coding benchmarks (HumanEval, MBPP) or math (GSM8k). In the same vein, our models have not been trained to reason. We focus heavily on pretraining in this paper, but an important future direction would be seeing whether we can finetune MoEs with our method.

A.2 Experimental Setup Details

Architecture. 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 and Salazar [2019] for all other layers. We use the AdamW optimizer [Loshchilov and Hutter, 2019]. We use a sequence length of 2048 and a global batch size of 1024, resulting in a global token batch size of 2^{21}. We use a standard cosine decay schedule.

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 Team [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 Triton kernels from [Hsu et al., 2024].

A.3 More details on other Routing Mechanism Comparisons

We see in Figure 5 that our method significantly outperforms SparseMixer for at least the first $10B$ tokens.

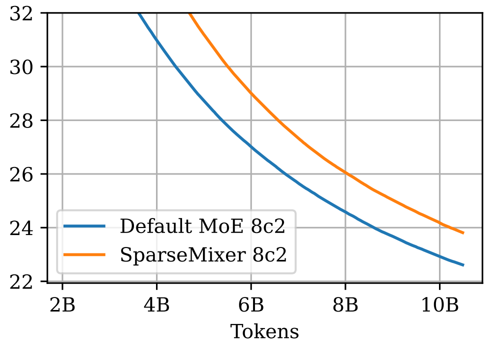


Figure 5: **Comparison of Default MoE and SparseMixer.** We compare Default MoE with SparseMixer, both configured with 8 experts and Top-K=2 active experts. The results report training perplexity throughout training, demonstrating that Default MoE consistently outperforms SparseMixer.

A.4 Ablating EMA design choices

Tuning β. The lone hyperparameter introduced by our method is the β parameter of the EMA. While we use $\beta = 0.9$ for the 8c1 and 8c2 MoEs in Section 4.5, β requires more careful tuning for the 32 expert MoEs. For $N = 32$, we use $\beta = 0.65$, $\beta = 0.95$, and $\beta = 0.999$ for $K = 1$, $K = 2$, and $K = 4$ respectively. In other words, sparser MoEs require a lower β. We believe this is due to each expert receiving less tokens at each step, which leads to a sparser "history" for each default vector. As a result, the default vector's estimate for the average expert output improves when assigning higher weight to the current batch.

We vary the β used in the Default MoE in Figure 6 for an 8 expert MoE 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. For example, we need to tune β more carefully for a 32 expert MoE. In Figure 7 we sweep across multiple values of β for a 32c1, 32c2, and 32c4 Default MoE.

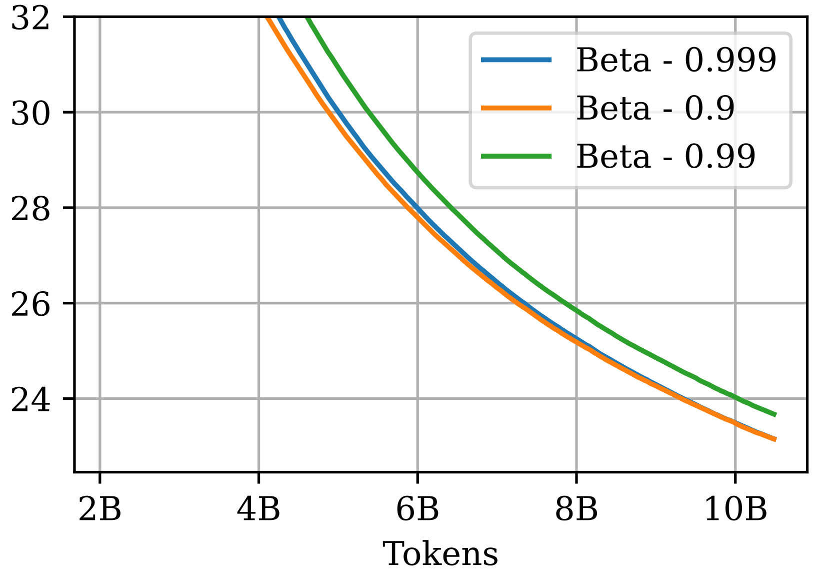
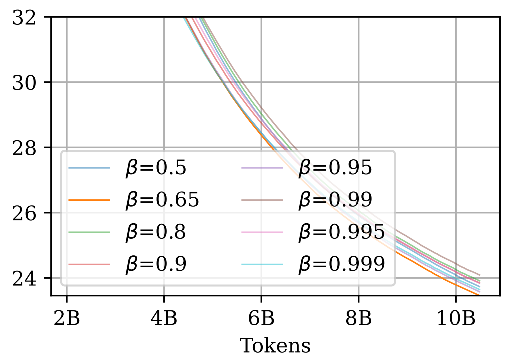


Figure 6: **Selecting the Best EMA Beta Parameter.** We compare different values of the exponential moving average (EMA) smoothing factor β for an 8c1 Default MoE, evaluating $\beta = 0.999$, $\beta = 0.99$, and $\beta = 0.9$. The results show that $\beta = 0.9$ provides the most stable and effective training behavior, leading to lower perplexity. Based on these findings, we use $\beta = 0.9$ to train our 8c1 model for 320B tokens.

