A Note on Sparse Mixture of Experts

Junke Wang

Abstract

This note develops a mathematical description of a modern Mixture of Experts (MoE) layer built from SwiGLU feed-forward networks (FFNs). We first specify Top-K routing and the standard auxiliary balancing loss, then derive how the task and auxiliary objectives affect router logits. We next study DeepSeek's auxiliary-loss-free controller and Kimi K3's Quantile Balancing. Finally, we discuss shared and fine-grained experts, route and gate activations, and the normalization of routed outputs.

1 Math Preliminaries

Readers familiar with these mathematical basics may skip this section. Otherwise, read it before proceeding to the main development.

1.1 Chain rule

If a variable is composed as $y = f(u)$ and $u = g(x)$, then

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}. \tag{1}$$

When the intermediate variable is the vector $\boldsymbol{u} = (u_1, \ldots, u_n)$, we write $y = f(\boldsymbol{u})$ and $\boldsymbol{u} = g(x)$. Then

$$\frac{\partial y}{\partial x} = \sum_{r=1}^{n} \frac{\partial y}{\partial u_r}\frac{\partial u_r}{\partial x}. \tag{2}$$

1.2 Product and quotient rules

For scalar functions $u(x)$ and $v(x)$,

$$\frac{d}{dx}(u(x)v(x)) = u'(x)v(x) + u(x)v'(x), \tag{3}$$

$$\frac{d}{dx}\left(\frac{u(x)}{v(x)}\right) = \frac{u'(x)v(x) - u(x)v'(x)}{\big(v(x)\big)^2}, \quad v(x) \neq 0. \tag{4}$$

1.3 Finite double sums and changing order

For any finite rectangular index set and any scalars, vectors, or matrices $a_{i,j}$, the two summation orders are equal.

$$\sum_{i=1}^{M}\sum_{j=1}^{N} a_{i,j} = \sum_{j=1}^{N}\sum_{i=1}^{M} a_{i,j}. \tag{5}$$

Both expressions add exactly the same MN terms. Only their order differs. Thus no assumption of independence, nonnegativity, or differentiability is needed. All token and expert sums in this note have this finite rectangular form, so this is the only order-exchange rule used later.

1.4 Softmax Jacobian

For a vector $\boldsymbol{z} \in \mathbb{R}^N$, let $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{z})$ with

$$p_j = \frac{\exp(z_j)}{\sum_{l=1}^{N} \exp(z_l)}. \tag{6}$$

For output index ℓ and input-logit index j, its Jacobian is

$$\frac{\partial p_\ell}{\partial z_j} = p_\ell \left(\delta_{\ell j} - p_j\right). \tag{7}$$

where $\delta_{\ell j}$ is the Kronecker delta. Thus, the diagonal derivative is $p_j(1 - p_j)$ when $\ell = j$, while changing logit z_j changes every other probability by $-p_\ell p_j$ when $\ell \neq j$.

Section 5.1 uses this identity to separate the task-loss and auxiliary-loss gradients of the router.

2 Notation

The symbols used throughout this note are fixed as follows.

Symbol	Definition
M	Number of tokens in the routing group over which expert statistics are aggregated; this may pool tokens across data-parallel and sequence-parallel shards
N	Total number of routed experts
K	Number of experts activated per token, with $1 \leq K \leq N$
S	Number of shared width-H expert slices
C	Model hidden dimension
H	Intermediate dimension of each SwiGLU expert
X	Input hidden states, with $X \in \mathbb{R}^{M \times C}$
E_j	The j-th expert, with $E_j : \mathbb{R}^C \to \mathbb{R}^C$ and $j \in \{1, \ldots, N\}$
$\boldsymbol{x}_i$	Input to an expert for token i, with $\boldsymbol{x}_i \in \mathbb{R}^C$
$\boldsymbol{e}_{i,j}$	Output of expert j for token i, defined as $\boldsymbol{e}_{i,j} = E_j(\boldsymbol{x}_i) \in \mathbb{R}^C$
Z	Router-logit matrix in $\mathbb{R}^{M \times N}$
$P_{i,j}$	Full-softmax affinity used by the standard auxiliary-loss construction
$G_{i,j}$	Sigmoid affinity $\sigma(Z_{i,j})$ used by auxiliary-loss-free methods
$\mathcal{R}_i$	Set of K experts selected for token i
$Q_{i,j}$	Gate weight actually applied to selected expert j
b_j	Dispatch-only balancing bias used by auxiliary-loss-free methods
α	Scalar gain applied to the routed branch before the residual add
$A_{i,j}$	Binary assignment variable that equals one when token i selects expert j and zero otherwise
ce_j	Number of token assignments received by expert j
me_j	Mean routing affinity assigned to expert j before Top-K selection

3 Sparse Mixture of Experts

The modern sparsely gated MoE layer uses a trainable router to activate only a small subset of a large expert pool for each input [7].

3.1 SwiGLU Expert

For an input token $\boldsymbol{x} \in \mathbb{R}^C$, a SwiGLU expert contains a gate projection, an up projection, and a down projection.

$$\boldsymbol{g} = W_g \boldsymbol{x}, \qquad W_g \in \mathbb{R}^{H \times C}, \tag{8}$$

$$\boldsymbol{u} = W_u \boldsymbol{x}, \qquad W_u \in \mathbb{R}^{H \times C}, \tag{9}$$

$$E(\boldsymbol{x}) = W_d(\mathrm{SiLU}(\boldsymbol{g}) \odot \boldsymbol{u}), \qquad W_d \in \mathbb{R}^{C \times H}. \tag{10}$$

All bias terms are omitted throughout this note. Here $\odot$ denotes elementwise multiplication, and

$$\mathrm{SiLU}(z) = z\,\sigma(z) = \frac{z}{1 + \exp(-z)}. \tag{11}$$

3.2 Router affinities

Given $X \in \mathbb{R}^{M \times C}$, a linear router produces logits for all tokens and experts:

$$Z = XW_r, \qquad W_r \in \mathbb{R}^{C \times N}, \quad Z \in \mathbb{R}^{M \times N}. \tag{12}$$

The affinity between token i and expert j is

$$P_{i,j} = \mathrm{softmax}(Z_{i,:})_j = \frac{\exp(Z_{i,j})}{\sum_{\ell=1}^{N} \exp(Z_{i,\ell})}. \tag{13}$$

In the standard construction considered below, $P_{i,j}$ is used both as the routing affinity for Top-K selection and as the pre-selection gate score.

In auxiliary-loss-free routing, these roles are separated. Biased scores choose the Top-K routes, while unbiased scores determine the mixture gates.

3.3 Top-K selection and a common renormalization

Let $\mathcal{R}_i$ denote the index set of the K largest affinities for token i.

$$\mathcal{R}_i = \mathrm{TopK}(P_{i,:}, K). \tag{14}$$

After discarding the remaining $N - K$ experts, the retained affinities no longer sum to one. In the standard construction used for the gradient analysis below, we renormalize them over the selected set:

$$Q_{i,j} = \frac{P_{i,j}}{\sum_{\ell \in \mathcal{R}_i} P_{i,\ell}}, \qquad j \in \mathcal{R}_i. \tag{15}$$

Let $S_i^{\mathcal{R}} = \sum_{\ell \in \mathcal{R}_i} P_{i,\ell}$. Without renormalization, the selected experts contribute total mass $S_i^{\mathcal{R}} \leq 1$. At initialization, router logits are usually of similar size, hence $P_{i,j} \approx 1/N$ and $S_i^{\mathcal{R}} \approx K/N$. For $K \ll N$, the unnormalized sparse output would therefore be approximately K/N times the normalized mixture. Thus, the sparse FFN branch can initially be much smaller than intended relative to the residual stream. Renormalization is nevertheless an architectural choice, not part of Top-K selection itself; Section 7.3 separates these operations explicitly.

