Finding the Right Optimization for Mixture-of-Experts

Zeynep Tandogan
EPFL

Alexander Hägele
EPFL

Prof. Martin Jaggi
EPFL

Abstract

Mixture-of-Experts (MoE) models offer massive capacity at reduced compute by routing each input to a subset of "experts," but in practice they suffer from severe load imbalance, expressivity loss in infrequently activated experts, and a trade-off between strict balancing and overall performance. In this work, we perform a comprehensive study of MoE optimization techniques, including (i) differentiated learning-rate schedules for expert vs. non-expert parameters, (ii) systematic tuning of auxiliary loss coefficients along with auxiliary loss free method, (iii) comparison of optimizers (e.g., AdamW vs. Shampoo) and activation functions (sigmoid vs. softmax), and (iv) varying the number of experts. Our experiments reveal that certain configurations yield slight improvements in validation loss, perplexity, and accuracy, while other hyperparameter choices can lead to modest performance degradations. Our code is available at https://github.com/zeyneptandogan/moe-project.

1 Introduction

Modern deep learning increasingly relies on architectural innovations that expand a model's representational capacity without a proportional increase in computational cost. Mixture-of-Experts (MoE) models achieve this by activating only a sparse subset of specialized sub-networks ("experts") for each input token. This sparse routing mechanism enables MoEs to scale to hundreds of billions of parameters and becomes the foundation of leading systems such as DeepSeek-V3 and Google's Gemini.

Despite their computational efficiency and expressive power, MoE architectures introduce a host of optimization challenges. Because each expert sees only a fraction of the training data, imbalances in routing can lead to under- or over-utilization of individual experts. Furthermore, an expert's effective batch size fluctuates with the number of activated experts (top-k), complicating the selection of learning rates, update frequencies, and optimizer hyperparameters. Left unaddressed, these issues can slow convergence, degrade final accuracy, or even stall training—effects.

Key open questions include:

- How do auxiliary-loss tuning and "loss-free" balancing methods compare in terms of expert utilization, convergence speed, and final accuracy?
- Can assigning distinct learning rates to expert versus non-expert parameters yield improvements in accuracy, perplexity, and training loss?
- Can per-expert, load-based update schedules more effectively mitigate routing imbalance without harming model performance?
- To what extent do non-diagonal optimizers (e.g., Shampoo) inherently promote uniform expert utilization?

In this project, we address these gaps through a systematic ablation of MoE optimization dynamics. We systematically evaluate auxiliary-loss and loss-free load-balancing strategies to measure their impact on expert utilization and model accuracy; conduct extensive hyperparameter sweeps to derive both static and load-adaptive expert learning-rate schedules; investigate how the Shampoo optimizer influences router behavior and expert load distribution; and integrate and benchmark our MoE workflow in Megatron-LM.

2 Background and Related Work

Mixture-of-Experts (MoE) architectures become the preferred route to scale large language models because they decouple model *capacity* from per-token *compute*. Public releases such as DeepSeek-V2 and DeepSeek-V3 (DeepSeek-AI et al., 2025), Mixtral-8×7B (Jiang et al., 2024), and OLMoE (Muennighoff et al., 2025) demonstrate that sparsity can cut pre-training FLOPs

by 30–50% while matching or surpassing dense baselines. This decoupling enables frontier models to grow in parameter count without proportional increases in training cost. At the core of this efficiency lies conditional routing, a mechanism that activates only a subset of model components, typically a few selected "experts" per token, thus reducing overall compute while maintaining high capacity.

Conditional Routing Foundations Conditional routing is first demonstrated by the Sparsely-Gated MoE layer (Shazeer et al., 2017) (see Figure 1), which increases model capacity at modest overhead by routing each token to its top-k experts via a softmax over router logits. Building on this foundation, later works such as GShard (Lepikhin et al., 2020), which provides a seamless framework for massive model sharding; Switch Transformer (Fedus et al., 2021), which employs lean top-1 gating; and BASE Layers (Lewis et al., 2022), which formulates token-to-expert dispatch as an exact linear-assignment problem, collectively shape the architectural principles driving today's largest sparse models.

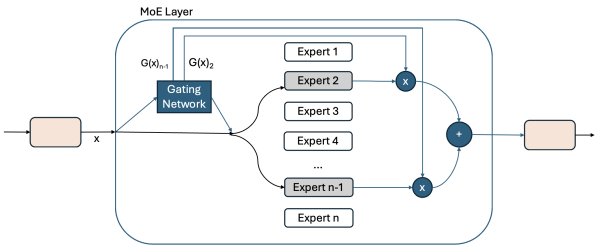


Figure 1: The Sparsely-Gated MoE layer

Top-k vs. Expert-Choice Gating At the front of routing design, two contrasting approaches emerge. The original Sparsely-Gated design often produces skewed expert loads unless corrected by auxiliary balance losses or bias terms. Expert-Choice MoE (Lewis et al., 2022) flips this by assigning each expert a hard token quota per batch and "choosing" its highest-scoring tokens up to that limit, guaranteeing perfect load balance without extra penalties. However, Expert-Choice incurs additional all-to-all communication and collective sorting across all tokens, demands extra memory for index/mask tensors, and can underutilize experts (if quotas are too low) or drop tokens (if quotas are too high), while also restricting the router's flexibility to adaptively reallocate capacity. Thus, it trades the simpler routing and smoother hardware mapping of top-k gating for strict balance at the cost of added system complexity.