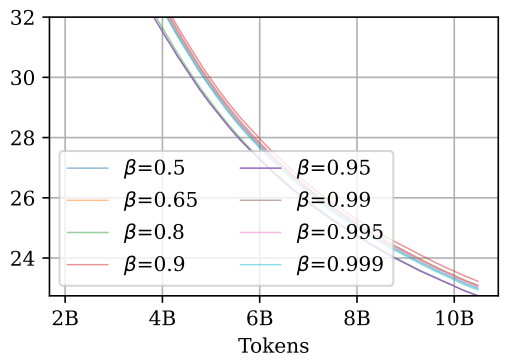
Weighted Updates Remove Sensitivity to EMA Decay Rate (β). We explore different values of the EMA decay coefficient β used to update the default vector. Naively, performance is sensitive to β. However, we find that if we accumulate activations in the the default vector weighted by their router score performance becomes robust to the choice of β. In fact, multiple β values converge to the a similar final performance as shown in Figure 8.

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 9. 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.

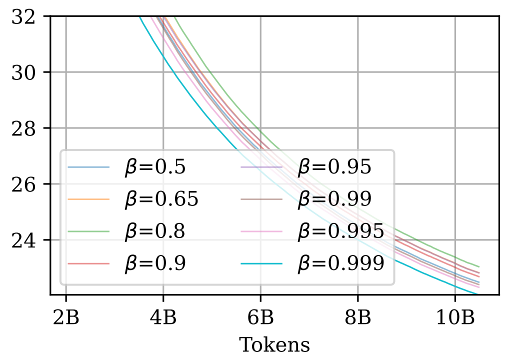
Passing the Default Vector Forward. Our goal is to provide a default expert activation for unactivated experts so that the router can receive a gradient for all experts. We do this by passing the default vector forward through the network, and the router gradient is automatically computed. However, we 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.



(a) $K = 1$



(b) $K = 2$



(c) $K = 4$

Figure 7: **Sweeps for the β hyperparameter for Default MoE with 32 experts.** For $K = 1$ the optimal β is 0.65; for $K = 2$ it is 0.95, for $K = 4$ it is 0.999. Notably, as K decreases, β must also decrease. With roughly 10 billion tokens of training, we can clearly distinguish the best β for any given configuration.

B Detailed Empirical Analysis

B.1 Domain Specialization

We plot the domain specialization for 8c1 TopKMoE (Figure 14,Figure 15,Figure 16), 8c1 DefaultMoE (Figure 20,Figure 21, Figure 22), 32c4 TopKMoE (Figure 17,Figure 18,Figure 19) and 32c4 DefaultMoE (Figure 23,Figure 24,Figure 25). All models are pretrained on FineWeb-Edu for 160B tokens.

B.2 Load Balancing

We plot the load balancing for 8c1 TopKMoE (Figure 26), 32c4 TopKMoE (Figure 27), 8c1 DefaultMoE (Figure 28) and 32c4 DefaultMoE (Figure 29). All models are pretrained on FineWeb-Edu for 160B tokens.

B.3 Expert Coactivation

We plot the expert coactivation for 32c4 DefaultMoE at Layer 0 (Figure 30) and Layer 16 (Figure 31). All models are pretrained on FineWeb-Edu for 160B tokens.

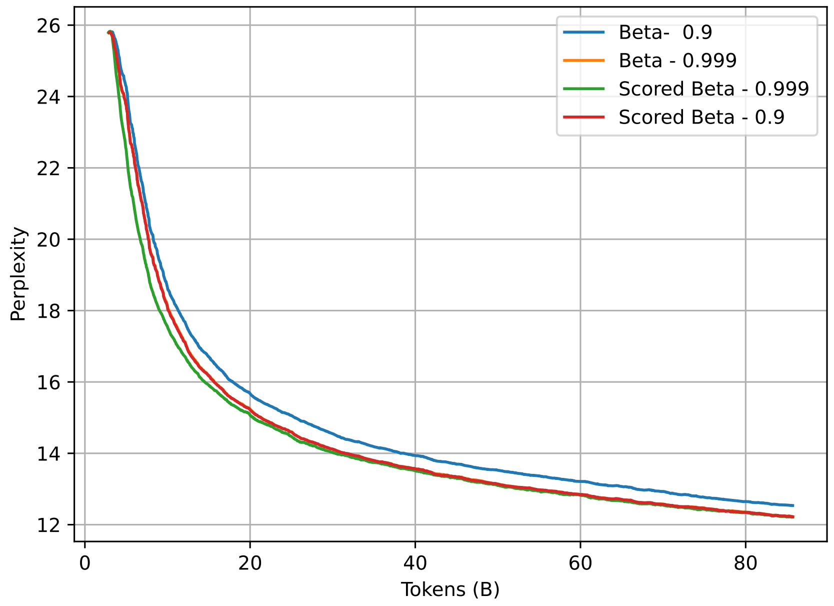


Figure 8: We compare exponential moving average (EMA) updates to the default vector with and without scoring. In the scored version, each expert's contribution is weighted by its router probability before being added to the EMA update. Without scoring, the choice of β (e.g., 0.9 vs. 0.999) significantly affects performance. With scoring, this sensitivity disappears—both scored variants converge to the same strong final performance. The unscored $\beta = 0.999$ curve (orange) is nearly hidden behind the scored curves.