4 Load Balancing Objectives

4.1 Expert statistics

We first introduce a binary assignment variable

$$A_{i,j} = \begin{cases} 1, & \text{if token } i \text{ selects expert } j, \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

This variable records the Top-K decision in the simplest possible form. For a fixed expert j, we add its assignment variables over all M tokens.

$$\mathrm{ce}_j = A_{1,j} + A_{2,j} + \cdots + A_{M,j} = \sum_{i=1}^{M} A_{i,j}. \tag{17}$$

Therefore, ce_j is simply the number of tokens assigned to expert j. Because each token creates exactly K assignments,

$$\sum_{j=1}^{N} \mathrm{ce}_j = MK. \tag{18}$$

The second statistic is the mean affinity of expert j before Top-K selection.

$$\mathrm{me}_j = \frac{P_{1,j} + P_{2,j} + \cdots + P_{M,j}}{M} = \frac{1}{M} \sum_{i=1}^{M} P_{i,j}. \tag{19}$$

We also have

$$\sum_{j=1}^{N} \mathrm{me}_j = \frac{1}{M} \sum_{j=1}^{N} \sum_{i=1}^{M} P_{i,j} = \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{N} P_{i,j} \tag{20}$$

$$= \frac{1}{M} \sum_{i=1}^{M} 1 = 1. \tag{21}$$

The distinction between ce_j and me_j is what makes the later auxiliary-loss gradient possible. During differentiation, ce_j is treated as a fixed measurement of realized load, while me_j changes smoothly with every router logit. Thus $\mathrm{ce}_j \mathrm{me}_j$ uses hard counts to measure imbalance and soft affinities to provide a differentiable correction.

4.2 Standard auxiliary loss

A standard load balancing objective combines the realized assignment fraction with the mean router probability, as in Switch Transformer [3]. In the count-based notation above, it is

$$\mathcal{L}_{\mathrm{aux}} = \lambda \sum_{j=1}^{N} \mathrm{sg}[\mathrm{ce}_j] \mathrm{me}_j. \tag{22}$$

Here λ is the balancing coefficient and sg denotes the stop-gradient operator.

5 Backward Analysis of the Router Objectives

5.1 What Top-K stops and what remains differentiable

Let $\boldsymbol{w}_j \in \mathbb{R}^C$ denote the j-th column of the router weight W_r. For token i,

$$Z_{i,j} = \boldsymbol{x}_i^\top \boldsymbol{w}_j, \qquad P_{i,j} = \mathrm{softmax}(Z_{i,:})_j, \tag{23}$$

and the router performs

$$P_{i,:} \xrightarrow{\text{TopK}} \mathcal{R}_i \xrightarrow{\text{renormalize selected } P} Q_{i,:} \xrightarrow{\text{mix experts}} \boldsymbol{y}_i. \tag{24}$$

During backward, $\mathcal{R}_i$ is held fixed. Let $\boldsymbol{\theta}_j$ denote the parameters of expert j, and let $b \notin \mathcal{R}_i$. Two gradient boundaries follow immediately:

$$\boldsymbol{y}_i = \sum_{j \in \mathcal{R}_i} Q_{i,j} E_j(\boldsymbol{x}_i; \boldsymbol{\theta}_j) \quad \Longrightarrow \quad \frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{\theta}_b} = 0, \tag{25}$$

because Top-K dispatch excludes the unselected expert from the computation, and

$$Q_{i,j} = \frac{\exp(Z_{i,j})}{\sum_{m \in \mathcal{R}_i} \exp(Z_{i,m})} \quad \Longrightarrow \quad \frac{\partial Q_{i,j}}{\partial Z_{i,b}} = 0, \qquad j \in \mathcal{R}_i, \tag{26}$$

because post-selection renormalization removes all unselected logits from the gate expression. Hence token i sends no direct task gate gradient to the unselected router column $\boldsymbol{w}_b$.

Selected experts and selected route columns remain differentiable.

5.2 How the task loss affects gates and routes

Backpropagation from the layers above first provides the gradient with respect to the MoE output. Since the output is a weighted sum of the selected expert outputs, the chain rule from $\boldsymbol{y}_i$ to one selected gate gives

$$\begin{aligned} \boldsymbol{d}_i &:= \nabla_{\boldsymbol{y}_i} \mathcal{L}_{\text{task}} \in \mathbb{R}^C, \qquad \boldsymbol{y}_i = \sum_{m \in \mathcal{R}_i} Q_{i,m} \boldsymbol{e}_{i,m}, \\ \frac{\partial \mathcal{L}_{\text{task}}}{\partial Q_{i,j}} &= \left(\nabla_{\boldsymbol{y}_i} \mathcal{L}_{\text{task}}\right)^\top \frac{\partial \boldsymbol{y}_i}{\partial Q_{i,j}} = \boldsymbol{d}_i^\top \boldsymbol{e}_{i,j} =: s_{i,j}, \qquad j \in \mathcal{R}_i. \end{aligned} \tag{27}$$

Thus $s_{i,j}$ is not an additional assumption: it is the task derivative of gate $Q_{i,j}$, or equivalently the alignment between expert j's output and the downstream task gradient.

The selected gates form a softmax, so changing one selected logit changes all selected gates. Applying its Jacobian yields, for $j \in \mathcal{R}_i$,

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{task}}}{\partial Z_{i,j}} &= \sum_{m \in \mathcal{R}_i} \frac{\partial \mathcal{L}_{\text{task}}}{\partial Q_{i,m}} \frac{\partial Q_{i,m}}{\partial Z_{i,j}} \\ &= \sum_{m \in \mathcal{R}_i} s_{i,m} Q_{i,m} (\delta_{mj} - Q_{i,j}) \\ &= Q_{i,j} \left(s_{i,j} - \underbrace{\sum_{m \in \mathcal{R}_i} Q_{i,m} s_{i,m}}_{=: \bar{s}_i} \right). \end{aligned} \tag{28}$$

The reference value $\bar{s}_i$ therefore arises from softmax coupling: it is the gate-weighted mean of the selected experts' task derivatives. Combining the selected and unselected cases gives

$$\boxed{g_{i,j}^{\text{task}} := \frac{\partial \mathcal{L}_{\text{task}}}{\partial Z_{i,j}} = \begin{cases} Q_{i,j}(s_{i,j} - \bar{s}_i), & j \in \mathcal{R}_i, \\ 0, & j \notin \mathcal{R}_i. \end{cases}} \tag{29}$$

The sign is easiest to interpret geometrically. Because $\boldsymbol{d}_i = \nabla_{\boldsymbol{y}_i} \mathcal{L}_{\text{task}}$ points toward increasing loss, $s_{i,j} = \boldsymbol{d}_i^\top \boldsymbol{e}_{i,j}$ is a marginal cost: smaller values are better. The baseline $\bar{s}_i$ is the selected-gate-weighted mean of these costs. If $s_{i,j} < \bar{s}_i$, then $g_{i,j}^{\text{task}} < 0$, so gradient descent raises $Z_{i,j}$ and increases the expert's selected gate weight. If $s_{i,j} > \bar{s}_i$, the update lowers the logit and suppresses that expert. Thus, the task gradient redistributes weight within the fixed selected set according to relative task utility, with $\sum_{j \in \mathcal{R}_i} g_{i,j}^{\text{task}} = 0$.

There is no gradient through the discrete Top-K set itself. For this token, an unselected logit has $g_{i,j}^{\text{task}} = 0$ because it does not appear in the post-selection mixture. Nevertheless, updating the shared router weights changes future logit rankings, so a later forward pass may select a different route. In the special case $K = 1$, $Q_{i,j} = 1$ and $s_{i,j} = \bar{s}_i$, hence the task path supplies no router gradient at all.