Load-Balance and Router Regularization Underlying both approaches is a persistent challenge in MoE training: load imbalance, where some experts receive disproportionately few tokens and remain undertrained. To mitigate this in Sparsely-Gated and related models, early works such as (Shazeer et al., 2017) and the Switch Transformer (Fedus et al., 2021) introduce two auxiliary penalties in addition to the standard cross-entropy loss L_{CE}. In ST-MoE (Zoph et al., 2022), the total loss is formalized as

$$L_{\text{tot}} = L_{\text{CE}} + c_B\, L_B + c_Z\, L_Z,$$

where h_{ij} denotes the pre-softmax logit for expert j on token i, and we define

$$p_{ij} = \frac{\exp(h_{ij})}{\sum_{k=1}^{n} \exp(h_{ik})},$$

$$m_j = \frac{1}{B} \sum_{i=1}^{B} I\{\text{token } i \to j\},$$

$$\ell_j = \frac{1}{B} \sum_{i=1}^{B} p_{ij}.$$

The auxiliary terms are given by

$$L_B = \sum_{j=1}^{n} m_j\, \ell_j, \qquad L_Z = \frac{1}{B} \sum_{i=1}^{B} \Big(\log \sum_{j=1}^{n} e^{h_{ij}} \Big)^2.$$

Here, L_B (the *load-balance* loss) penalizes uneven token allocation across experts, and L_Z (the *router Z-loss*) regularizes the gating logits to prevent numerical instability. Careful tuning of the weights c_B and c_Z is essential: setting them too low fails to correct imbalance, while setting them too high can degrade overall model quality.

Recent work therefore proposes "aux-free" methods that achieve balanced routing without relying on additional loss terms. The motivation stems from a key limitation of existing auxiliary-loss-based approaches: while they help correct load imbalance, large auxiliary terms can introduce non-negligible interference gradients during training, ultimately degrading model performance. To address this, Loss-Free Balancing (Wang et al., 2024) introduces a small, learnable bias vector applied before the softmax gating scores. This bias dynamically shifts tokens toward under-utilized experts, enabling near-perfect load uniformity without the need for auxiliary penalties and without injecting any interfering gradients into the main optimization objective.

Expert-Wise Learning Rates In standard MoE training, all parameters share a single global learning rate, even though each expert only sees a fraction of the tokens and therefore experiences a smaller effective batch size. Hunyuan-Large (Sun et al., 2024) demonstrates that down-scaling the learning rate for low-traffic experts in proportion to their smaller effective batch size markedly stabilises MoE training and improves final accuracy.

System Parallelism Advances Efficient MoE training also depends on advanced parallelism. Megatron-LM combines tensor and pipeline model parallelism, fuses CUDA kernels, and recomputes activations to scale transformers nearly linearly to thousands of GPUs (Shoeybi et al., 2020). DeepSpeed-MoE introduces ZeRO-3 optimizer sharding, overlaps expert all-to-all communication with compute, and provides a compressed checkpoint format that decouples sparse expert parameters from the dense backbone for faster loading and lower storage costs(Rajbhandari et al., 2022). These systems advances make it feasible to train trillion-parameter MoEs in practice.

3 Methodology

3.1 Model Architecture

We implement a decoder-only Transformer in the Llama tradition.

Every hidden state is normalized via RMSNorm, which divides by its root-mean-square magnitude (omitting mean subtraction) and applies a learned scale.

In our dense variant each layer applies pre-norm self-attention followed by a two-layer SiLU-gated feed-forward network: the hidden dimension is expanded to approximately $\frac{8}{3}d$, passed through a SiLU activation and then projected back to d dimensions.

If Mixture-of-Experts (MoE) is enabled, the SiLU-MLP is replaced by a sparse expert layer. During training we augment the cross-entropy loss with auxiliary router losses—entropy regularization, load-balancing, and z-loss—or optionally activate an aux-loss-free mode.

The full model is composed of N sequential Transformer blocks, each following the pattern

$$\text{RMSNorm} \rightarrow \text{Self-Attention} \rightarrow \text{Residual} \rightarrow \text{RMSNorm} \rightarrow \begin{cases} \text{FFN} & \text{(dense)} \\ \text{MoE} & \text{(sparse expert)} \end{cases} \rightarrow \text{Residual}. \tag{1}$$

A final RMSNorm precedes a tied-weight linear head that produces vocabulary logits. During training, we minimize the standard next-token cross-entropy (plus any auxiliary MoE losses); at inference time, we generate tokens autoregressively by re-evaluating the entire context in each step.

