A Journey Through MoE: 8. Enforcing Sequence-Level Balance

Su Jianlin

2026-05-22

To date, the "A Journey Through MoE" series has published seven articles, five of which have focused on routing and load balancing in Mixture-of-Experts (MoE) models. In terms of routing form, these can be categorized into static and dynamic computation; in terms of load balancing methods, they fall into two classes: Aux-Loss and Loss-Free. The latter further includes the SignSGD approach proposed by DeepSeek and the Quantile Balancing (QB) method proposed by the author.

Another detail is that all Loss-Free schemes discussed so far aim to achieve load balancing at the global batch level. Often, an auxiliary loss (Aux Loss) at the sequence level is added to prevent extreme imbalance within individual sequences. If we assume that every Aux-Loss method has a corresponding Loss-Free variant, how should sequence-level Loss-Free balancing be implemented? This is the topic explored in this article.

1 Review of Previous Work

First, let us briefly review previous discussions on load balancing. In the article "A Journey Through MoE: 2. It's Not Scarcity But Imbalance That Matters", the load balancing problem in MoE was first discussed, with the proposed solution being an auxiliary loss (Aux Loss)an additional loss function designed to promote balance. The drawback of Aux Loss is the need to tune an extra hyperparameter, while its advantage lies in flexible control over the granularity of balancing, enabling sequence-level, group-level, or global-level balance (though the more global the balance, the higher the communication cost).

Starting from "A Journey Through MoE: 3. A Different Approach to Allocation", we began discussing Loss-Free load balancing strategies. These methods introduce an additional bias term to regulate the degree of balance and use manually designed SignSGD-style rules to update the bias term. This approach is more flexible than Aux Loss and more infrastructure-friendly. In "A Journey Through MoE: 4. Invest More Where It's Difficult", this idea was extended to MoE with dynamically adjustable expert activation counts.

Later, in "A Journey Through MoE: 6. Optimal Allocation for Balance", the load balancing problem was analyzed from the perspective of optimal allocation. By alternately solving the dual objective, a new load balancing strategyQuantile Balancing (QB)was derived. In effect, QB can be seen as a more accurate and novel method for solving the bias term in Loss-Free schemes. Subsequently, in "A Journey Through MoE: 7. Minimalist Solution for Dynamic Activation", QB was also generalized to dynamic MoE.

One important detail: the bias term introduced in Loss-Free methods is globally shared. Therefore, all existing Loss-Free methods can only achieve global load balancing. As mentioned earlier, Aux Loss can be used to promote sequence-level balance. Is there a corresponding Loss-Free method for achieving sequence-level balance?

2 Local Centering

Coincidentally, @ChangJonathanC explored a related idea in the blog post "Causal Routing Bias for Aux-Loss-Free MoE Training". The author proposed two strategies; we briefly review them here to contrast with our own approach later. Let $\boldsymbol{s} \in \mathbb{R}^{l\times n}$ denote the router outputs for a sequence of length l, and $\boldsymbol{s}_i \in \mathbb{R}^n$ denote the router output for the i-th token. The first proposed strategy subtracts a sliding average (EMA) from $\boldsymbol{s}$:

$$\hat{\boldsymbol{s}}_i = \boldsymbol{s}_i - \bar{\boldsymbol{s}}_i, \qquad \bar{\boldsymbol{s}}_i = \gamma \bar{\boldsymbol{s}}_{i-1} + (1-\gamma)\boldsymbol{s}_i \tag{1}$$

Then $\hat{\boldsymbol{s}}$ is used to determine which experts to activate (e.g., selecting Top-k, possibly combined with QB). The underlying idea is straightforward: if an expert j is frequently activated when using $\boldsymbol{s}$ for routing, its scores $\boldsymbol{s}_{:,j}$ are on average higher, thus it should be penalized accordingly, with the penalty determined by the EMA-estimated local average.

Thus, in the new $\hat{\boldsymbol{s}}$, each expert's scores approximately have zero mean locally, preventing any single expert from being overly dominant, thereby achieving sequence-level balance. However, this is purely heuristic and lacks theoretical guarantees of improved balance. The author conducted simple tests and found that it provided no significant improvement in heavily imbalanced scenarios (e.g., the first MoE layer). Perhaps recognizing this limitation, the author proposed a second approach.