B.4 Dense Gradient Similarity

We plot the similarity to the dense gradient for 8c1 TopKMoE and DefaultMoE at K=1 (Figure 32), K=2 (Figure 33), K=3 (Figure 34), K=4 (Figure 35), K=5 (Figure 36), K=6 (Figure 37) and K=7 (Figure 38). All models are pretrained on FineWeb-Edu for 160B tokens.

B.5 Default Vector Similarities

We plot the cosine similarity of default vectors in 8c1 DefaultMoE (Figure 39, Figure 40, Figure 41) and 32c4 DefaultMoE (Figure 42, Figure 43, Figure 44). All models are pretrained on FineWeb-Edu for 160B tokens.

B.6 Note on References

To avoid an extremely long bibliography, we abbreviated the bibliography entries for very large teams of authors [DeepSeek-AI Team, 2024b, Mistral Team, 2024, Phi Team, 2024, DeepSeek-AI Team, 2024a, Gemini Team, 2024, OpenAI Team, 2024, Llama 3 Team, 2024, Hunyuan Team, 2024]. Full author lists are available at the links provided in the bibliography.

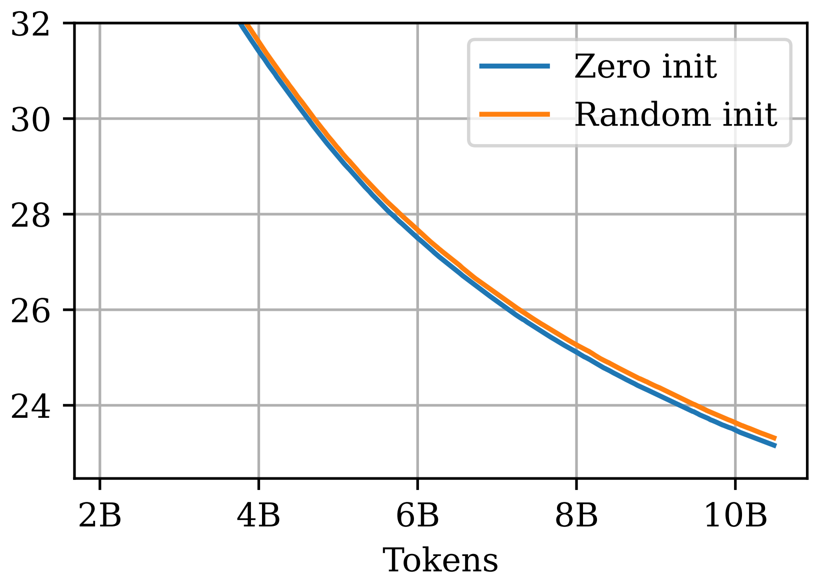


Figure 9: **EMA Buffer Initialization Strategies for Default MoE.** We compare two approaches for initializing the exponential moving average (EMA) buffer for Default MoE: zero initialization and random initialization from a Gaussian distribution. The results indicate that zero initialization is more effective, leading to lower perplexity. Based on this finding, we adopt zero initialization for our experiments.

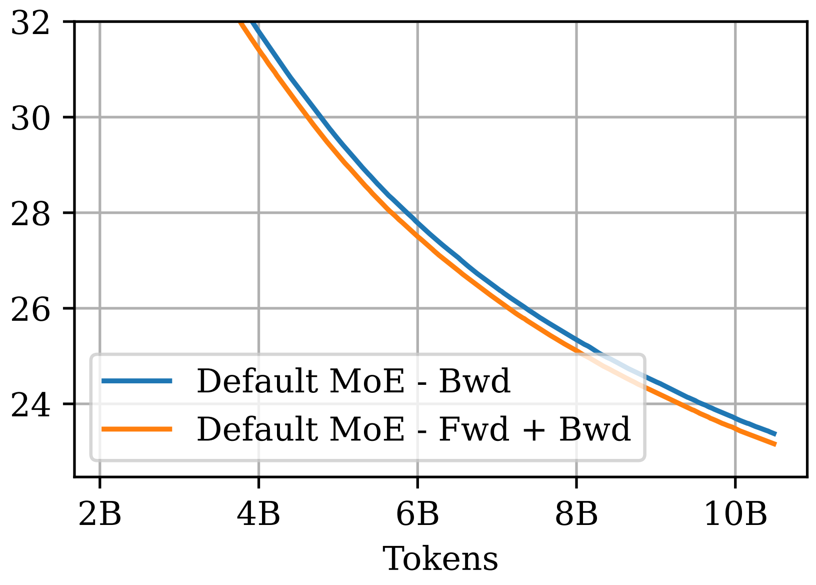


Figure 10: **The importance of applying the EMA during the forward pass.** We perform an ablation study to examine the effect of applying the exponential moving average (EMA) during both the forward and backward passes versus only the backward pass. The results show that using the EMA in both passes 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.

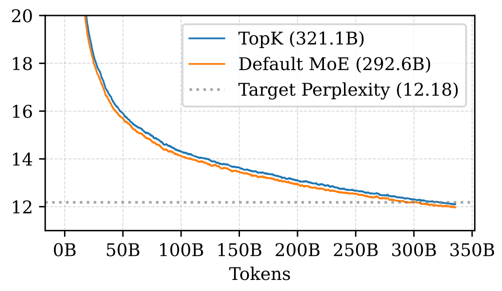


Figure 11: **Default MoE Beats TopK.** Our Default MoE reaches a perplexity of ≈ 12 about 9% faster than the baseline TopK MoE, without introducing any additional overhead. Both MoEs are configured with 8 experts and Top-K=1 active experts.

