A Brief Talk on K3's MoE and Attention

Su Jianlin

August 4, 2026

Last month, we released K3, our largest open-source model to date.

As the successor to K2, K3 is not a redesign from scratch, but a natural evolution along our past series of work, incorporating our latest understanding and improvements regarding effectiveness, efficiency, and stability. It can be said that it is a continuous, "all-encompassing" research result, rather than an all-or-nothing gamble.

In this article, let's talk about some of K3's architectural design philosophies.

1 Foreword

Simply put, in terms of architecture, K3 = KDA + MLA + Stable LatentMoE + AttnRes. The training optimizer is still the Moonlight version of Muon, but the weights of the Attention part have been changed to a Per-Head form for optimization.

Among these things, we have previously given a detailed introduction to AttnRes in *Attention Residuals Memoirs*, and we have also shared a detailed technical report on KDA, *Kimi Linear: An Expressive, Efficient Attention Architecture*. As for Per-Head Muon, it actually has no advantage in terms of effectiveness (of course, no disadvantage either), and switching to it is more out of correctness considerations (each Head is inherently relatively independent and should not be coupled together).

So next, we will mainly focus on the MoE and MLA parts: the former will discuss how we "tamed" LatentMoE; the latter will discuss our trade-offs in Attention.

2 Mixture of Experts

The MoE used in K3, we call it "Stable LatentMoE". As the name implies, it is a "Stable" "LatentMoE". LatentMoE is not new; it comes from *LatentMoE: Toward Optimal Accuracy per FLOP and Parameter in Mixture of Experts*. The benefit is that it can achieve better effectiveness under roughly the same training and inference costs, but its introduction also triggered stability issues, so we proposed some improvements.

2.1 SiTU

In current mainstream MoE architectures, a single Expert usually adopts the SwiGLU form:

$$\boldsymbol{W}_3(\mathrm{SiLU}(\boldsymbol{W}_1\boldsymbol{x}) \odot \boldsymbol{W}_2\boldsymbol{x}) \tag{1}$$

where $\mathrm{SiLU}(x) = x\sigma(x)$ (Sigmoid Linear Unit, also known as Swish), and σ is the Sigmoid function. As the main source of non-linearity, a frequent problem with SwiGLU is that a certain row $\boldsymbol{w}$ of $\boldsymbol{W}_1$ aligns with a certain input $\boldsymbol{x}$, causing the output $\boldsymbol{w} \cdot \boldsymbol{x}$ to be very large. More extremely, this phenomenon also occurs simultaneously in $\boldsymbol{W}_2\boldsymbol{x}$, and at the exact same position, resulting in outliers on the order of $\mathcal{O}(\|\boldsymbol{x}\|^4)$ in the intermediate part.

To address this, we first replaced SiLU with SiTU (Sigmoid Tanh Unit):

$$\mathrm{SiTU}(x;\beta) = \underbrace{\beta \tanh\left(\frac{x}{\beta}\right)}_{\mathrm{softcap}(x;\beta)} \cdot \sigma(x) \tag{2}$$

This first constrains the gating part within $(-\beta, \beta)$, where $\beta = 4$. Further stress testing found that this still could not completely prevent expansion, so we simply added the softcap operation to the linear part as well, forming the current SiTU-GLU:

$$\boldsymbol{W}_3\Big(\mathrm{SiTU}(\boldsymbol{W}_1\boldsymbol{x};\beta_1) \odot \mathrm{softcap}(\boldsymbol{W}_2\boldsymbol{x};\beta_2)\Big) \tag{3}$$

where $\beta_1 = 4, \beta_2 = 25$. Introducing Clip operations into SwiGLU is no longer new; GPT-OSS and DSV4 have already introduced Hard Clip operations. However, we found that under the same bounds, softcap often achieves better results, so we chose softcap.

2.2 Norm

The difference between LatentMoE and MoE is:

$$\text{MoE:} \qquad \underbrace{d \to D \to d}_{n \text{ select } k} \tag{4}$$

$$\text{LatentMoE:} \qquad d \to \underbrace{d/2 \to D \to d/2}_{2n \text{ select } 2k} \to d \tag{5}$$

That is, LatentMoE first reduces the dimension, then performs the $2n$ select $2k$ MoE, and finally increases the dimension. This keeps the training and inference costs roughly the same, but the effectiveness is slightly better. The dimension reduction/increase are both linear projections. The original LatentMoE did not add extra operations here, so adding the MoE in the middle resulted in a pattern of four consecutive matrix multiplications, which was extremely unstable.