5.3 How the auxiliary loss changes the full score distribution

Recall

$$\mathcal{L}_{\text{aux}} = \lambda \sum_{j=1}^{N} \text{sg}[\text{ce}_j] \text{me}_j = \frac{\lambda}{M} \sum_{i=1}^{M} \sum_{j=1}^{N} \text{sg}[\text{ce}_j] P_{i,j}, \qquad \text{ce}_j = \sum_{i=1}^{M} \mathbb{1}[j \in \mathcal{R}_i]. \tag{30}$$

Here ce_j is the raw assignment count; any conventional normalization by M, K, or N can be absorbed into λ and does not change the gradient direction. For each token, define its affinity-weighted average load

$$\bar{c}_i := \sum_{\ell=1}^{N} P_{i,\ell} \text{sg}[\text{ce}_\ell]. \tag{31}$$

$\bar{c}_i$ is a token-dependent comparison baseline, not the global target MK/N: the same batch counts are averaged using token i's current affinity distribution. Holding those counts fixed and applying the full-softmax Jacobian gives

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{aux}}}{\partial Z_{i,j}} &= \frac{\lambda}{M} \sum_{\ell=1}^{N} \text{sg}[\text{ce}_\ell] P_{i,\ell} (\delta_{\ell j} - P_{i,j}) \\ &= \frac{\lambda}{M} P_{i,j} \left(\text{sg}[\text{ce}_j] - \bar{c}_i \right) =: g_{i,j}^{\text{aux}}. \end{aligned} \tag{32}$$

Unlike the task gradient, this signal exists for selected and unselected experts. Gradient descent lowers the logit of an expert whose count is above $\bar{c}_i$ and raises it when the count is below $\bar{c}_i$. Thus the update is a relative competition under the softmax, rather than an independent pull of each count toward MK/N. The factor $P_{i,j}$ makes the correction weak when the current affinity is tiny, and $\sum_j g_{i,j}^{\text{aux}} = 0$. When all expert counts are equal, the gradient vanishes.

The hard counts are measurements and receive no gradient. Consequently, $\mathcal{L}_{\text{aux}}$ does not directly assess expert quality or differentiate through the current assignment. It changes the next route by reshaping all logits, and it can also change the relative selected gates because those gates are computed from the same logits.

6 Auxiliary-Loss-Free Routing Methods

6.1 DeepSeek-V3

6.1.1 DeepSeek auxiliary-loss-free balancing

For token i and routed expert j, DeepSeek-V3 converts the router logit $Z_{i,j}$ into an affinity score [10, 2]:

$$G_{i,j} = \sigma(Z_{i,j}). \tag{33}$$

Here σ is the sigmoid function. DeepSeek-V3 normalizes only the selected affinities when forming the mixture weights below. An expert-specific bias b_j is added when selecting the K experts:

$$\mathcal{R}_i = \mathrm{TopK}(G_{i,:} + \boldsymbol{b}, K), \tag{34}$$

where $\boldsymbol{b} = (b_1, \ldots, b_N)$. The mixture weights are computed from the original affinity scores:

$$Q_{i,j} = \frac{G_{i,j}}{\sum_{\ell \in \mathcal{R}_i} G_{i,\ell}}, \qquad j \in \mathcal{R}_i. \tag{35}$$

The bias affects expert selection but is not included in $Q_{i,j}$, and it is not optimized by backpropagation.

After training step t, let $\mathrm{ce}_j^{(t)}$ be the number of assignments received by expert j in the batch. Since M tokens each select K experts, the mean assignment count is $\bar{c} = MK/N$. The bias is updated as

$$b_j^{(t+1)} = b_j^{(t)} + \gamma \, \mathrm{sign}\left(\bar{c} - \mathrm{ce}_j^{(t)}\right), \qquad \gamma > 0. \tag{36}$$

An overloaded expert has $\mathrm{ce}_j^{(t)} > \bar{c}$ and receives a smaller bias at the next step; an underloaded expert receives a larger bias.

DeepSeek-V3 sets $\gamma = 0.001$ for the first 14.3T training tokens and $\gamma = 0$ for the final 500B tokens [2].

6.1.2 Empirical sigmoid–softmax comparison

DeepSeek-V3 uses the sigmoid affinity for both dispatch and mixture weighting, although these are logically distinct roles. Section 7.3 will separate their design requirements.

Wang et al. compare sigmoid- and softmax-gated 1B MoE models while varying the auxiliary-loss coefficient λ [10]. Each point in Figure 1 is one trained model. The horizontal axis is the global maximal load violation, so moving right means a more imbalanced expert assignment; the vertical axis is validation perplexity (the exponential of validation negative log-likelihood, so it induces the same performance ordering as that loss; lower is better). At comparable imbalance, their sigmoid runs have lower perplexity, and the softmax runs deteriorate more rapidly as imbalance increases. This experiment motivates sigmoid empirically, but it changes the activation in both roles at once and therefore does not identify which role causes the improvement.

6.1.3 A surrogate auxiliary-loss view of bias balancing

The bias update can be understood as matching the current expert-load distribution to an ideal uniform distribution. At training step t, expert j receives $\mathrm{ce}_j^{(t)}(\boldsymbol{b})$ assignments under the current

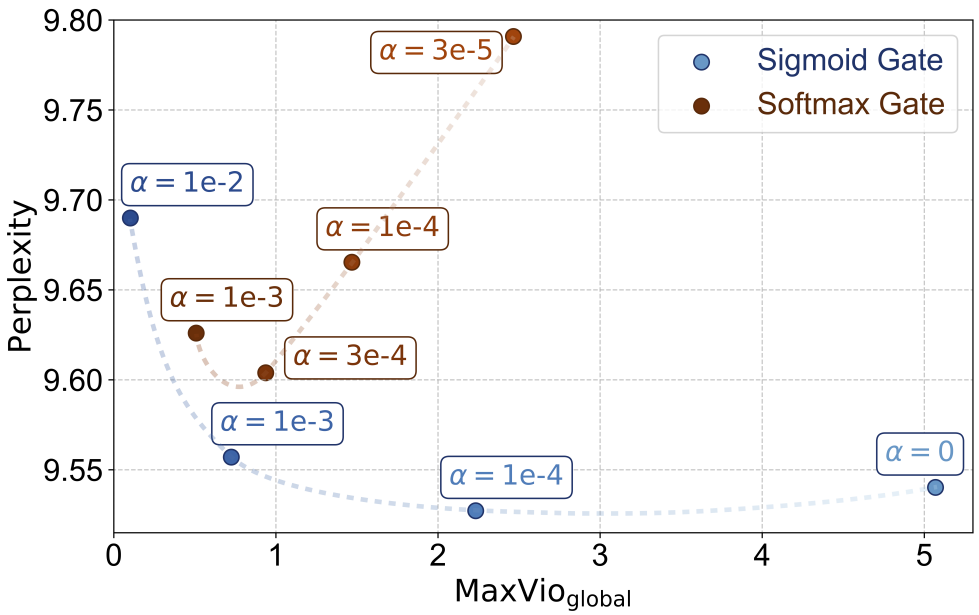

Figure 1: Sigmoid versus softmax on a 1B MoE model while varying the auxiliary-loss coefficient λ. Lower perplexity and lower $\text{MaxVio}_{\text{global}}$ are better. Reproduced from Figure 7 of Wang et al. [10].

bias $\boldsymbol{b}$. Because M tokens each choose K experts, $\sum_j \text{ce}_j^{(t)} = MK$. The current normalized load distribution is therefore

$$\pi_j^{(t)}(\boldsymbol{b}) = \frac{\text{ce}_j^{(t)}(\boldsymbol{b})}{MK}, \qquad \sum_{j=1}^{N} \pi_j^{(t)}(\boldsymbol{b}) = 1. \tag{37}$$

Perfect balance corresponds to the uniform target distribution

$$u_j = \frac{1}{N}, \qquad j = 1, \ldots, N. \tag{38}$$

The most direct balancing objective is then the MSE between the current and ideal distributions,

$$\mathcal{L}_{\text{MSE}}(\boldsymbol{b}) = \frac{1}{2} \sum_{j=1}^{N} \left(\pi_j^{(t)}(\boldsymbol{b}) - \frac{1}{N} \right)^2. \tag{39}$$

An overloaded expert has $\pi_j^{(t)} > 1/N$ and should have its bias reduced; an underloaded expert has $\pi_j^{(t)} < 1/N$ and should have its bias increased.