3.2 Mixture-of-Experts Block

In our implementation, we have two seperate approaches:

- **Standard Gating (Switch) MoE:** Router computes per-token logits, selects top-k experts (plus shared experts), and aggregates their outputs weighted by softmax.
- **ExpertChoice MoE:** Each expert independently pulls tokens from the router distribution (top-k before or after softmax) to enforce balanced capacity.

In this work, we focus exclusively on the Standard Gating variant, enabling us to isolate and analyze the optimization dynamics introduced by sparse routing without any other factors.

3.3 Routing Mechanisms

At the core of every Mixture-of-Experts (MoE) layer is a *router* that determines which experts will process each input token by computing *router logits*. While selecting the experts, there are two possible strategies: topk_softmax and softmax_topk. These two strategies differ only in how and when these logits are normalized to produce sparse expert weights.

In the **softmax_topk** scheme, one first applies a standard softmax across all E experts to obtain probabilities p_i, then retains only the largest k of these values (setting the remainder to zero), so that each selected expert's weight reflects its share of the global distribution.

By contrast, **topk_softmax** begins by selecting the k experts with the highest unnormalized logits z_i, and subsequently applies a softmax over just those k values, so re-normalizing them in isolation. Although both methods select the same expert indices, they differ in whether the pruned experts continue to influence the final weight magnitudes. In our experiments we utilized topk_softmax.

3.3.1 Auxiliary Routing Losses

To encourage balanced expert usage and stable routing, we compute the following losses on the router logits:

- *Entropy regularization*, which encourages peaked routing distributions.
- *Load-balancing loss*, penalizing uneven token assignments across experts (as in the Switch Transformer).
- *Z-loss*, discouraging large log-partition values in the router to improve numerical stability.

These losses are optionally weighted and summed into the overall training objective, with configurable factors controlling their relative importance. In evaluation mode, we use all three auxiliary losses alongside the cross-entropy loss, but during training we calculate only the load-balancing and Z-loss terms.

3.3.2 Load-Balancing Loss Coefficient Ablation

In our ablation studies, we systematically vary the load-balancing loss coefficient to assess its impact not only on expert utilization but also on overall model performance. We begin with conservative values of 0.0001, 0.001, 0.01, and 0.1, and then extend our exploration to more extreme settings of 0.2, 0.4, and 0.6. For each coefficient, we measure the final training loss, validation perplexity, and accuracy, as well as the MaxVio metric introduced in 3.3.4 to quantify load imbalance.

These comprehensive experiments allow us to observe how increasing the weight of the load-balancing term affects both the smoothness of expert assignment and downstream language modeling performance.

3.3.3 Auxiliary Loss Free Method

As suggested by Wang et al. (Wang et al., 2024), instead of introducing auxiliary regularization losses into the backpropagation pipeline, we add learnable bias term that is apply solely to the router logits during expert selection. The logic is simply that we track each expert's usage, and whenever an expert is underutilized we boost its bias b_k, adding that bias to its router logit so that on the next pass its softmax probability rises and it receives more tokens, thereby maintaining balanced expert workloads.

3.3.4 Maximal Violation (MaxVio)

While re-producing Auxiliary Loss Free method, we also follow their way to analyze the balance througout training and evaluation steps. It basically compares each expert's token load L_i against the ideal load which is the uniform load of the tokens.

$$\text{MaxVio} = \max_i \frac{L_i - \bar{L}}{\bar{L}}$$

In their work, they use two variations of this metric:

- $\text{MaxVio}_{\text{global}}$: L_i accumulated over the entire validation set, reflecting long-term balance.
- $\text{MaxVio}_{\text{batch}}$: L_i measured per training batch (and averaged), capturing short-term fluctuations.

3.4 Different Learning Rates for Non-expert and Expert layers

In our ablation studies, one of our main focuses is on learning rates. In an MoE architecture, since not all experts are active at once, each expert sees far fewer tokens compared to a dense model—in other words, their effective batch size is smaller. Under top-k routing (with E experts and k selected per token), each expert is active only with probability

$$p = \frac{k}{E} \quad \Longrightarrow \quad B_{\text{expert}} = p \times B_{\text{global}}.$$

The gradient signal-to-noise ratio (SNR) scales as

$$\text{SNR} \propto \frac{\|E[\hat{g}]\|}{\sqrt{\text{Var}[\hat{g}]}} \propto \sqrt{B},$$

so because each expert's effective batch size B_{expert} is reduced by a factor of p, its SNR is lower by $\sqrt{p}$, making its gradient updates correspondingly noisier. With this motivation we set up learning rate combinations for non-experts and experts. We perform our analysis based on these combinations. We also have tried to see the relative change in the results when we increase the number of experts while keeping top k is the same. In this analysis, we have updated the learning rate by using cosine scheduler for all.