To stabilize training, a naive idea is to add an RMS Norm at both positions: after dimension reduction (the output of $d \to d/2$) and before dimension increase (the input of $d/2 \to d$). But further ablation found that the RMS Norm at the latter position plays the most essential role, so following the "principle of minimal modification", we only kept the last RMS Norm, forming the current Stable LatentMoE structure.

Afterwards, we conducted more careful comparative experiments and found that this extra RMS Norm not only stabilizes training but also has a magical effect on effectiveness. Specifically, when all models can converge normally, whether adding this RMS Norm or not has little impact on the Valid Loss, but for certain Benchmarks, not adding this RMS Norm leads to a stable degradation.

The reason might be that this Norm better balances the ratio between Routed Expert and Shared Expert (in practice, with this Norm added, there is no need for an extra Scaling Factor), or it might be because Norm is a non-linear operation (although the non-linearity is extremely weak), and its addition invisibly increases the equivalent depth of LatentMoE.

2.3 QB

K3's MoE was originally designed to be 448 select 8, and after introducing LatentMoE, it became 896 select 16. Although the sparsity remains unchanged, the increase in the total number of Experts still exacerbates the load balancing problem.

Like the previous generation model K2, K3 also adopts the Loss-Free load balancing scheme. However, the SignSGD-style update rule used previously has become somewhat unstable under K3's large total number of Experts, so we introduced QB (Quantile Balancing). The advantage is that it is mathematically more reasonable and has no extra hyperparameters. Its details have been introduced in *MoE Travelogue: 6, Optimal Allocation Promotes Balance*.

The core operation of QB is to find the global quantile, but quantile is a non-linear operation. If calculated naively, the communication cost is very large. The approach I previously proposed was to calculate local quantiles and then take a global average, but it was later found that as the scale further expands, this approach is still not accurate enough. Therefore, K3 ultimately used a binning approximation (histogram estimation): after compressing the scores requiring quantiles to 0–1, it estimates the distribution of scores by binning, and then reads the quantile from the distribution.

For the number of bins, we found that 10,000 bins do not provide a better gain for load balancing than 1,000 bins, so it is recommended to use 1,000 bins. Due to the additivity of distributions, we can aggregate distribution information across machines and across gradient accumulation with extremely low communication volume, thereby obtaining the global approximate quantile.