The hard Top-K counts $\text{ce}_j^{(t)}$ are piecewise constant in $\boldsymbol{b}$, so the ordinary derivative of this MSE cannot provide the desired update. Following the straight-through interpretation in Su [8], define a proxy load distribution

$$\widetilde{\boldsymbol{\pi}}^{(t)}(\boldsymbol{b}) = \boldsymbol{b} + \text{sg}\left[\boldsymbol{\pi}^{(t)}(\boldsymbol{b}) - \boldsymbol{b} \right], \tag{40}$$

where $\text{sg}[\cdot]$ is stop-gradient. In the forward pass, $\widetilde{\boldsymbol{\pi}}^{(t)} = \boldsymbol{\pi}^{(t)}$, so the proxy uses the actual load distribution. In the backward pass, $\partial \widetilde{\pi}_j^{(t)} / \partial b_k = \mathbb{1}[j = k]$. Substituting this proxy into the MSE gives

$$\mathcal{L}_{\text{proxy}}^{\text{DS}}(\boldsymbol{b}) = \frac{1}{2} \left\| \widetilde{\boldsymbol{\pi}}^{(t)}(\boldsymbol{b}) - \frac{1}{N} \mathbf{1} \right\|_2^2, \qquad \nabla_{\boldsymbol{b}} \mathcal{L}_{\text{proxy}}^{\text{DS}} = \boldsymbol{\pi}^{(t)}(\boldsymbol{b}) - \frac{1}{N} \mathbf{1}. \tag{41}$$

Applying SignSGD to this surrogate gradient yields

$$b_j^{(t+1)} = b_j^{(t)} - \gamma \, \text{sign}\left(\pi_j^{(t)} - \frac{1}{N} \right) = b_j^{(t)} + \gamma \, \text{sign}\left(\frac{MK}{N} - \text{ce}_j^{(t)} \right), \tag{42}$$

which is exactly the DeepSeek bias rule. Thus the rule can be read as one SignSGD step that reduces the MSE between the observed expert distribution and the uniform distribution. The loss is only a surrogate objective for updating $\boldsymbol{b}$: it is not added to $\mathcal{L}_{\text{task}}$, and no gradient from it is propagated into the router matrix W_r or the experts.

6.1.4 An RMS-normalized refinement of the bias update

Reading the DeepSeek rule as SignSGD on the surrogate MSE $\mathcal{L}^{\text{DS}}_{\text{proxy}}$ immediately suggests an improvement, proposed by Su [9]. Write the load-error vector as

$$\boldsymbol{\epsilon}^{(t)} = \nabla_{\boldsymbol{b}} \mathcal{L}^{\text{DS}}_{\text{proxy}} = \boldsymbol{\pi}^{(t)} - \tfrac{1}{N}\mathbf{1}, \qquad \epsilon_j^{(t)} = \pi_j^{(t)} - \tfrac{1}{N} \;\propto\; \text{ce}_j^{(t)} - \tfrac{MK}{N}. \tag{43}$$

DeepSeek applies $\text{sign}(\cdot)$ elementwise, whereas the refinement normalizes the whole vector by its root mean square,

$$b_j^{(t+1)} = b_j^{(t)} - \gamma \, \frac{\epsilon_j^{(t)}}{\text{RMS}(\boldsymbol{\epsilon}^{(t)})}, \qquad \text{RMS}(\boldsymbol{\epsilon}^{(t)}) = \sqrt{\frac{1}{N}\sum_{\ell=1}^{N} \big(\epsilon_\ell^{(t)}\big)^2}. \tag{44}$$

This is Normalized SGD in place of SignSGD, and it retains the sign of each component while restoring its *magnitude*.

Table 1 contrasts the three ways of turning the load error $\boldsymbol{\epsilon}^{(t)}$ into a bias step.

Table 1: Three ways to turn the load error $\boldsymbol{\epsilon}$ (with $\epsilon_j = \pi_j - \frac{1}{N} \propto \text{ce}_j - \frac{MK}{N}$) into a bias update direction Δb_j.

Scheme	Direction Δb_j	Kept	Discarded
No norm (GD)	$-\gamma\, \epsilon_j$	direction and absolute magnitude	nothing
Sign (SignSGD)	$-\gamma\, \text{sign}(\epsilon_j)$	direction only	magnitude and relative sizes
RMS norm	$-\gamma\, \dfrac{\epsilon_j}{\text{RMS}(\boldsymbol{\epsilon})}$	direction and relative magnitude	overall scale only

Why the same γ is a reasonable starting point. Both updates are scale-matched. For SignSGD every component of $\text{sign}(\boldsymbol{\epsilon}^{(t)})$ has unit magnitude, so $\text{RMS}(\text{sign}(\boldsymbol{\epsilon}^{(t)})) = 1$; for the normalized update $\text{RMS}(\boldsymbol{\epsilon}^{(t)}/\text{RMS}(\boldsymbol{\epsilon}^{(t)})) = 1$ by construction. The two step directions therefore have the same overall RMS scale. DeepSeek's γ is consequently a natural initialization for the normalized update, although different dynamics may still justify retuning.

Why retaining relative magnitude helps. The difference is entirely in how the fixed budget is distributed across experts. SignSGD discards magnitude: an expert that is only marginally imbalanced ($\epsilon_j^{(t)}$ near zero) and an expert that is severely imbalanced both move the bias by the same γ. This overshoots the nearly balanced experts, pushing them away from an equilibrium they had almost reached, and the resulting oscillation limits how large γ can safely be. The RMS-normalized update instead keeps the *relative* sizes of $\epsilon_j^{(t)}$: a severely overloaded expert receives a proportionally larger correction while a nearly balanced one receives a small one. The step is therefore adaptive rather than uniform and concentrates the correction where the imbalance is largest. Because RMS scaling uses one positive scalar, it never flips a component's sign and preserves the descent direction of the surrogate objective.

Why RMS, and not L_2 or no normalization. Table 1 places RMS between the two extremes: sign keeps only the sign pattern, plain GD keeps the absolute scale, and RMS keeps relative severity while fixing the overall scale.

- *No normalization* ($\Delta b_j = -\gamma\, \epsilon_j^{(t)}$) lets the step scale vary with the imbalance and with the normalization used to define $\boldsymbol{\epsilon}$. This can be useful near equilibrium, but it makes one fixed γ less transferable across training stages and expert counts.
- *L_2 normalization* ($\epsilon_j^{(t)}/\|\boldsymbol{\epsilon}^{(t)}\|_2$) is the same direction, since $\mathrm{RMS}(\boldsymbol{\epsilon}^{(t)}) = \|\boldsymbol{\epsilon}^{(t)}\|_2/\sqrt{N}$; the two differ only by the constant $\sqrt{N}$, absorbed into γ. RMS is preferred purely for scale matching: an L_2-unit step has norm 1 whereas the sign step has L_2 norm $\sqrt{N}$, so RMS puts both directions on the same unit-RMS scale without rescaling γ.