3.4.1 Load Based LR Update

We then extend this analysis with a new approach: for each expert (and expert–layer combination), we measure its specific token load and use that load as a coefficient to adjust its individual learning-rate update. This coefficient is defined in two different ways, and we evaluate how each variant affects the final outcomes:

1. **Batch-fraction load ratio:**

$$r_i = \frac{L_i}{T},$$

where L_i is the number of assignments to expert i and T is the number of tokens in the batch. Since each token contributes k assignments under top-k routing, $\sum_i L_i = kT$ and thus $\sum_i r_i = k$. Consequently,

$$r_i \in [0,\, 1],$$

and r_i directly indicates the fraction of all token-expert assignments that went to expert i.

2. **Ideal-Normalized load ratio:**

$$r_i = \frac{L_i}{\bar{L}} \quad \text{with} \quad \bar{L} = \frac{\sum_i L_i}{E} = \frac{k\,T}{E},$$

where E is the number of non-shared experts. Here $r_i = 1$ corresponds to perfect balance, $r_i > 1$ indicates overload, and $r_i < 1$ indicates underload. This ratio is bounded below by 0 but has no fixed upper limit.

3.5 Shampoo vs AdamW

In our experiments, we primarily use the AdamW optimizer, a first-order method that maintains lightweight, per-parameter moving averages of gradients and squared gradients while decoupling weight decay for principled ℓ_2 regularization. To evaluate whether a non-diagonal optimizer could promote more balanced expert utilization, we have also experimented with Shampoo (Gupta et al., 2018). Shampoo operates at the matrix (or block) level: it estimates the gradient covariance within each block, computes a low-rank or block-diagonal approximation of its inverse square-root (analogous to a Hessian preconditioner), and applies this "whitening" transform to the gradient. By equalizing curvature across directions, Shampoo often converges substantially faster on very large or deep networks.

For scalability, we use Meta's distributed PyTorch implementation of Shampoo (Shi et al., 2023), which falls back to a base optimizer for exceedingly large parameter blocks. Any block whose dimensions exceed a user-specified threshold is updated with AdamW rather than the Shampoo preconditioner. We set this threshold to encompass all expert weight matrices, ensuring that those submodules benefit from second-order preconditioning, while still preserving efficiency on larger blocks.

4 Experiments

We have conducted our experiments using two separate implementations: (1) our in-house MoE framework built on PyTorch, which enabled rapid prototyping and variant evaluation; and (2) the Megatron-LM codebase, which we use to extend and validate our inferences.

4.1 Experimental Setup

4.1.1 Model Architecture

We adapt a LLaMA-style, 24-layer decoder-only Transformer with 12 heads and hidden dimension 768. Each MoE layer contains 8 experts (Top-2 routing, temperature=1.0), where each expert's FFN has a 2048-dim intermediate.

4.1.2 Datasets

We sample 10B tokens from the FineWeb EDU corpus; 9.95B for training and 4.9M for validation (Lozhkov et al., 2024).

Table 1: Core Hyperparameters

Component	Value
Layers	24 (decoder-only)
Hidden size	768
FFN dim (dense)	2048
Experts per layer	8 (Top-2 routing)
Expert FFN dim	2048
Sequence length	512
Training steps	50,000
Batch size	40 sequences (20 480 tokens/update)
Optimizer	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$)
weight decay	0.1
LR schedule	Cosine decay with 300-step warmup
Gradient clipping	1.0

During our experiments, we performe ablations on four key factors: (i) separate learning-rate schedules for expert vs. non-expert parameters; (ii) optimizer choice (Adam vs. Shampoo); (iii) auxiliary-loss coefficient (including zero); and (iv) number of experts per MoE layer.

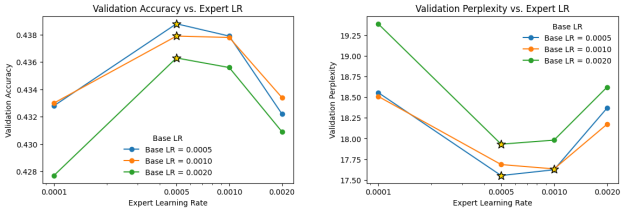
4.2 Results

4.2.1 Learning Rate for Non-expert and Experts

In our setup for this analysis, we use a micro-batch of 40 sequences and accumulate gradients over 4 steps. We have tried several expert and non-expert learning rate combinations. As shown in Figure 2, the final validation metrics exhibit a consistent "U-shaped" dependence on the expert learning rate. Based on the small differences we observe between the uniform-LR and different-LR setups, we cannot conclude any meaningful improvement from using separate expert and non-expert learning rates.