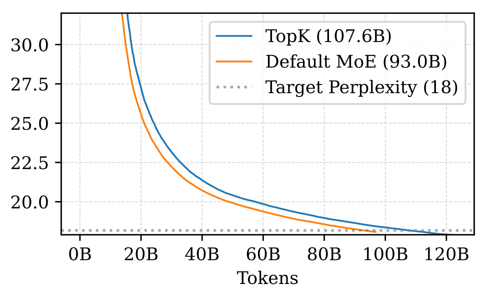


Figure 12: **Default MoE beats a tuned TopK baseline on FineWeb.** Our method reaches the target perplexity of 18 about 15% faster than the baseline TopK method, without introducing any additional overhead.

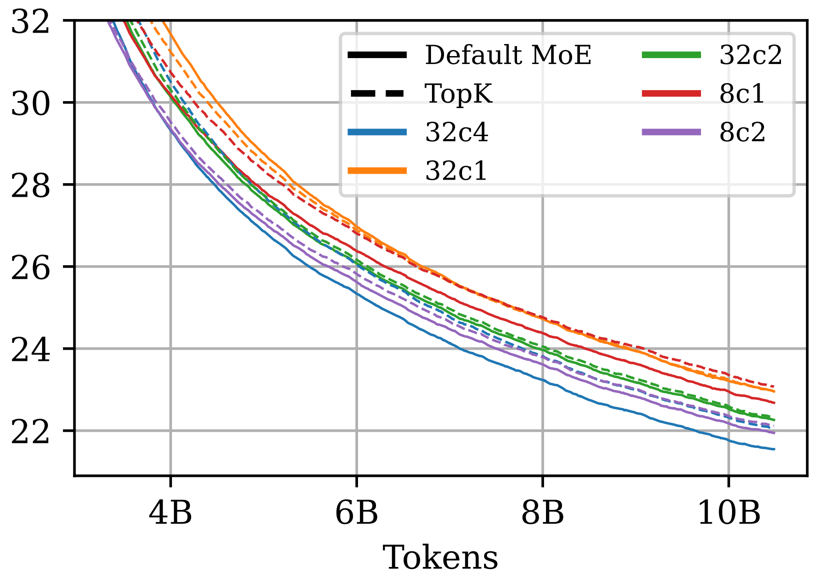


Figure 13: **Comparison of MoE Configurations.** We compare Default MoE and Top-K across five configurations: 8c1, 8c2, 32c1, 32c2, and 32c4. The results show that Default MoE outperforms Top-K in all MoE configurations. Both 8c1 and 8c2 use $\beta = 0.9$; however, 32c1, 32c2, and 32c4 require more careful tuning of β. We use $\beta = 0.65$, $\beta = 0.95$, and $\beta = 0.999$ for 32c1, 32c2, and 32c4, respectively. We detail our choices of β in Section A.4.

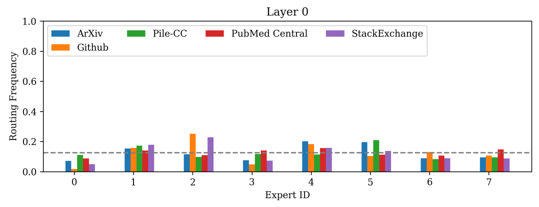


Figure 14: Domain Specialization for 8c1 TopKMoE at Layer 0.

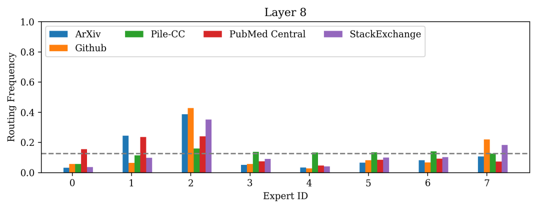


Figure 15: Domain Specialization for 8c1 TopKMoE at Layer 8.

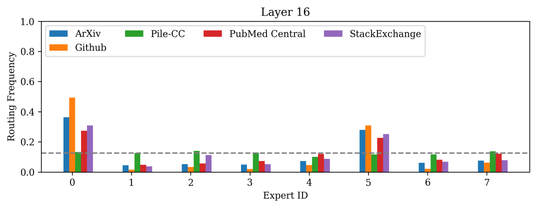


Figure 16: Domain Specialization for 8c1 TopKMoE at Layer 16.

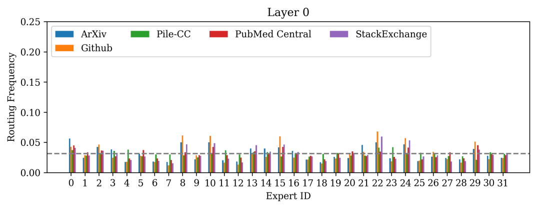


Figure 17: Domain Specialization for 32c4 TopKMoE at Layer 0.

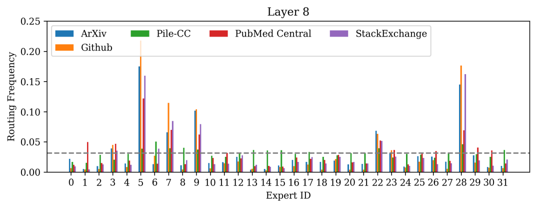


Figure 18: Domain Specialization for 32c4 TopKMoE at Layer 8.