6.2 Kimi K3

6.2.1 Kimi K3 Quantile Balancing

Kimi K3 replaces the fixed-size sign update with Quantile Balancing (QB) [5]. We index a bias by when it becomes available: the current forward at training step t uses $\boldsymbol{b}^{(t-1)}$, and the same batch is then used to estimate $\boldsymbol{b}^{(t)}$ for the next step.

Current step t: forward routing with the old bias. For token i, the router first computes logits and sigmoid affinities,

$$Z_{i,:} = \boldsymbol{x}_i^\top W_r, \qquad G_{i,:} = \sigma(Z_{i,:}), \tag{45}$$

and adds the bias saved from the preceding step only to the dispatch scores,

$$D_{i,j}^{(t)} = G_{i,j} + b_j^{(t-1)}. \tag{46}$$

The current routes and mixture weights are

$$\mathcal{R}_i^{(t)} = \mathrm{TopK}\Big(D_{i,:}^{(t)}, K\Big), \tag{47}$$

$$Q_{i,j}^{(t)} = \frac{G_{i,j}}{\sum_{r\in\mathcal{R}_i^{(t)}} G_{i,r}}, \qquad j \in \mathcal{R}_i^{(t)}. \tag{48}$$

Thus the forward sequence is router logits $\to$ sigmoid $\to$ add $\boldsymbol{b}^{(t-1)} \to$ Top-K. The old bias changes which experts execute, but it is absent from $Q_{i,j}^{(t)}$ and is not optimized by backpropagation.

Estimating $\boldsymbol{b}^{(t)}$ for the next step. QB uses this step-t batch to estimate the next bias. If M tokens each select K out of N experts, the target number of assignments per expert is

$$q = \frac{MK}{N}. \tag{49}$$

Step 1: one cutoff $\alpha_i^{(t)}$ per token (row-wise). QB assumes $K < N$; when $K = N$, every expert is selected and no balancing decision remains. For each token, QB reads one additional entry from the biased dispatch scores. Write their descending order as

$$D_{i,(1)}^{(t)} \geq D_{i,(2)}^{(t)} \geq \cdots \geq D_{i,(N)}^{(t)}. \tag{50}$$

The first K entries give the actual routes, while

$$\alpha_i^{(t)} = D_{i,(K+1)}^{(t)} \tag{51}$$

is the best currently unselected score. Equivalently, one Top-$(K+1)$ operation returns both the K routes and this cutoff. Therefore $\boldsymbol{\alpha}^{(t)} \in \mathbb{R}^M$ contains *one threshold per token*; it is an intermediate batch statistic, not an expert bias or model parameter. The old bias $\boldsymbol{b}^{(t-1)}$ is already included in $\alpha_i^{(t)}$ through $D_{i,:}^{(t)}$.

Step 2: one quantile bias $\widehat{b}_j^{(t)}$ per expert (column-wise). Freeze the M token cutoffs and consider a candidate next bias for expert j. QB uses the provisional threshold test

$$G_{i,j} + \widehat{b}_j^{(t)} > \alpha_i^{(t)}. \tag{52}$$

This is a conditional calibration with the old cutoffs held fixed; it does not rerun an exact Top-K operation for every candidate bias. Its purpose is to turn M token-wise rankings into M scalar tests for the same expert. Define the margin and, equivalently, the required bias by

$$m_{i,j}^{(t)} = G_{i,j} - \alpha_i^{(t)}, \qquad r_{i,j}^{(t)} = \alpha_i^{(t)} - G_{i,j} = -m_{i,j}^{(t)}. \tag{53}$$

Here $r_{i,j}^{(t)}$ is the bias that would put expert j exactly on token i's cutoff. The candidate passes that cutoff when $r_{i,j}^{(t)} < \widehat{b}_j^{(t)}$.

This exposes the change of axis. Step 1 works across each row of the $M \times N$ score matrix and produces one $\alpha_i^{(t)}$. After subtracting that row's cutoff, Step 2 fixes a column j and collects

$$r_{:,j}^{(t)} = \left(r_{1,j}^{(t)}, \ldots, r_{M,j}^{(t)}\right), \tag{54}$$

one required-bias value from every token. A single candidate $\widehat{b}_j^{(t)}$ is shared by all M tests, giving the provisional load

$$\widehat{c}_j\left(\widehat{b}_j^{(t)}\right) = \sum_{i=1}^{M} \mathbb{1}\left[r_{i,j}^{(t)} < \widehat{b}_j^{(t)}\right]. \tag{55}$$

To set this count to q, choose a bias above the fraction $q/M = K/N$ of the required-bias values. Ignoring ties, this is the K/N-quantile of the expert-j column. Equivalently, its negative is the $(1 - K/N)$-quantile of the margins:

$$\widehat{b}_j^{(t)} = \operatorname{quantile}_{K/N}\left(r_{:,j}^{(t)}\right) = -\operatorname{quantile}_{1-K/N}\left(m_{:,j}^{(t)}\right), \tag{56}$$

$$b_j^{(t)} = \widehat{b}_j^{(t)} - \frac{1}{N}\sum_{r=1}^{N} \widehat{b}_r^{(t)}. \tag{57}$$

Thus Step 1 produces M token quantities $\alpha_i^{(t)}$, whereas Step 2 produces N expert quantities $\widehat{b}_j^{(t)}$. The second line mean-centers the expert biases. They are committed only after the current forward is complete: step $t+1$ uses $\boldsymbol{b}^{(t)}$, and the batch that estimated the bias is never rerouted with it.

6.2.2 Mean-centering and the gauge freedom of the bias

The route decision has a common-shift, or gauge, freedom: adding the same scalar to every bias moves every dispatch score equally and leaves the Top-K selection unchanged,

$$\operatorname{TopK}(G_{i,:} + \boldsymbol{b}, K) = \operatorname{TopK}(G_{i,:} + \boldsymbol{b} + c\mathbf{1}, K) \quad \text{for any scalar } c. \tag{58}$$

Only the *relative* biases affect dispatch, so the common offset is a free gauge that neither method needs to fix for correctness. The two methods differ only in whether they leave that gauge to drift.

DeepSeek's sign update is incremental, $\boldsymbol{b}^{(t+1)} = \boldsymbol{b}^{(t)} + \gamma\,\mathrm{sign}(\cdot)$, and commonly omits centering. Its signed increment is not guaranteed to have zero mean, so the common offset can drift, although that drift does not change routes in exact arithmetic. Subtracting the mean after any update is therefore a valid numerical safeguard. For the RMS-normalized refinement, the unnormalized error vector sums to zero; dividing it by one scalar preserves that property, so its increment is already mean zero apart from numerical error.

QB, by contrast, recomputes an *absolute* bias vector from scratch each step, since the quantile calculation returns absolute required biases rather than increments. Nothing pins their common level, so without centering the offset could drift arbitrarily across steps and, in the histogram implementation below, push the required biases outside the tracked range. Mean centering therefore chooses a canonical representative of the equivalent bias vectors, keeps the offset from drifting, and keeps the histogram range tied to relative corrections. In both methods centering is a numerical and state-management convention, not an additional balancing force. It is more important for QB because QB produces absolute values, whereas DeepSeek only accumulates bounded increments.

6.2.3 Histogram: estimating the expert-axis quantile at scale

The exact update requires the M margins of every expert, or $O(MN)$ values, which cannot be gathered at each training step. Recall that the required bias $r_{i,j}^{(t)} = \alpha_i^{(t)} - G_{i,j}$ places expert j on token i's cutoff. Therefore, expert j should receive the (K/N)-quantile of $r_{:,j}^{(t)}$. Precisely q token–expert pairs require no more than this bias. This reversed quantile is equivalent to the negative upper-margin quantile above.