4.2.2 Load Based Learning Rate Update for Experts

Building on our exploration of distinct learning rates for experts, we hypothesize that dynamically adjusting each expert's learning rate based on the loads may further improve training stability and model quality. As described in Subsection 3.4.1, we have implemented two variants of the load-based learning-rate update:



(a) Final Accuracy (b) Final Perplexity

(c) Final Loss

Figure 2: Impact of Expert Learning Rate on Final Validation Metrics. (a) Accuracy, (b) Perplexity, and (c) Loss for varying expert-LR and base-LR.

1. **Batch-fraction load ratio:** The coefficient $r_i = L_i/T$ always lies in $[0, 1]$, so every expert's learning rate is reduced relative to the base—lightly loaded experts suffer the greatest drop.

2. **Ideal-normalized load ratio:** The coefficient $r_i = L_i/\bar{L}$ has no upper bound, allowing $r_i > 1$ to boost overloaded experts and $r_i < 1$ to dampening underloaded ones while keeping the mean scaling at 1.

When we analyze the evaluation metrics for these approaches, we observe that they perform worse than the aux loss 0.1 and aux loss free methods under the same configurations (3).

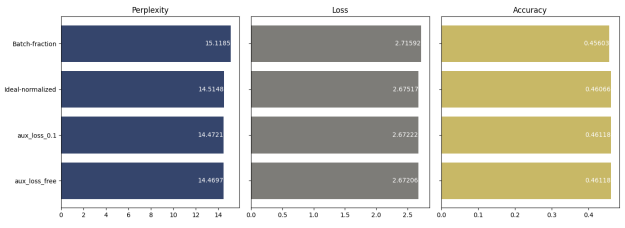


Figure 3: Comparison of Evaluation Metrics in runs with both lrs 0.001

It should be noted that the batch-fraction method poses an underfitting risk, since its reduced learning rate causes the model to learn too slowly, whereas the ideal-normalized method exhibits parameter-norm changes similar to the baseline throughout training (4). It is also seen that MaxVio trend is closer to the aux loss-free method in both learning update variations (5).

Figure 4: Evolution of the overall parameter norm ($\|\theta\|$) over training steps

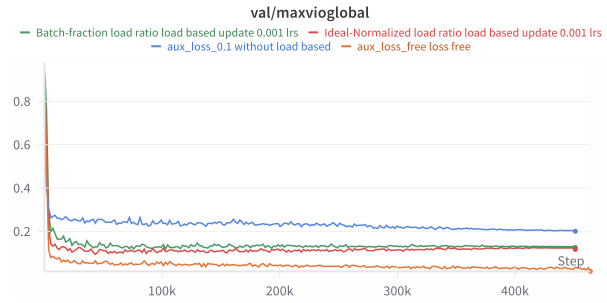
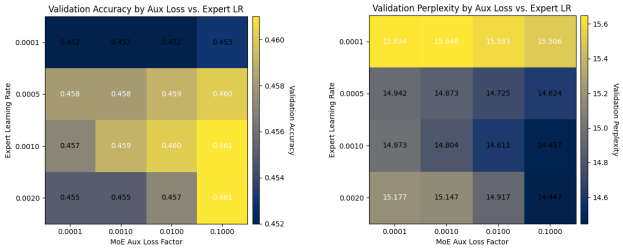


Figure 5: Validation Maxvio Curve

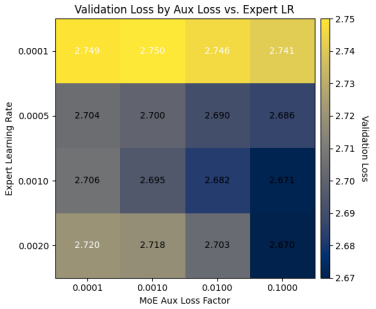
4.2.3 Aux Loss Coefficient Ablations for Different LRs

Our next step is to see the effect of aux loss coefficient for various non-expert LR and LR combinations. Keeping non-expert LR 0.001, we have tried various expert LR and aux loss coefficients as results listed in the following heatmaps in 6. It is clear that a strong auxiliary loss (weight = 0.1) consistently boosts performance. Under this setting, expert learning rates from 5×10^{-4} to 2×10^{-3} all deliver nearly equivalent results, but dropping the expert LR to 1×10^{-4} causes a clear performance decline, even with the high aux coefficient.

In addition, we have run experiments with both expert and non-expert learning rates fixed at 5×10^{-4} while varying the auxiliary-loss coefficient. Consistent with the heatmap analysis, we have found that



(a) Final Accuracy (b) Final Perplexity



(c) Final Loss

Figure 6: Heatmaps showing the impact of auxiliary-loss coefficient and expert learning rate on final validation metrics

increasing the aux-loss weight consistently improves performance: the highest coefficient tested (0.1) produces the best validation accuracy, lowest perplexity, and smallest loss.