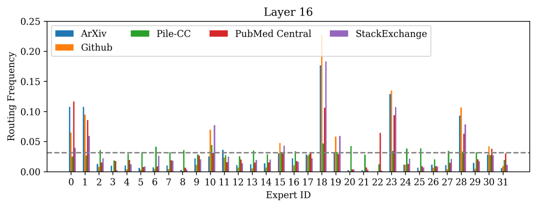


Figure 19: Domain Specialization for 32c4 TopKMoE at Layer 16.

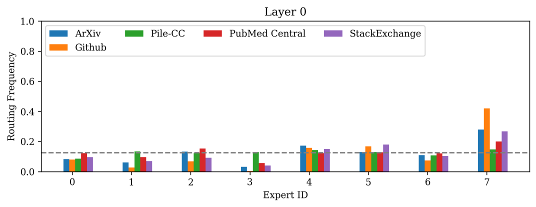


Figure 20: Domain Specialization for 8c1 DefaultMoE at Layer 0.

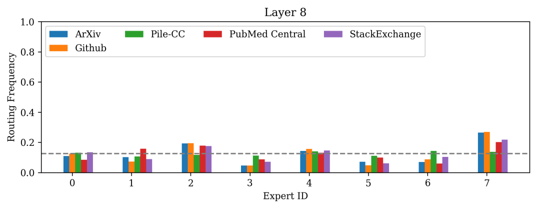


Figure 21: Domain Specialization for 8c1 DefaultMoE at Layer 8.

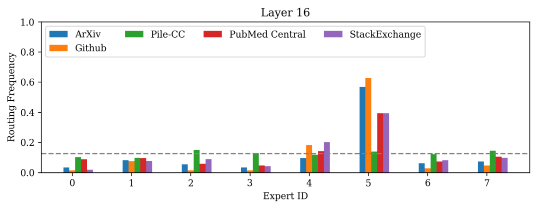


Figure 22: Domain Specialization for 8c1 DefaultMoE at Layer 16.

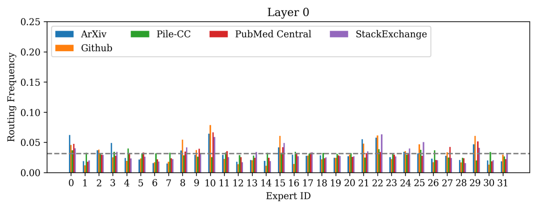


Figure 23: Domain Specialization for 32c4 DefaultMoE at Layer 0.

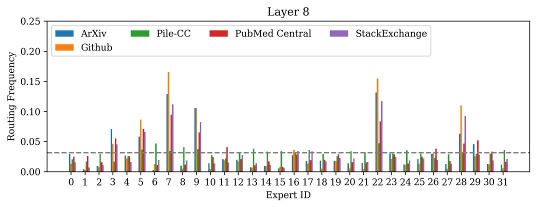


Figure 24: Domain Specialization for 32c4 DefaultMoE at Layer 8.

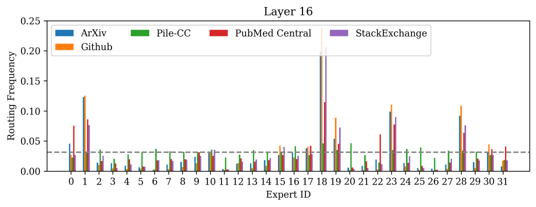


Figure 25: Domain Specialization for 32c4 DefaultMoE at Layer 16.

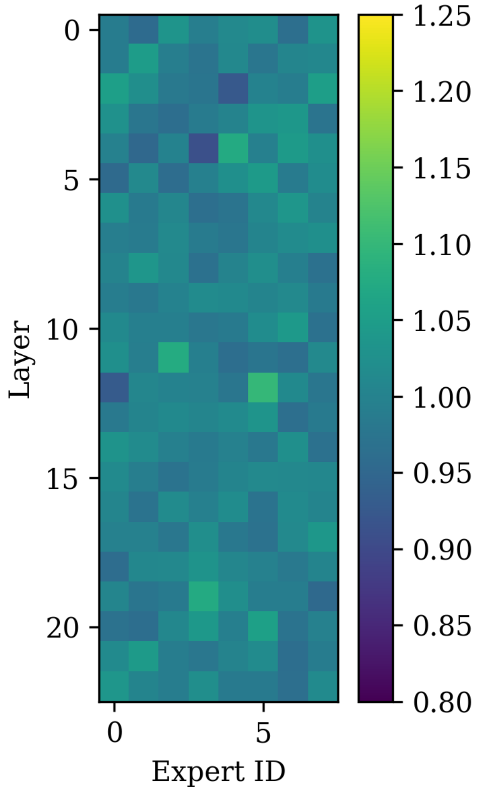


Figure 26: Routing Frequency for 8c1 TopKMoE.