Kimi K3 uses the fact that its router score is sigmoid-bounded, $G_{i,j} \in (0, 1)$. If $b_{\min}$ and $b_{\max}$ are the minimum and maximum current biases, then every required bias lies in

$$r_{i,j}^{(t)} \in [b_{\min} - 1,\, b_{\max} + 1]. \tag{59}$$

Partition this interval into B equal-width bins with

$$w = \frac{b_{\max} - b_{\min} + 2}{B}. \tag{60}$$

For each expert j, each rank scatter-adds its local required biases into a histogram $H_{j,u}$, where $u \in \{0, \ldots, B-1\}$ is the bin index. The local histograms accumulate across gradient-accumulation micro-batches without communication. At the end of the training step, one all-reduce computes

$$H_{j,u}^{\text{global}} = \sum_{\text{ranks } a} H_{j,u}^{(a)}. \tag{61}$$

Thus, each expert's histogram represents the pooled global batch, not an average of independently estimated per-rank quantiles. This works because histogram counts are additive:

$$H_{j,u}^{\text{global}} = \sum_{a} \sum_{i \in \text{shard } a} \mathbb{1}\left[r_{i,j}^{(t)} \in \text{bin } u\right] = \sum_{a} H_{j,u}^{(a)}. \tag{62}$$

With shared bin edges, the all-reduce therefore reconstructs exactly the same *binned histogram* that one device would build from all MN token–expert required-bias values. The distributed merge is exact; replacing the raw values by bins is the approximation. Communication is $O(NB)$ integer counts per layer and step, rather than an all-gather of $O(MN)$ raw values.

For each expert j, let u_j be the first bin with cumulative count at least $\lceil q \rceil$. If c_j counts all entries in preceding bins and h_j is the count inside bin u_j, linear interpolation gives

$$\widehat{b}_j^{(t)} = b_{\min} - 1 + \left(u_j + \frac{q - c_j}{h_j} \right) w. \tag{63}$$

In practice, the fraction is clipped to $[0, 1]$ for boundary cases. The raw quantile and its histogram estimate lie in the same bin, so their difference is at most one bin width w.

7 Popular Improvements and Discussion

7.1 Shared experts: purpose, construction, and branch scaling

DeepSeekMoE combines always-active shared experts with finer-grained routed experts: the shared path captures common knowledge, while the larger routed pool is intended to support more specialized combinations [1]. A routed expert is updated only by the tokens assigned to it. Consequently, knowledge useful to almost every token is either duplicated across many routed experts or learned slowly by experts that see only a fraction of the data. A shared expert provides an always-active path for such common transformations. It also gives every token a useful fallback when routing is immature or a routed expert is capacity limited, leaving the routed pool free to model the conditional residual. The tradeoff is that shared computation is dense: too much shared width reduces the sparsity benefit and can dominate the routed branch.

We represent the shared path by S width-H SwiGLU slices. They may be implemented as S separate experts or fused into one SwiGLU of intermediate width SH; in the absence of slice-specific operations these are algebraically equivalent. For token i, define

$$\boldsymbol{s}_i = \sum_{s=1}^{S} \boldsymbol{e}_{i,s}^{\mathrm{sh}}, \qquad \boldsymbol{r}_i = \sum_{j \in \mathcal{R}_i} Q_{i,j} \boldsymbol{e}_{i,j}, \qquad \boldsymbol{o}_i = \boldsymbol{s}_i + \alpha \boldsymbol{r}_i. \tag{64}$$

For the initialization analysis below, we use the common normalized-gate construction, $\sum_{j \in \mathcal{R}_i} Q_{i,j} = 1$. The relevant design objective is not to reproduce the variance of a hypothetical dense FFN containing all N routed experts: only K of them execute for this token. Instead, we want the two branches to enter the residual stream with comparable L2 norm,

$$\mathbb{E}_i \|\alpha \boldsymbol{r}_i\|_2 = \mathbb{E}_i \|\boldsymbol{s}_i\|_2. \tag{65}$$

The exact population-level scale for this objective is therefore

$$\boxed{\alpha_{\mathrm{L2}} = \frac{\mathbb{E}_i \|\boldsymbol{s}_i\|_2}{\mathbb{E}_i \|\boldsymbol{r}_i\|_2}}. \tag{66}$$

This statement does not require zero coordinate means, isotropy, or independence between output coordinates. It directly matches the quantity that matters at the branch addition. In practice, an RMS-L2 estimator,

$$\alpha_{\text{RMS-L2}} = \sqrt{\frac{\mathbb{E}_i \|\boldsymbol{s}_i\|_2^2}{\mathbb{E}_i \|\boldsymbol{r}_i\|_2^2}}, \tag{67}$$

is easier to estimate from batches. The two scales are close when norms concentrate, as they typically do in a high-dimensional residual stream.

For an initialization rule, suppose only that different width-H expert outputs are approximately uncorrelated and have the same expected squared norm $\mathbb{E}\|\boldsymbol{e}\|_2^2 = \eta H$. Define the gate

concentration

$$\rho = \mathbb{E}_i\left[\sum_{j\in\mathcal{R}_i} Q_{i,j}^2\right], \qquad \frac{1}{K} \le \rho \le 1. \tag{68}$$

Then

$$\mathbb{E}\|\boldsymbol{s}_i\|_2^2 \approx S\eta H, \qquad \mathbb{E}\|\boldsymbol{r}_i\|_2^2 \approx \rho\eta H, \tag{69}$$

and Eq. (67) becomes

$$\boxed{\alpha_{\text{init}} \approx \sqrt{\frac{S}{\rho}}}. \tag{70}$$

With nearly uniform selected gates, $Q_{i,j} \approx 1/K$, this reduces to $\alpha_{\text{init}} \approx \sqrt{SK}$. Crucially, the scale contains the number of active routed experts K through their gate weights, but not the total pool size N. Increasing the number of dormant choices does not increase the norm of the realized routed output.

The approximation can fail after specialization because expert outputs may be correlated and may have unequal norms. Let $\boldsymbol{q}_i = (Q_{i,j})_{j\in\mathcal{R}_i}$. In full generality, $\|\boldsymbol{r}_i\|_2^2 = \boldsymbol{q}_i^\top \Gamma_i \boldsymbol{q}_i$, where $(\Gamma_i)_{ab} = \boldsymbol{e}_{i,a}^\top \boldsymbol{e}_{i,b}$ is the Gram matrix of the selected expert outputs. Thus, a robust implementation can initialize with Eq. (70) and monitor exponential moving averages of the two branch energies. A learned scalar gain can adapt slowly, but per-token norm matching is undesirable: it couples the routed branch directly to the shared branch and removes potentially useful confidence information carried by its magnitude.

7.2 Increasing the total number of experts versus making experts finer-grained

"More experts" can describe two different interventions. The first increases the *total number of full experts*: each expert keeps its original width, so the model acquires more parameters and more conditional capacity. The second holds the total parameter budget approximately fixed but divides each coarse expert into several narrower routing units, usually allowing a token to select more of them. The first asks whether the model needs more capacity; the second asks how a fixed amount of capacity should be organized.

Increasing the total expert number is model scaling. Adding full-width experts scales the total parameter count of the MoE. It can provide new high-dimensional transformations for different domains, languages, modalities, or reasoning patterns, while preserving the ability of features within one expert to interact. This is advantageous when the existing experts are already coherent but there are too few of them to cover the recurring token populations.