To see the change in the results due to the number of experts, we have experimented with 8 and 16 experts while keeping active experts (top-k) as 2. The benefit of adding experts is significantly larger when the auxiliary loss coefficient is high (0.1), there the perplexity drop is 0.394, while at auxiliary = 0.001 it is only 0.054. This suggests that a stronger auxiliary loss signal helps additional experts to specialize more effectively.

Aux Loss	# Experts	Accuracy	Perplexity	Loss
0.1	8	0.4564	15.0520	2.7115
	16	**0.4599**	**14.6582**	**2.6850**
0.001	8	0.4535	15.4165	2.7354
	16	**0.4541**	**15.3626**	**2.7319**

Table 2: Effect of Expert Count and Auxiliary-Loss Coefficient on Validation Performance (both lr and expert lr : 0.0005)

4.2.4 Aux Loss Free Comparisons

To compare with aux loss included experiments, we have implemented aux loss free method and analyzed

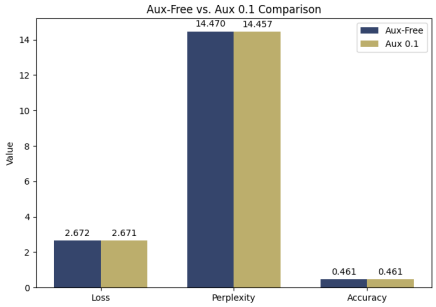
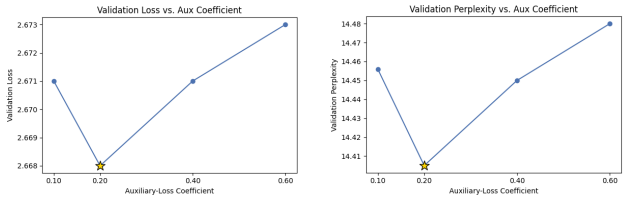


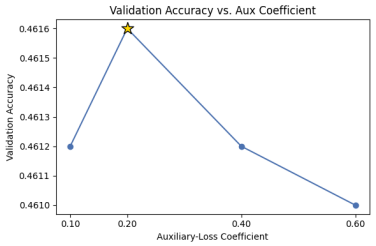
Figure 7: Comparison of validation Loss, Perplexity, and Accuracy between Aux-Free and Aux 0.1.

our evaluation results. We compare that method with the best performing aux loss coefficient which is 0.1. They perform equally well in all metrics as in 7.

Next, we need to see at what point the auxiliary loss coefficient will cause distortion in the results, so we increase the auxiliary loss coefficient up to 0.6. It is seen that the performance starts to decrease after the coefficient is 0.2 (as in 8).



(a) Validation Loss vs. Aux Coefficient (b) Validation Perplexity vs. Aux Coefficient



(c) Validation Accuracy vs. Aux Coefficient

Figure 8: Turning point analysis in Aux Coefficients

When we evaluate MaxVio metric for these, we observe the expected trend of decreasing MaxVio values as the aux coefficient increases.

4.2.5 Evaluation of Shampoo Optimizer

We have integrated Meta's Distributed Shampoo into our training pipeline, allowing Shampoo's second-order updates on the MoE expert layers every 100 steps. For very large parameter matrices, the opti-

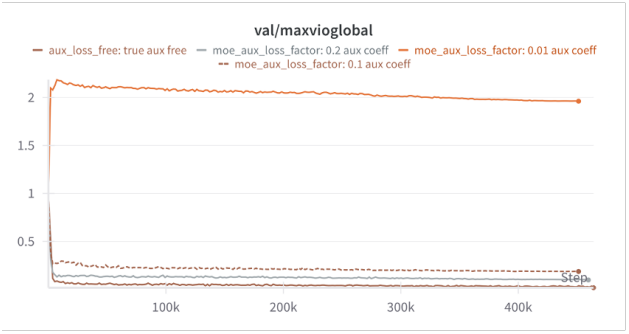


Figure 9: Validation MaxVio as a function of the auxiliary-loss coefficient

mizer automatically falls back to AdamW to prevent inefficiencies and memory errors in the training. With shampoo optimizer, the metric scores are slightly better in loss, perplexity and accuracy (figure 10)

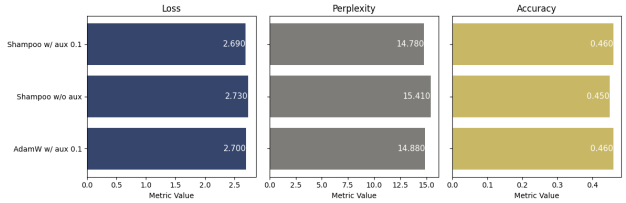


Figure 10: Comparison of optimizer performance for Shampoo (with and without auxiliary loss) versus AdamW.

When we look at the MaxVio changes, shampoo optimizer achieves slightly lower MaxVio in both batch and global settings, indicating improved load balance (in 11).