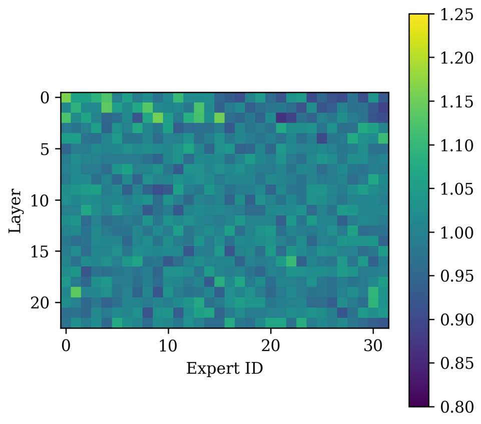


Figure 27: Routing Frequency for 32c4 TopKMoE.

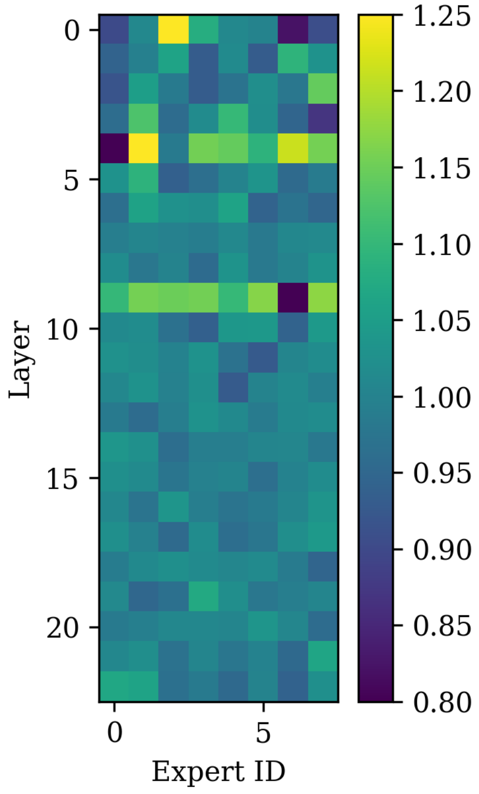


Figure 28: Routing Frequency for 8c1 DefaultMoE.

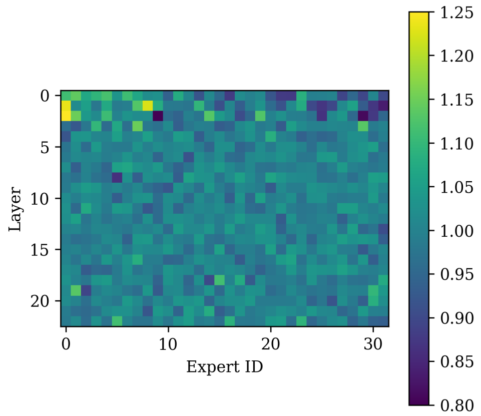


Figure 29: Routing Frequency for 32c4 DefaultMoE.

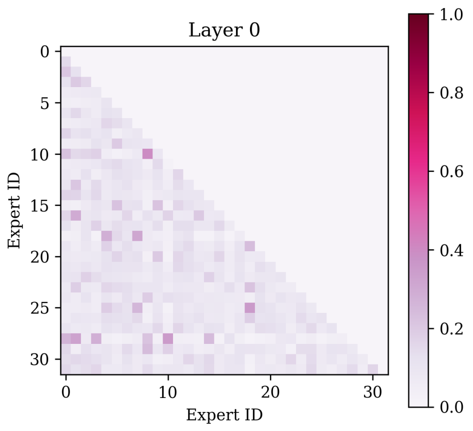


Figure 30: Expert Coactivation for 32c4 DefaultMoE at Layer 0.

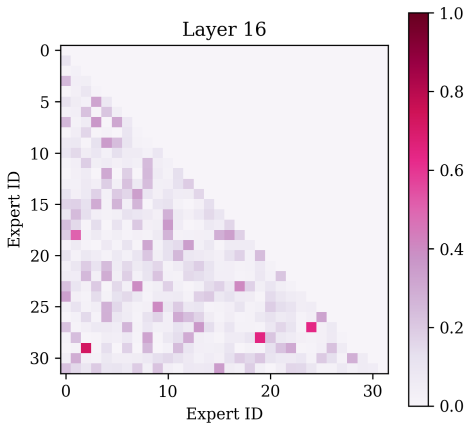


Figure 31: Expert Coactivation for 32c4 DefaultMoE at Layer 16.

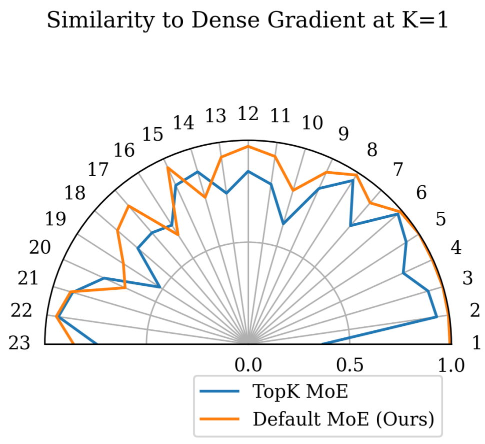


Figure 32: Similarity to Dense Gradient for 8c1 TopKMoE and DefaultMoE at K=1.