The extra capacity is useful only if the training data scale with it. Each expert learns from the subset of tokens routed to it; as the total expert number grows, learning distinct and reliable experts requires more tokens and usually more diverse data. On a fixed dataset, there is an effective upper limit to the model capacity that can be supported: beyond it, new experts tend to remain undertrained, duplicate existing experts, or memorize narrow patterns instead of improving validation loss.

This is the assumption hidden by a one-variable model-size scaling law such as $L(N_{\text{params}})$. It should be interpreted as the model-size slice of a joint law $L(N_{\text{params}}, D_{\text{data}})$ in a regime where the dataset size D_{data} is large enough not to be the bottleneck—conceptually, the infinite-data limit. It does not imply that increasing parameters indefinitely will improve loss on a fixed D_{data} [4]. Increasing the total number of experts should therefore be paired with data scaling; otherwise the experiment mostly measures the finite-data ceiling rather than the value of additional experts.

Fine-grained splitting under a fixed total parameter budget. Fine-grained experts start from the opposite control condition. Suppose each coarse expert is divided into m narrower subexperts and the number selected per token is increased correspondingly. The model has not been given more total capacity; it has exchanged a small number of powerful, complete experts for a larger number of weaker, more precise specialists.

This weakens the standalone ability of each expert because it has fewer features with which to represent a complete transformation. At the same time, it can sharpen expert responsibility. A narrow expert can focus on a more homogeneous behavior—for example, a language feature, a syntactic operation, or a domain-specific pattern—instead of modeling every feature needed by a broad token cluster. A token then composes several such specialists, producing a combination that need not be stored in any single coarse expert. Fine-grained routing therefore behaves like a factorized expert space: it trades individual expert strength for finer specialization and more reusable combinations.

This trade is beneficial only when useful transformations are decomposable. If their intermediate features require strong within-expert interactions, a narrow subexpert may be too weak to learn the desired capability. Excessive splitting can also fragment the learning signal among nearly interchangeable units, producing shallow or redundant specialization. Fine-grained routing should therefore make expert responsibility more precise, not merely increase the raw number of expert IDs.

The central comparison is not simply "few experts versus many experts." Increasing the total number of full experts expands model capacity and is preferable when additional data are available and the existing experts are capacity-limited. Splitting experts keeps total capacity controlled and is preferable when coarse experts contain several reusable behaviors that tokens should combine differently. DeepSeekMoE uses this second strategy to increase the flexibility of expert combinations [1]; LatentMoE and Kimi K3 push it toward routing transformations in a shared latent space [5].

7.3 Route and gate activations: separate roles and requirements

A router produces logits $Z_{i,:}$, but two downstream decisions follow. The *route activation* determines which experts execute, whereas the *gate activation* determines how their outputs are combined. In general they can be written with two different functions. The routing rule is

$$\mathcal{R}_i = \mathrm{TopK}(\phi_{\mathrm{route}}(Z_{i,:}) + \boldsymbol{b}, K)\,. \tag{71}$$

For auxiliary-loss routing there is no dispatch bias, so this equation should be read with $\boldsymbol{b} = \mathbf{0}$. The auxiliary loss changes the logits through backpropagation instead. In auxiliary-loss-free routing, $\boldsymbol{b}$ is the explicit balancing state added only for Top-K selection. For each selected expert, the gate activation first produces an unnormalized weight

$$\widetilde{Q}_{i,j} = \phi_{\mathrm{gate}}(Z_{i,j}), \qquad j \in \mathcal{R}_i. \tag{72}$$

This definition does not require softmax or renormalization. In practice, ϕ_{gate} usually produces nonnegative scores, and many MoE implementations renormalize them over the selected experts:

$$Q_{i,j} = \frac{\widetilde{Q}_{i,j}}{\sum_{r\in\mathcal{R}_i} \widetilde{Q}_{i,r}}, \qquad j \in \mathcal{R}_i. \tag{73}$$

With $\phi_{\mathrm{gate}}(z) = \exp(z)$ this is a selected-expert softmax; with $\phi_{\mathrm{gate}}(z) = \sigma(z)$ it is normalized sigmoid, as used by DeepSeek-V3. Renormalization is an architectural choice rather than part of the definition of a gate activation. Without it, one simply sets $Q_{i,j} = \widetilde{Q}_{i,j}$, and the routed-output

magnitude also depends on the total selected gate mass. The discussion below uses $Q_{i,j}$ for the weights actually applied to the expert outputs, whether or not they were renormalized.

There is no mathematical requirement that $\phi_{\text{route}} = \phi_{\text{gate}}$. Using the same function is a convenient architectural choice: it reuses one score and tends to align the experts preferred for execution with those given the largest output weight.

For auxiliary-loss-free routing, should the route activation be sharp or smooth? Without a balancing bias, the route side is only a ranking mechanism: any strictly increasing expert-wise function gives the same Top-K set as the raw logits. In particular,

$$\text{TopK}(Z_{i,:}, K) = \text{TopK}(\sigma(Z_{i,:}), K) = \text{TopK}(\text{softmax}(Z_{i,:}), K). \tag{74}$$

Thus, for the ordinary auxiliary-loss method with $\boldsymbol{b} = \mathbf{0}$, making a monotone route activation sharper or smoother does not change the selected experts. The situation changes in auxiliary-loss-free routing because the activated score and the bias compete inside Top-K. Expert j outranks expert k precisely when

$$\phi_{\text{route}}(Z_{i,j}) - \phi_{\text{route}}(Z_{i,k}) > b_k - b_j. \tag{75}$$

A sharper route activation enlarges the score differences relative to the bias. It preserves the router's task-driven preference and makes balancing less likely to redirect a token, but the controller then needs a larger correction to move load away from an overloaded expert. A smoother activation compresses the score differences, so a modest bias can balance load effectively; if it is too smooth, however, dispatch is dominated by the expert-level bias, tokens near a boundary switch routes frequently, and semantic routing is weakened. The desired route activation is therefore not maximally sharp or maximally smooth. Its score scale should leave the bias enough authority to resolve system-level overload without routinely overturning large token–expert affinity gaps.

Sigmoid is useful here because it computes every expert score independently and bounds it in $(0, 1)$, giving the bias a consistent unit and histogram QB a finite range. Softmax introduces a token-dependent denominator: changing an unrelated third logit changes the score margin between experts j and k and therefore changes the effective strength of the same $b_j - b_k$. This is why Wang et al. use an error-proportional rather than fixed-step bias update in their softmax experiment [10]. QB adapts its update to the observed score distribution and therefore weakens this particular advantage of sigmoid, but it still benefits from a bounded, stable score scale.

For both balancing families, should the gate activation be sharp or smooth? Unlike route sharpness, gate sharpness matters with or without an auxiliary loss. The selected gate weights directly determine the mixture output and the task gradient received by each selected expert. For selected experts j and k, softmax and normalized sigmoid give

$$\frac{Q^{\text{sm}}_{i,j}}{Q^{\text{sm}}_{i,k}} = \exp(Z_{i,j} - Z_{i,k}), \qquad \frac{Q^{\text{sig}}_{i,j}}{Q^{\text{sig}}_{i,k}} = \frac{\sigma(Z_{i,j})}{\sigma(Z_{i,k})}. \tag{76}$$

Softmax can express a sharper preference; sigmoid compresses large positive gaps and can keep the nominal Top-K from degenerating into an effective Top-1. A useful diagnostic is

$$K_{\text{eff},i} = \frac{\left(\sum_{j \in \mathcal{R}_i} Q_{i,j}\right)^2}{\sum_{j \in \mathcal{R}_i} Q_{i,j}^2}, \qquad 1 \le K_{\text{eff},i} \le K. \tag{77}$$

This scale-invariant form applies to both normalized and unnormalized nonnegative gates; when the selected weights sum to one, it reduces to $K_{\text{eff},i} = 1/\sum_j Q_{i,j}^2$. Values near 1 mean that a

sharp gate has reduced the nominal Top-K mixture to an effective Top-1: one expert carries nearly all of the output and task gradient, while the other selected experts contribute little. This can express high router confidence and strong specialization, but an overly sharp gate wastes the additional selected paths and starves them of learning signal. A smoother gate raises $K_{\text{eff},i}$, shares gradients among the selected experts, and makes genuine multi-expert composition possible. If it is too smooth, it averages away useful confidence differences and forces weakly matched experts to influence the output almost as much as the best match.