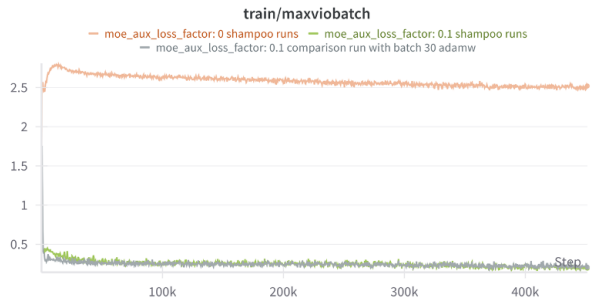
Although we see better results in all evaluation metrics and MaxVio changes with shampoo, the training time is much longer than AdamW due to complexity in the calculations (as shown in Table 3).

Table 3: Wall-clock time for 50 K steps: AdamW vs. Shampoo.

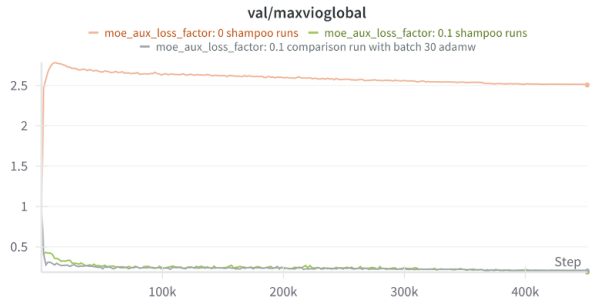
Optimizer	Wall-clock Time	Relative Cost
AdamW	1 d 16 h (40 h)	1×
Shampoo	2 d 2 h (50 h)	1.25×

4.2.6 Further Experiments with Megatron LM

We extend our experiments forward on top of Megatron LM, most of the routing infrastructure is already in place, and we simply added our MaxVio metric. Up to this point all of our runs use a softmax gating function, we have extended our experiments with sigmoid.



(a) Maximum Violation per Batch with Shampoo



(b) Maximum Global Violation with Shampoo

Figure 11: Analysis of violation metrics under distributed Shampoo

As it is suggested by (Wang et al., 2024), sigmoid achieves more uniform distribution between experts compared to the softmax with aux free loss method. However, in the auxiliary-loss-free method, we have found that using a smaller bias update rate (1e-3) with softmax results in a MaxVio of around 1.5, much higher than in the sigmoid version. To achieve lower MaxVio, we need to increase the bias update rate to 1e-2 (table 4).

Table 4: Average loss metrics for 0.001-lr configurations

Configuration	LM Loss	MaxVio	Z Loss
0.001-lr, sigmoid e-3, loss free	**2.871**	0.086	0.003
0.001-lr, sigmoid e-2, loss free	2.893	0.313	0.003
0.001-lr, softmax e-3, loss free	3.068	1.574	0.003
0.001-lr, softmax e-2, loss free	2.890	0.081	0.003
0.001-lr, sigmoid aux 0.1	3.223	0.072	0.010
0.001-lr, softmax aux 0.1	2.896	**0.032**	**0.002**

Table 5 compares two softmax aux 0.1 configurations under an 8-expert setup: one using a uniform learning rate of 0.001 for both non-experts and experts, and the other using 0.001 for the non-experts but 0.0005

Table 5: Average loss metrics for 8-expert configurations (non expert LR 0.001 with softmax aux coefficient 0.1)

Configuration	lm loss	maxvio	z_loss
0.001 expert lr, 0.0005 expert lr	**2.896** 2.960	**0.032** 0.041	**0.002** 0.003

for the experts. The results show that matching both learning rates (0.001) yields both lower LM loss and smaller max violation.

To determine whether this advantage holds at scale, we have repeated the experiment with 128 experts. To utilize our sources without any errors, we have made the model size smaller (having attention heads 12 rather than 24 and moe ffn hidden size 1024 instead of 2048). Although different LR variant achieves a marginally lower LM loss (3.090 vs. 3.110), it suffers from a substantially higher max-violation (0.588 vs. 0.439) compared to the both-expert-LR setup. In other words, it trades a small gain in token-prediction accuracy for a larger imbalance in expert loads (in 6). Our experiment code with Megatron-LM is available at: https://github.com/zeyneptandogan/Megatron-LM.

Table 6: Average loss metrics for 128-expert configurations (non-expert LR 0.001)

Configuration	LM Loss	MaxVio	Load Balancing	Z Loss
0.0005 expert lr	**3.090**	0.587	0.999	0.007
0.001 expert lr	3.109	**0.439**	**0.998**	0.007

5 Discussion

Our findings clarify the specific conditions under which Mixture of Experts (MoE) techniques are effective. Varying the expert learning rate reveals a clear U-shaped response: values around 5×10^{-4} minimize loss, and splitting expert/backbone schedules provides no additional benefit. A modest auxiliary routing loss (0.05–0.1) reliably improves accuracy, perplexity, and MaxVio by encouraging balanced traffic; the benefits plateau beyond 0.2 and reverse as the term begins to dominate optimization. As the number of experts increases, the impact of the auxiliary coefficient becomes more pronounced.