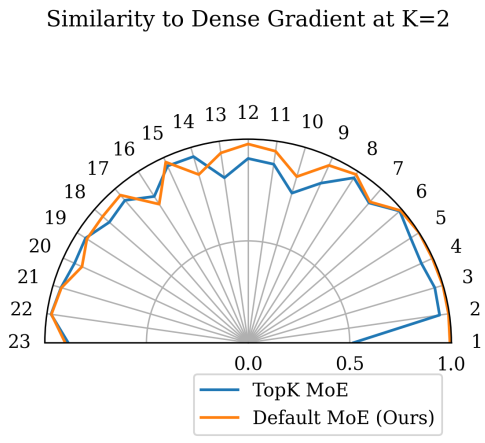


Figure 33: Similarity to Dense Gradient for 8c1 TopKMoE and DefaultMoE at K=2.

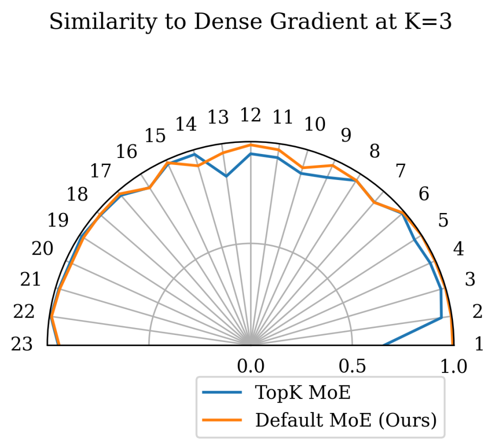


Figure 34: Similarity to Dense Gradient for 8c1 TopKMoE and DefaultMoE at K=3.

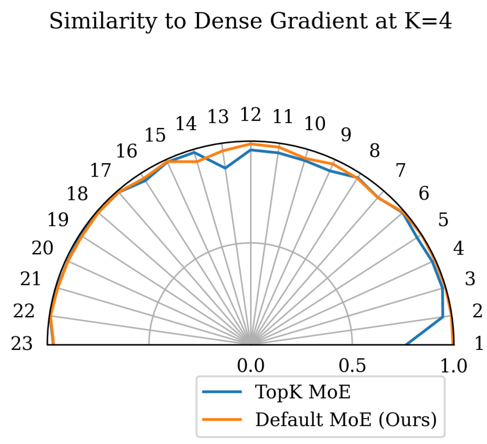


Figure 35: Similarity to Dense Gradient for 8c1 TopKMoE and DefaultMoE at K=4.

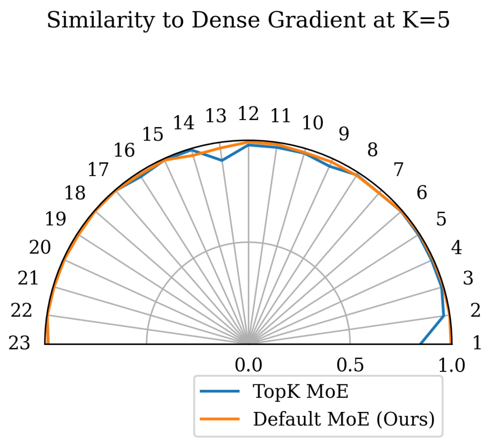


Figure 36: Similarity to Dense Gradient for 8c1 TopKMoE and DefaultMoE at K=5.

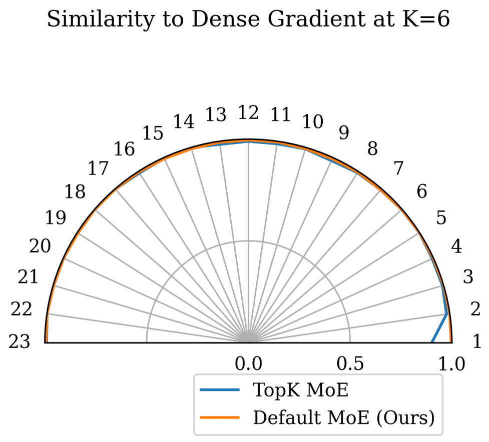


Figure 37: Similarity to Dense Gradient for 8c1 TopKMoE and DefaultMoE at K=6.

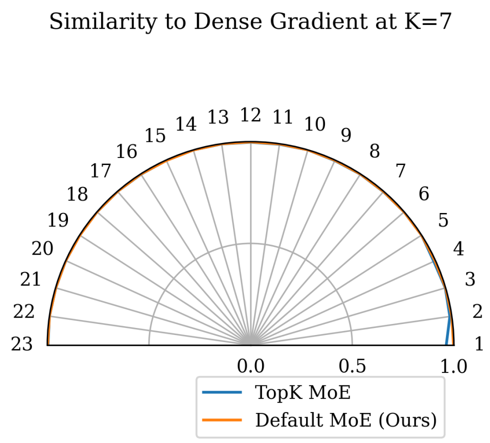


Figure 38: Similarity to Dense Gradient for 8c1 TopKMoE and DefaultMoE at K=7.

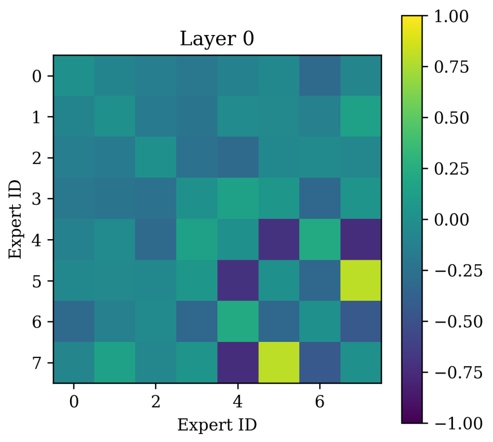


Figure 39: Similarities Between Default Vectors for 8c1 DefaultMoE at Layer 0.