3 Test-Time Training

The second approach is also quite intuitive. In "A Journey Through MoE: 3. A Different Approach to Allocation", a bias term was introduced to control global balance and updated via SignSGD. By analogy with the "Test-Time Training" (TTT) idea introduced in "Test-Time Training", one can activate experts token-by-token along the sequence dimension and update the bias term based on the activation distribution so far.

Taking $\boldsymbol{s} \in \mathbb{R}^{l\times n}$ and Top-k MoE as an example, the original Loss-Free method introduces a global bias vector $\boldsymbol{\beta} \in \mathbb{R}^n$ and activates the top-k experts based on $\boldsymbol{s}_i - \boldsymbol{\beta}$. The new idea is to assign a separate $\boldsymbol{\beta}_i \in \mathbb{R}^n$ for each $\boldsymbol{s}_i$, so that the activation rule becomes selecting top-k from $\boldsymbol{s}_i - \boldsymbol{\beta}_i$. The update rule for $\boldsymbol{\beta}_i$ can be sign-gradient ascent:

$$\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i-1} + \eta\, \mathrm{sign}(\boldsymbol{f}_{\leq i-1} - 1/n) \tag{2}$$

where $\boldsymbol{f}_{\leq i-1}$ is the distribution of activated experts up to token $i-1$. The author's final version modifies this by removing the sign function and replacing $\boldsymbol{f}_{\leq i-1}$ with $\boldsymbol{f}_{i-1}$ (the activation distribution at token $i-1$) to better achieve local balance:

$$\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i-1} + \eta(\boldsymbol{f}_{i-1} - 1/n) \tag{3}$$

This method preserves causal dependency and can ensure nearly perfect sequence-level balance. However, since Top-k selection and bias updates must be performed step-by-step, it essentially introduces a non-linear RNN. Non-linear RNNs are non-parallelizable, so long sequences become a bottleneck. One could consider chunk-wise updating (Mini-batch TTT) to improve efficiency, but this comes at the cost of elegance.

4 The Optimal Solution

Next, we introduce the method proposed in this article, named "Moving Quantile Balancing (MQB)". As the name suggests, it evolved from "Quantile Balancing (QB)". We first briefly recap QB.

Although the author wrote "A Journey Through MoE: 6. Optimal Allocation for Balance" before "A Journey Through MoE: 7. Minimalist Solution for Dynamic Activation",

the latter is conceptually and methodologically simpler, so we start with it. Given router scores $\boldsymbol{s} \in \mathbb{R}^{l \times n}$, we introduce a bias vector $\boldsymbol{\beta} \in \mathbb{R}^n$. In static MoE, each token activates the k experts with the highest values in $\boldsymbol{s}_i - \boldsymbol{\beta}$. In dynamic MoE, all experts with $\boldsymbol{s}_i - \boldsymbol{\beta} > 0$ are activated (variable number).

To simultaneously maintain load balance and control the average number of activated experts to k, an appropriate $\boldsymbol{\beta}$ must be chosen. Remarkably, in this setting, the optimal $\boldsymbol{\beta}$ has an exact closed-form expression:

$$\boldsymbol{\beta} = \text{desc\_sort}(\boldsymbol{s}, \text{axis=0})_{[lk/n:lk/n+1]} = \text{quantile}(\boldsymbol{s}, 1 - k/n, \text{axis=0}) \tag{4}$$

That is, sorting $\boldsymbol{s}$ in descending order along the sequence dimension (axis=0) and taking the (lk/n)-th largest element gives the optimal $\boldsymbol{\beta}$this is equivalent to the $(1-k/n)$-quantile.

Actually, this is merely an alternative representation of Expert Choice. We know that Expert Choice violates causality (non-causal), which can be seen from computing $\boldsymbol{\beta}$ across the sequence dimension. While this is acceptable in encoder scenarios, it is not desirable for causal autoregressive models.