Distributed Shampoo edges out AdamW on every metric and lowers MaxVio, but costs 25 % more wall-clock time. Load-based LR scaling under-performs both aux-based and aux-free baselines: batch-fraction scaling under-fits, while ideal-normalised scaling risks instability without clipping. Megatron experiments confirm that sigmoid gating can match the 0.1-aux setup even without an explicit aux loss, offering a lighter alternative.

Table 7 summarizes our recommended hyperparameter settings alongside the rationale for each choice.

Table 7: Summary of Our Findings

Hyper-parameter	Recommended Setting	Rationale
Base LR (non-experts)	0.001	**Stable** across models and expert counts.
Expert LR	0.0005–0.001	**U-shaped optimum** at 5×10^{-4}; safe up to 2×10^{-3} (aux-loss=0.1); **splitting adds no gain**.
Aux-loss weight	0.05–0.1	**Balances traffic** to improve accuracy, perplexity & MaxVio; **plateaus beyond 0.2** and worsens if too high.
Gating	Sigmoid, top-$k = 2$	Sigmoid+aux(0.1) **yields good balance**; softmax needs **10× more bias updates**.
Load-balance regularizer	Aux-free (or ideal-normalized)	**Lowest MaxVio**; batch-fraction **under-fits**; ideal-norm **unstable without clipping**.
Load-based LR scaling	Not recommended	**Underperforms** aux-based/aux-free; under-fits or unstable.
# Experts	8–16 (128 if FFN dims ↓)	**8–16 optimal**; aux-loss scales with more experts; **128 gives minor gains but worsens balance**.
Optimizer	AdamW (Distributed Shampoo optional)	Shampoo **cuts loss, perplexity & MaxVio** but adds **25% training time**.

6 Conclusion and Future Work

In this paper, we present a systematic exploration of optimization strategies for Mixture-of-Experts (MoE) models, focusing on (i) expert vs. non-expert learning-rate schedules, (ii) auxiliary balancing losses (including an aux-loss-free variant), (iii) choice of optimizer and gating activation, and (iv) expert cardinality. Ultimately, our findings highlight how fine-grained control over learning rates, auxiliary losses, and gating functions can make or break MoE training.

Further studies can be done extending load based update LR approaches, making Batch-fraction LR scaling less aggressive can be useful and effective since it is also so close to aux loss free method based on uniformity. Additionally, evaluating these schemes with a larger number of experts under varied expert-specific learning-rate settings could further reveal how they scale with model capacity.

References

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. (2025). Deepseek-v3 technical report.

Fedus, W., Zoph, B., and Shazeer, N. (2021). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. In *International Conference on Learning Representations (ICLR)*.

Gupta, V., Koren, T., and Singer, Y. (2018). Shampoo: Preconditioned stochastic tensor optimization.

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. (2024). Mixtral of experts.

Lepikhin, D., Shah, M., Shazeer, N., Moore, A., Murray, L., Vaswani, A., Ku, Z., and Kaiser, Ł. (2020). Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations (ICLR)*.

Lewis, M., Lee, Y., Radford, A., Olah, C., and Kim, J. (2022). Base layers: Simplifying training of large, sparse models. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*.

Lozhkov, A., Ben Allal, L., von Werra, L., and Wolf, T. (2024). Fineweb-edu: the finest collection of educational content.

Muennighoff, N., Soldaini, L., Groeneveld, D., Lo, K., Morrison, J., Min, S., Shi, W., Walsh, P., Tafjord, O., Lambert, N., Gu, Y., Arora, S., Bhagia, A., Schwenk, D., Wadden, D., Wettig, A., Hui, B., Dettmers, T., Kiela, D., Farhadi, A., Smith, N. A., Koh, P. W., Singh, A., and Hajishirzi, H. (2025). Olmoe: Open mixture-of-experts language models.

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. (2022). Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale.

Shazeer, N., Cheng, Y., Parmar, N., Uszkoreit, J., Jones, L., Gomez, Andreas N., Kaiser, Ł., and Polosukhin, I. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*.

Shi, H.-J. M., Lee, T.-H., Iwasaki, S., Gallego-Posada, J., Li, Z., Rangadurai, K., Mudigere, D., and Rabbat, M. (2023). A distributed data-parallel pytorch implementation of the distributed shampoo optimizer for training neural networks at-scale.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2020). Megatron-lm: Training multi-billion parameter language models using model parallelism.

Sun, X., Chen, Y., Huang, Y., Xie, R., and et al. (2024). Hunyuan-large: An open-source moe model with 52 billion activated parameters. *arXiv preprint arXiv:2411.02265*.

Wang, L., Gao, H., Zhao, C., Sun, X., and Dai, D. (2024). Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*.

Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N., and Fedus, W. (2022). St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*.