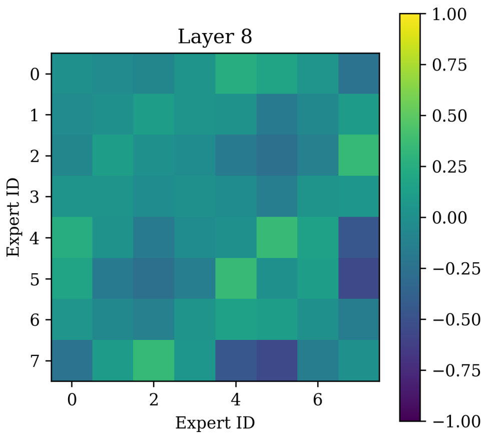


Figure 40: Similarities Between Default Vectors for 8c1 DefaultMoE at Layer 8.

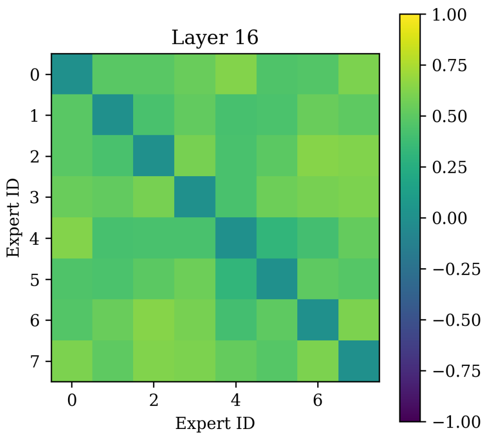


Figure 41: Similarities Between Default Vectors for 8c1 DefaultMoE at Layer 16.

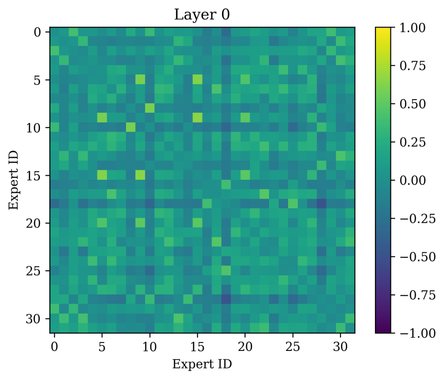


Figure 42: Similarities Between Default Vectors for 32c4 DefaultMoE at Layer 0.

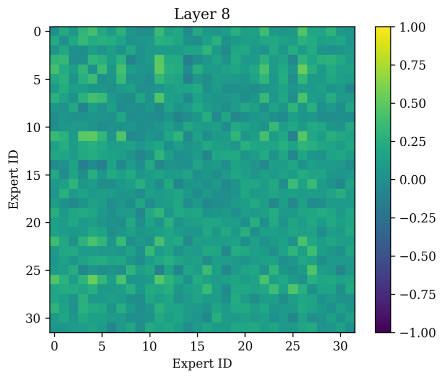


Figure 43: Similarities Between Default Vectors for 32c4 DefaultMoE at Layer 8.

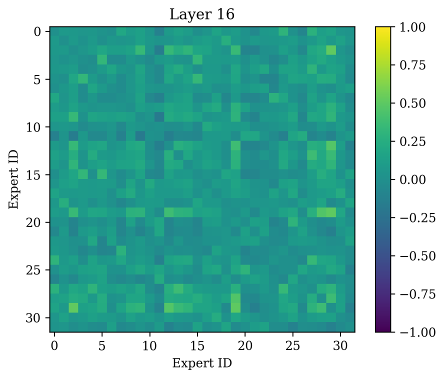


Figure 44: Similarities Between Default Vectors for 32c4 DefaultMoE at Layer 16.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the motivation, summarize methodology, and state the key contributions made in the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The first section of the Appendix states limitations. Other limitations, regarding the assumptions made are also discussed throughout the paper.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: We do not provide theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: In the main paper, each experiment details the setup and datasets used. In addition, we have open sourced our code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide link to the code and also explain the experimental setup in the main body. The data used in our experiments is publicly available.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We describe experimental details in the main body of the paper and also in our open sourced code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: w[Yes]

   Justification: The benchmark results report the standard error given by lm-eval-harness. The experiments focus on large-scale language model pretraining, where each run is computationally expensive and typically conducted once per configuration. As is common in this setting, we report complete training and validation results over training rather than multiple runs.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The LLM experiments were run on 64 GPUs on the AWS cluster. Our code contains configuration files which can be used to estimate compute resources.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: This work is methodological in nature, proposing improvements to training efficiency and stability in Mixture-of-Experts models. As such, it does not directly engage with societal use cases or their potential impacts.

    Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This work does not involve the release of pretrained models, datasets, or tools with direct misuse risk. It focuses on architectural and training improvements for LLMS trained on publicly available data.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We do cite creators and original owners of assets in our paper.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: No new assets are introduced as part of this work.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research involving human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The research presented in this manuscript does not involve the use of LLMs as an important, original, or non-standard component.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.