5 Sliding Quantiles

The "Test-Time Training" approach by @ChangJonathanC provided inspiration. We consider computing the bias for each token incrementally along the sequence using QB. Since QB directly computes the optimal bias from $\boldsymbol{s}$ without requiring knowledge of actual expert activations, some degree of parallelism becomes possible.

The author's initial idea was $\boldsymbol{\beta}_i = \text{quantile}(\boldsymbol{s}_{[:i]}, 1 - k/n, \text{axis=0})$: computing the bias using all tokens up to position i, resulting in a position-specific bias and guaranteed balance up to that point. Because this requires sequentially computing quantiles along the sequence, we call this operation "Cumulative Quantile", and the method "Cumulative Quantile Balancing (CQB)".

CQB is theoretically feasible, but since quantiles cannot be incrementally updated, each step requires a full pass over the data, leading to linearly increasing per-step complexity and quadratic total complexity.

To mitigate this and enhance local balance, the idea is to define a window w, and compute the optimal bias for each token only within a window of size w:

$$\boldsymbol{\beta}_i = \text{quantile}(\boldsymbol{s}_{[i-w:i]}, 1 - k/n, \text{axis=0}) \tag{5}$$

This transforms the operation from "Cumulative Quantile" to "Moving Quantile", yielding a prototype of "Moving Quantile Balancing (MQB)". It resembles Sliding Window Attention (SWA) with linear complexity and potential for parallelism. The reason it's still a "prototype" is that quantile computation remains non-incremental, making per-step cost relatively highrequiring further optimization.

6 Bucketing Estimation

The obstacles mentioned above stem fundamentally from the fact that "quantiles are nonlinear and non-incremental". Can we linearize thisapproximately, if not exactly? Fortunately, we can!

A key insight is: given the distribution of a variable, its quantile can be computed via cumulative probability, and distributions can be updated incrementally! Therefore, the key is to reformulate quantile estimation as distribution estimation. We achieve this using bucketed countingalso known as "histogram approximation". First, assume router scores $\boldsymbol{s}$ lie within $[0, 1]$, which can be ensured via Sigmoid or similar activation functions. Then,