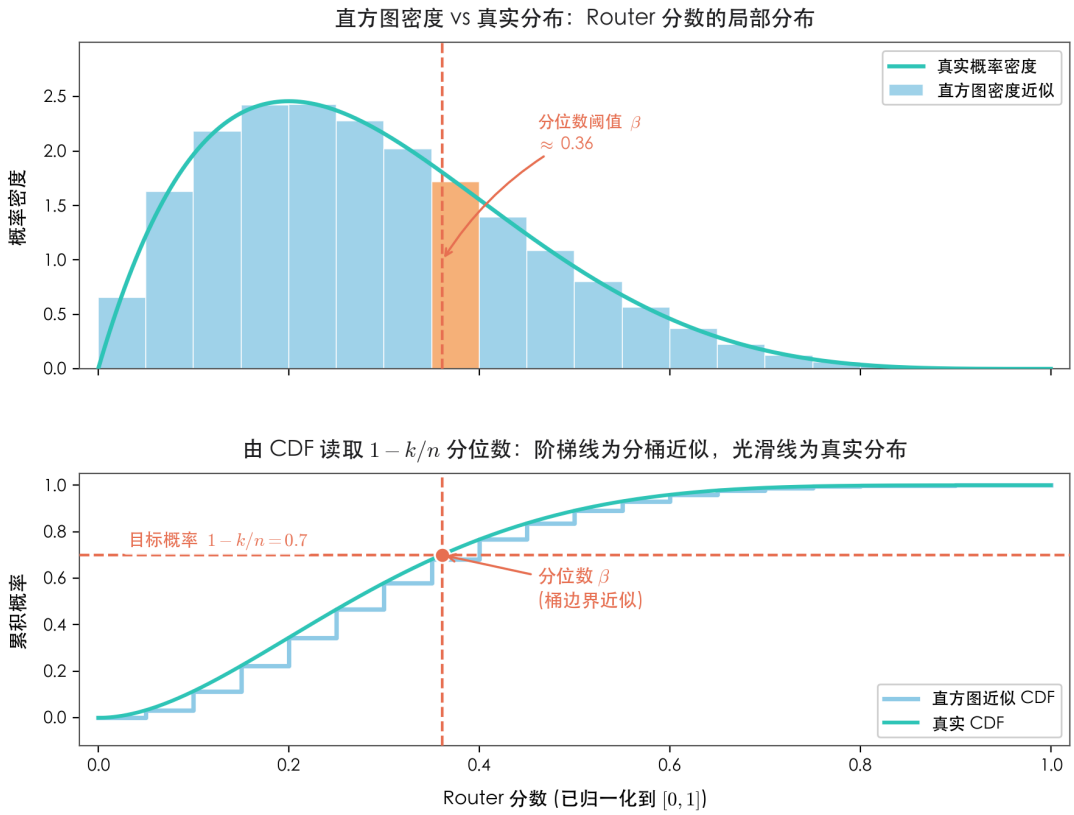

Figure 1: Schematic diagram of histogram approximate estimation of quantiles

3 Attention Mechanism

K3's attention is a hybrid of "KDA + MLA". Here we mainly talk about MLA. Of course, MLA is no longer new, but some details can be brought up for discussion. Some readers might scoff: DSV4 has given up on MLA, yet you guys are still using it, have you run out of ideas? Of course not. K3's use of MLA is still the result of careful consideration.

3.1 MLA

A year ago, I wrote *Transformer Upgrade Path: 20, Where is MLA Good? (Part 1)* and *Transformer Upgrade Path: 21, Where is MLA Good? (Part 2)* to explore the benefits of MLA from experimental and theoretical perspectives. The judgment I gave at the time was: "**Under the same training cost and inference cost, MLA might be the most effective Full Attention variant.**"

Is this judgment still correct today? Basically yes, but with some minor changes. Given a fixed training cost and KV Cache size, MLA is still nearly the optimal Attention. But now, besides KV Cache, Decoding has a new variable — MTP, or speculative decoding, whose idea is to trade computation for speed. However, MLA behaves like an MQA with head_dims=512+ during Decoding, which already consumes most of the compute in advance, so "MLA+MTP" easily suffers.

However, the choice of Attention requires comprehensive consideration from multiple aspects. MTP is only a part; switching to another design might be friendlier to MTP, but other aspects are not necessarily better.

MLA is an MHA with 192+128 (qk_dims and v_dims) during the training phase. If scaled down, such as GQA8 with 128+128, it is hard to outperform MLA in terms of effectiveness. Even if it can tie, the KV Cache of GQA8 is more than three times that of MLA, which is unrealistic. Note that the addition of MTP only means that the Decoding speed is not solely determined by the KV Cache size; it does not mean that the KV Cache can be arbitrarily large. In long-context scenarios, the smaller the KV Cache, the better.

If scaled up, such as switching to MFA with 256+256 (essentially an MQA), the effectiveness can indeed be caught up, and the KV Cache is not larger than MLA, but the training cost goes up, making it highly likely to suffer in Scaling Law. Moreover, the Prefill cost also increases at this time, which is equally non-negligible — because in the current mainstream Agent/Coding scenarios, the Prefill length in each round is also not short.

Therefore, an ideal Attention design better than MLA must at least satisfy the following conditions:

1. The effectiveness must not be worse than MLA (guaranteeing effectiveness);
2. The training and Prefill costs must at least not exceed MLA (computational efficiency);
3. The KV Cache is smaller than MLA (extra-long context);

4. The computation volume of Decoding is smaller than MLA (MTP friendly).

At least from my current perspective, there is no simple and elegant Attention design that can simultaneously satisfy the above characteristics, so we must make trade-offs. In the context of mixing with KDA, some of MLA's problems are mitigated, so we still chose MLA.

3.2 DSV4

Let's add an interlude here and briefly discuss the Attention of DSV4. DSV4 appears to have abandoned MLA and redesigned a completely different Attention, but if we scrutinize it, we will find that it still has the shadow of MLA and conforms to the two MLA blog posts mentioned above and the four directions just discussed.