The appropriate gate is therefore moderately selective: sharp enough to represent meaningful token–expert affinity, but smooth enough that the chosen K experts actually participate. This criterion is the same for auxiliary-loss and auxiliary-loss-free routing because neither balancing mechanism removes the gate's role in the task computation. Nguyen et al. show improved expert-estimation sample efficiency for sigmoid gating in an idealized over-specified MoE regression model [6]; this supports, but does not prove for Top-K LLMs, a benefit from weaker gate competition.

7.4 Normalization of the routed branch

The right place for an RMSNorm in the routed branch is *after routed aggregation*. Normalizing $\boldsymbol{r}_i = \sum_j Q_{i,j}\boldsymbol{e}_{i,j}$ directly targets route-dependent scale changes caused by K, gate concentration, and expert-norm differences. This is the most defensible location when such variation is empirically unstable. In a latent architecture it should be placed after aggregation and before the shared up projection,

$$\boldsymbol{u}_i = \sum_{j\in\mathcal{R}_i} Q_{i,j}E_j^{\text{lat}}(W_{\downarrow}\boldsymbol{x}_i), \qquad \boldsymbol{r}_i^{\text{out}} = W_{\uparrow}\,\text{RMSNorm}(\boldsymbol{u}_i). \tag{78}$$

This stabilizes the input seen by $W_{\uparrow}$ without constraining each latent expert separately. When only the routed aggregate is normalized for stability, its gain or following projection must still be initialized so that $\mathbb{E}\|\boldsymbol{r}_i^{\text{out}}\|_2^2 \approx \mathbb{E}\|\boldsymbol{s}_i\|_2^2$; RMSNorm fixes its own input RMS, not the L2 norm after $W_{\uparrow}$ and not the balance between the two branches.

7.5 Why the first few layers are made dense

Recent large MoE models, including DeepSeek-V3 [2] and Kimi K3 [5], replace the routed MoE blocks in the first few Transformer layers with ordinary dense FFNs, and only switch to sparse routing in the deeper layers. This is a deliberate design choice rather than an implementation artifact, and it has several reinforcing motivations.

- **Early representations are not yet specialized.** The first layers operate close to token and positional embeddings, where the useful computation is generic feature extraction shared by essentially all tokens. There is little conditional structure for experts to specialize on, so a dense FFN, which every token uses in full, matches what these layers actually need, whereas routing would only fragment a common transformation across experts that each see a fraction of the tokens.
- **Routing is unreliable on immature features.** The router decides dispatch from the current hidden state. In the earliest layers those states are noisy and weakly clustered, so Top-K decisions are unstable and load tends to collapse onto a few experts. Making these layers dense removes the hardest routing decisions from the point in the network where they are least informative, and lets balancing act only where token representations already carry meaningful structure.

- **Stability and gradient quality.** Dense early layers give every token the same well-conditioned path near the input, avoiding the high-variance, sparsely-updated expert gradients that early routing would produce. Empirically this improves training stability, which matters most in the layers closest to the loss-bearing embeddings.
- **Favorable compute trade-off.** The MoE parameter budget is most useful where specialization exists, i.e. in the deeper layers. Because the early layers are few, keeping them dense costs little total FLOPs while removing routing and its balancing overhead there, so the sparsity budget is spent where it yields the largest capacity gain.

In short, dense early layers concentrate generic, all-token computation where specialization is absent and routing is least trustworthy, and reserve sparse experts for the deeper layers where conditional structure and stable routing make the extra capacity pay off.

8 Summary: Balancing Efficiency without Erasing Specialization

Load balancing is not the modeling objective of an MoE; it is a systems and optimization constraint that prevents routing collapse, device stragglers, and poorly trained cold experts. The desired endpoint is therefore not the most uniform routing possible. It is the weakest intervention that keeps sparse training efficient while preserving the task-driven differences among tokens and experts.

What the auxiliary loss gets right. An auxiliary loss is simple, differentiable, and stateless. It reuses the router scores already computed in the forward pass, changes the router itself rather than maintaining a separate inference-time controller, and provides a continuous correction before an expert becomes completely cold. Its coefficient λ also gives a direct knob for trading task fit against balance. These properties make it a strong baseline, especially when the expert count is moderate and exact per-step balance is unnecessary.

Its main weakness is that the balance signal is backpropagated through the same router parameters as the task loss. The extra gradient can oppose semantic routing, and λ is sensitive to model scale, batch composition, and the chosen balance statistic. A strong loss can make unrelated domains use similar expert proportions; a weak loss may not prevent a systems bottleneck. Moreover, the usual softmax form couples every expert score and does not directly control the realized per-device workload.

DeepSeek and QB solve different control problems. Both methods remove the balancing gradient and add a bias only to dispatch, so the unbiased gate weights and task gradient remain untouched. DeepSeek is an incremental feedback controller: it observes only whether each expert's count is above or below the target and moves its bias by a fixed signed step γ. Its state and communication are small, approximately one count per expert, but convergence is gradual and γ trades slow adaptation against oscillation.

QB is a score-aware calibration rule. It first obtains one cutoff per token, then uses the entire per-expert distribution of required biases to estimate the bias that would hit the target load under frozen cutoffs. It therefore adapts the step size to the current score geometry and can approach the conditional balance target in one update. The price is more state and communication: a practical implementation all-reduces $O(NB)$ histogram counts per layer for N experts and B bins, rather than the $O(N)$ expert counts sufficient for the DeepSeek update. The histogram and the simultaneous change of all cutoffs also mean that QB is a calibrated approximation, not a proof of exact balance on the next batch.

Why absolute balance can be harmful. Uniform aggregate counts are useful for hardware, but they are not evidence of good specialization. Different domains, modalities, and token types need not occur equally often or require the same experts. Enforcing equality on every sequence or small batch can send tokens to low-affinity experts, dilute useful domain specialization, increase route churn, and create a train–inference mismatch under distribution shift. Count balance also does not imply equal gate mass, equal gradient, or equal expert quality. The system should therefore constrain overload, not erase all nonuniformity.

References

[1] Damai Dai, Chengqi Deng, Chenggang Zhao, Runxin Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL: https://arxiv.org/abs/2401.06066.

[2] DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. URL: https://arxiv.org/abs/2412.19437.

[3] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL: https://arxiv.org/abs/2101.03961.

[4] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. URL: https://arxiv.org/abs/2001.08361, doi:10.48550/arXiv.2001.08361.

[5] Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026. URL: https://arxiv.org/abs/2607.24653.

[6] Huy Nguyen, Nhat Ho, and Alessandro Rinaldo. Sigmoid gating is more sample efficient than softmax gating in mixture of experts. In *Advances in Neural Information Processing Systems*, 2024. URL: https://arxiv.org/abs/2405.13997.

[7] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL: https://arxiv.org/abs/1701.06538.

[8] Jianlin Su. MoE tour 3: A different way to assign. Scientific Spaces, 2025. URL: https://kexue.fm/archives/10757.

[9] Jianlin Su. MoE tour 4: Balancing without an auxiliary loss. Scientific Spaces, 2025. URL: https://kexue.fm/archives/10815.

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