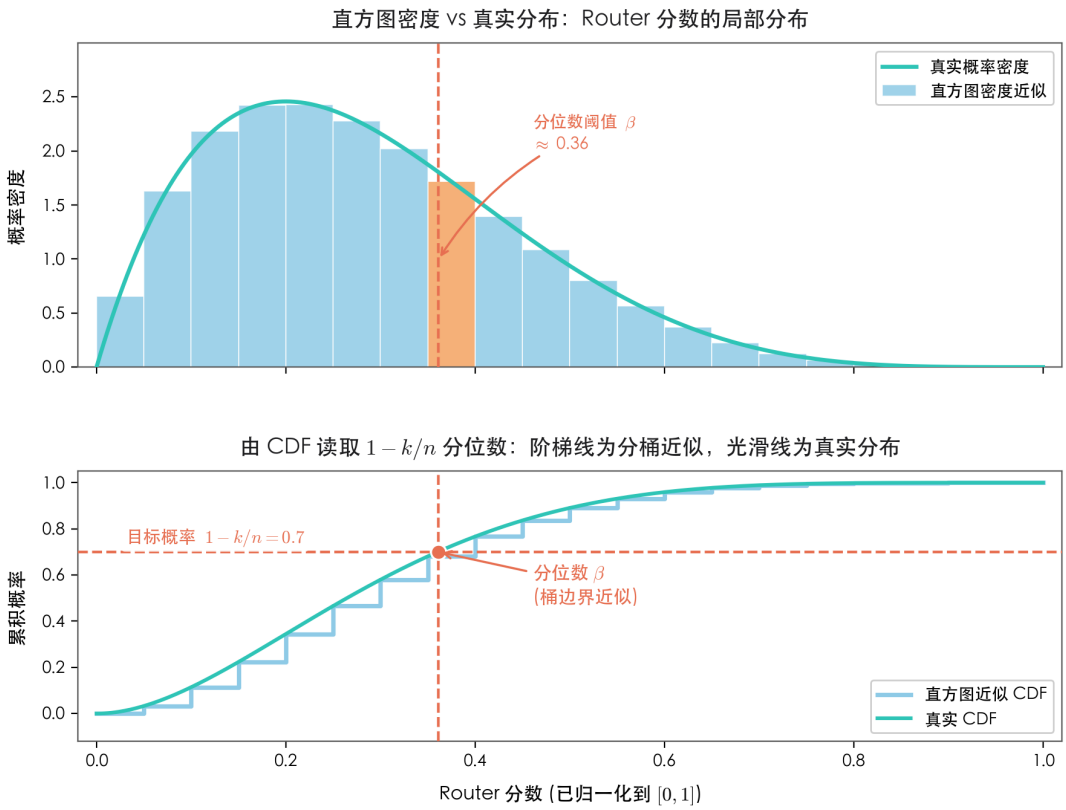

Figure 1: Schematic diagram of histogram-based approximation for quantile estimation

divide $[0, 1]$ into b equal buckets, discretize each value in $\boldsymbol{s}$ to an integer, and convert it to a corresponding one-hot vector.

This yields an $l \times n \times b$ binary array. Averaging along the sequence dimension l gives the score distribution for each expert (as a b-dimensional vector), from which quantiles can be computed. However, to preserve causality, we cannot average over the entire sequence. Instead, we use a "Cumulative Average"-like approach. To avoid the sharp boundaries of strict "Moving Quantile" and to make the process smoother, we use Exponential Moving Average (EMA).

Thus, after one-hot encoding $\boldsymbol{s}$ by buckets, we apply EMA along the sequence dimension. Each token yields a local distribution, from which the $(1 - k/n)$-quantile is computedthis becomes the key variable $\boldsymbol{\beta}_i$ of MQB. Since EMA is linear and parallelizable, we achieve sequence-level load balancing in a relatively efficient, Loss-Free manner. This is the final version of MQB.

Moving Quantile Balancing (MQB)
Input: score matrix $\boldsymbol{s} \in [0, 1]^{l \times n}$ Output: corrected score matrix $\hat{\boldsymbol{s}} \in \mathbb{R}^{l \times n}$
$1:\quad \boldsymbol{h}_{i,j} = \text{one\_hot}(\lfloor s_{i,j} \times b \rfloor) \in \{0, 1\}^b$ $2:\quad \bar{\boldsymbol{h}}_{i,j} = \gamma \bar{\boldsymbol{h}}_{i-1,j} + (1 - \gamma)\boldsymbol{h}_{i,j}$ $3:\quad m^*_{i,j} = \min\{m \mid \sum_{t=0}^{m} \bar{h}_{i,j,t} \geq 1 - \frac{k}{n}\}$ $4:\quad \beta_{i,j} = (m^*_{i,j} + 1/2)/b$ $5:\quad \hat{\boldsymbol{s}}_i = \boldsymbol{s}_i - \boldsymbol{\beta}_i,$

7 General Cases

So far, our discussion has focused on dynamic activation MoE, where experts satisfying $\boldsymbol{s}_i - \boldsymbol{\beta}_i > 0$ are activated (variable count). From "A Journey Through MoE: 6. Optimal Allocation for Balance", we know that for Top-k MoE, $\boldsymbol{\beta}$ lacks a simple closed-form solution and requires iterative alternation.

How then can Top-k MoE achieve sequence-level load balancing in a Loss-Free manner? The fact that MQB achieves sequence-level balance in dynamic activation implies that it flattens local peaks in router scores. Selecting Top-k from $\boldsymbol{s}_i - \boldsymbol{\beta}_i$ may still be unbalanced, but only at a global level. Thus, applying an additional global QB step can restore balance.

In other words, for Top-k MoE, our method for enforcing sequence-level balance is MQB+QB: applying QB again on top of $\boldsymbol{s}_i - \boldsymbol{\beta}_i$ (or using SignSGD). This achieves sequence-level load balancing. However, perfect sequence-level balance often hurts performance significantly. Usually, global balance sufficessequence-level balancing is only to prevent extreme casesso its strength should not be excessive.