In *Transformer Upgrade Path: 20, Where is MLA Good? (Part 1)*, we found experimentally that the invisibly enlarged head_dims are the key to MLA's effectiveness. Then in *Transformer Upgrade Path: 21, Where is MLA Good? (Part 2)*, we pointed out that given a fixed KV Cache size, the best-performing Attention is "an MQA where head_dims equals the KV Cache size, and K and V are shared".

Thus, DSV4 directly replaced Attention with an MQA with head_dims=512 and K=V (the positional encoding is QKVO-RoPE) to guarantee effectiveness, which is exactly the Decoding form of MLA. But doing so causes the computation of training and Prefill to surge, and DSV4 does not have linear attention; every layer has a KV Cache, and even if each Token only has 512 dimensions, it is still unbearable. To this end, DSV4 introduced Sparse and Compress: Sparse saves computation, and Compress further compresses the KV Cache, also saving computation.

Therefore, rather than saying DSV4 abandoned MLA, it is better to say that following the directions mentioned in the previous section, it pushed MLA to another extreme. However, this extension is not without cost. First is the complexity in Infra, and second, with such aggressive Sparse and Compress, its optimality feels like it needs further careful verification. But all in all, from MLA to DSV4, it is more like an inheritance and upgrade, rather than abandonment and rebuilding.

Of course, the "Linear+Full" route also has its shortcomings, so compared to the Sparse route, it is currently unknown which one can go further.

3.3 NoPE

K3's MLA has another detail: while maintaining the standard MLA structure, it directly removes RoPE, becoming NoPE. This change already appeared in Kimi Linear, but after K3 was released, it sparked some discussion again.

First, K3 can add RoPE back, but after adding RoPE, the effectiveness looks unchanged, so following the principle of simplicity, we simply do not add it. But note that if

it were a full MLA model like K2, RoPE is crucial, and removing RoPE would significantly degrade performance. K3 can use NoPE because it is a hybrid model of "KDA+MLA".

Why can "KDA+MLA" use NoPE? We derived in *Transformer Upgrade Path: 6, Completeness Analysis of Rotational Positional Encoding* that the power of any orthogonal matrix can be used to construct a generalized RoPE. The commonly referred to RoPE chooses simple rotation matrices, while PaTH tried another choice — the Householder matrix, and it also performed well.

In *A Brief History of Linear Attention: From Imitation, Innovation to Feedback*, we derived that in the orthogonal case, PaTH can be equivalently written as

$$\text{SoftmaxAttention}(\underbrace{\boldsymbol{Q} - \text{DeltaNet}(\boldsymbol{Q},\boldsymbol{W},\boldsymbol{W})}_{\tilde{\boldsymbol{Q}}}, \underbrace{\boldsymbol{K} - \text{DeltaNet}(\boldsymbol{K},\boldsymbol{W},\boldsymbol{W})}_{\tilde{\boldsymbol{K}}}, \boldsymbol{V}) \tag{6}$$

This equivalent form shows that adding DeltaNet to $\boldsymbol{Q},\boldsymbol{K}$ can also play a positional encoding role similar to RoPE. We know that KDA is a more general DeltaNet, so the hybrid model of KDA+MLA inherently comes with a positional encoding effect similar to RoPE and PaTH. We can also say that it's not that K3 no longer has RoPE, but rather that KDA implicitly provides a generalized RoPE.

In addition, there is an aesthetic issue: since MLA no longer adds RoPE, why does it still retain the practice of concatenating another 64 dimensions?

There are many reasons for this, such as better adapting to existing MLA infrastructure without rewriting a set of code. More importantly, if we directly project out a 576-dimensional Latent, and then project out 192+128-dimensional K and V, it would indeed be more elegant, but the computation would increase, and there would be no gain in effectiveness, making it a losing trade-off overall. Finally, K3 has already introduced new variables like KDA and AttnRes, so we don't want to introduce too many variables into MLA. After all, you have to eat a meal one bite at a time.

4 Article Summary

This article briefly talked about the design and trade-offs of K3 in the two components of MoE and Attention. Overall, none of K3's modifications are radical or flashy; behind them are basically clear motivations and experimental support. The coordination among effectiveness, efficiency, and stability remains the main theme of architectural design.