To minimize impact on the main task, Aux Loss can use a small coefficient; for MQB, one can consider taking a weighted average of scores:

$$\lambda(\boldsymbol{s}_i - \boldsymbol{\beta}_i) + (1 - \lambda)\boldsymbol{s}_i = \boldsymbol{s}_i - \lambda\boldsymbol{\beta}_i, \qquad \lambda \in [0, 1] \tag{6}$$

That is, introducing $\lambda < 1$ weakens the effect of sequence-level balancing. Global balance can then be enforced by applying QB afterward (since $\lambda < 1$ alone cannot guarantee balance, applying QB is necessary for both dynamic and Top-k MoE).

8 Other Details

After bucketing, the core operation of MQB becomes EMA. Thus, we ultimately return to EMAsimilar in form to the first method proposed by @ChangJonathanC, but differing in the object of EMA: @ChangJonathanC applies EMA directly to the original $l \times n$ scores, while MQB expands it to $l \times n \times b$ before EMA.

This resembles linear attention, which expands capacity via outer products, enabling more memory. The optimal properties of QB ensure actual sequence-level balance rather than just heuristic balance. Also like linear attention, EMA is an RNNso at inference time, an additional $n \times b$-sized state vector must be stored. In practice, $b = 100$ yields good performance, making this state size acceptable.

Furthermore, both QB and MQB are only used in expert activation decisions. The final gating weights for expert selection are still computed from the original router scores (before subtracting the bias)a shared trait of all Loss-Free methods, ensuring minimal interference with the main model.

Finally, the current experiments use Sigmoid activation followed by uniform bucketing on $[0, 1]$. How to bucket more finely remains an open experimental question, left to interested readers.

9 Experimental Results

The author conducted preliminary experiments on MQB using a 3B-parameter MoE model with 128 experts and $k = 4$. The EMA coefficient for MQB was set to 0.99 and the number of buckets to 100. MaxVio was used as the load balancing metric. Since the first MoE layer is the most imbalanced, results below focus on that layer.

First, the full MQB with $\lambda = 1$:

It is evident that $\lambda = 1$ achieves excellent load balance at the cost of a 0.06 increase in lossa significant degradation, indicating that excessive sequence-level balance harms model

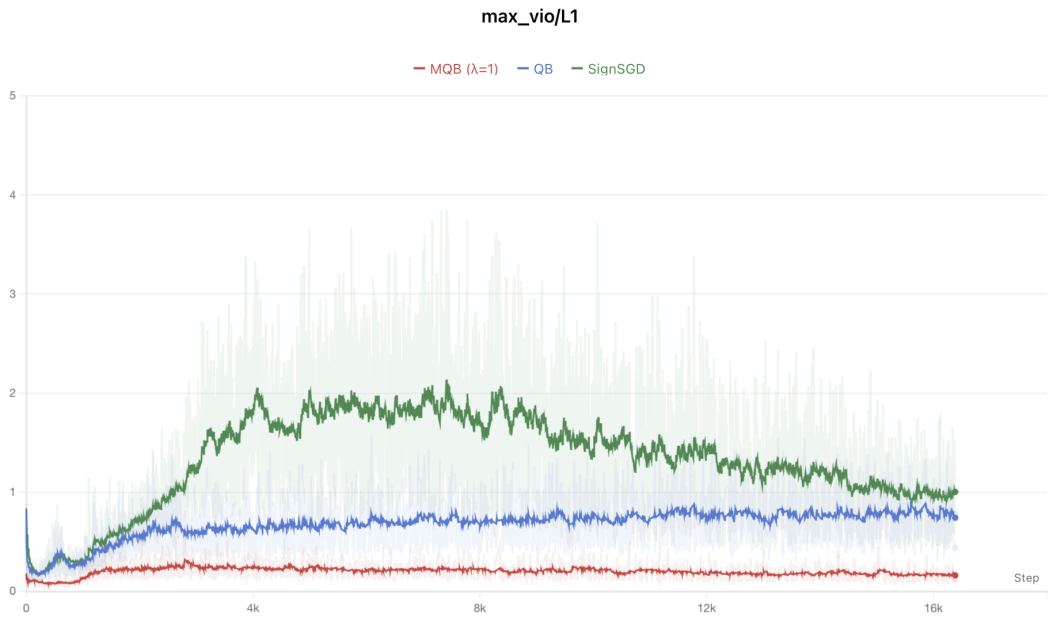

Figure 2: Comparison of MQB ($\lambda = 1$) with QB and SignSGD

performance. However, reducing λ to around 0.3 keeps the loss nearly unchanged while still improving balance, as shown below:

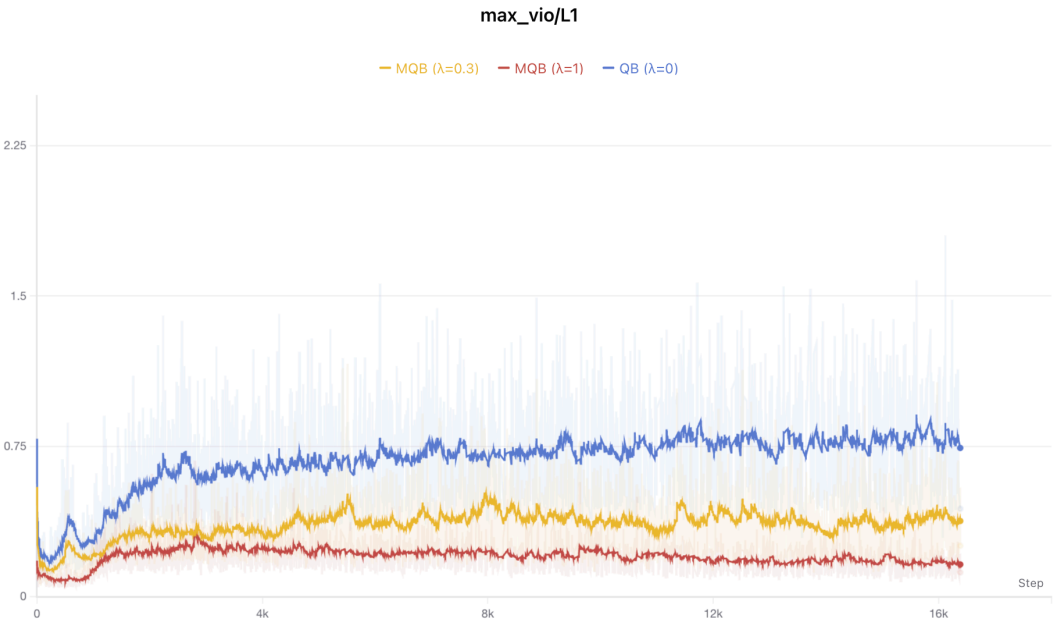

Figure 3: Load balancing performance for different values of λ

10 Further Thoughts

From these experiments, we further confirm that perfect sequence-level balance ($\lambda = 1$) significantly harms performance. With $\lambda = 0.3$, performance is largely preserved while still improving balance at each layer.

Overall, this article answers the question "how to achieve sequence-level load balancing in a Loss-Free manner", but does not resolve whether such balance is necessary,

why it might be needed, or to what extent. A preliminary understanding is that extreme sequence-level imbalance might degrade sequence-level performance (collapsing into a small effective model), justifying mild encouragement of sequence-level balance.

Besides MQB, other methods can achieve sequence-level balance in a Loss-Free way. For example, "How are tid2eid values in DeepSeek V4 generated?" introduces Hash Routing, which is a simple way to enforce sequence-level balance. However, Hash Routing redefines the routing form and thus lies outside conventional approaches.

The value of MQB is its compatibility with standard MoE forms, transforming "tuning an Aux Loss coefficient" into "adjusting an interpretable local bias term", allowing intuitive control over intervention intensity. Its parallelizability ensures efficiency. Since it only affects routing decisions without injecting gradients, its impact on main-task performance is minimized.

11 Summary

This article explores "how to achieve sequence-level load balancing in a Loss-Free manner". Starting from Quantile Balancing (QB), we derived Moving Quantile Balancing (MQB), successfully realizing this goal. However, whether sequence-level balancing is necessary, and to what extent, remains an open question.