Muon Implementation Based on Streaming Power Iteration: 3. Refinement

Su Jianlin

2026-04-07

Reviewing the previous two articles "Muon Implementation Based on Streaming Power Iteration: 1. First Encounter" and "Muon Implementation Based on Streaming Power Iteration: 2. Acceleration", we introduced the streaming power iteration scheme for Muon, initially verified its feasibility, and further discussed acceleration of the core operation—QR decomposition—to approach the efficiency of the Newton–Schulz iteration implementation.

In this article, we move beyond optimizing QR decomposition in isolation and instead take a more holistic view of streaming power iteration. By incorporating the specific computational context, we perform further "refinement" of its implementation details, aiming to reduce computational bottlenecks as much as possible and push its efficiency toward the theoretical limit.

Existing Results

Streaming power iteration essentially performs "SVD while training." The idea is to compute SVD via power iteration, using the result from the previous step as a cache to amortize computation across training steps, thus making it feasible to embed SVD within optimizers. As for Muon, it is merely a basic application, since the core operation msign of Muon is fundamentally implemented via SVD. Specifically, the update formula of Muon is

$$\begin{aligned}\boldsymbol{M}_t &= \beta \boldsymbol{M}_{t-1} + \boldsymbol{G}_t \\ \boldsymbol{W}_t &= \boldsymbol{W}_{t-1} - \eta_t[\text{msign}(\boldsymbol{M}_t) + \lambda \boldsymbol{W}_{t-1}]\end{aligned} \tag{1}$$

where all matrices are of size $n \times m$ and we assume $n \geq m$. Suppose the SVD of $\boldsymbol{M}$ is $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ (with $\boldsymbol{U} \in \mathbb{R}^{n\times m}$, $\boldsymbol{\Sigma}, \boldsymbol{V} \in \mathbb{R}^{m\times m}$), then $\text{msign}(\boldsymbol{M}) = \boldsymbol{U}\boldsymbol{V}^\top$, so achieving SVD equates to achieving msign. Direct SVD, however, is typically expensive; streaming power iteration makes it practically feasible.

In the previous article, we discussed four acceleration strategies for streaming power iteration. The first—using full-precision FP32 multiplication—is generally applicable, while the latter three are somewhat mutually exclusive, requiring a choice among them. The author recommends the second approach, which has a higher theoretical ceiling. The refinements discussed below are based on the second approach. Substituting this approach into the streaming power iteration for Muon yields the iteration formula:

$$\begin{aligned}\boldsymbol{M}_t &= \beta \boldsymbol{M}_{t-1} + \boldsymbol{G}_t \\ \boldsymbol{V}_t &= \text{QR}(\boldsymbol{M}_t^\top \,\text{QR}(\boldsymbol{M}_t \boldsymbol{V}_{t-1})) \\ \boldsymbol{U}_t &= \text{ColNorm}(\boldsymbol{M}_t \boldsymbol{V}_t) \\ \boldsymbol{W}_t &= \boldsymbol{W}_{t-1} - \eta_t(\boldsymbol{U}_t \boldsymbol{V}_t^\top + \lambda \boldsymbol{W}_{t-1})\end{aligned} \tag{2}$$

Clearly, the most expensive operation now is $\boldsymbol{V}_t = \mathrm{QR}(\boldsymbol{M}_t^\top \mathrm{QR}(\boldsymbol{M}_t\boldsymbol{V}_{t-1}))$, which is precisely the target of optimization going forward.

Accelerating Decomposition

To maintain efficiency, the QR here does not invoke the framework's built-in QR decomposition function but uses "Shifted Cholesky QR (SCQR)". SCQR splits the QR decomposition of a matrix $\boldsymbol{A}$ into two steps: (1) Cholesky decomposition of $\boldsymbol{A}^\top\boldsymbol{A} + \lambda\boldsymbol{I}$ to obtain an upper triangular matrix $\boldsymbol{R}$; (2) solving $\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{A}$ for the orthogonal matrix $\boldsymbol{Q}$.

Theoretically, both steps are highly efficient. However, a detection step is required to fall back to the standard QR function if SCQR fails; standard QR almost always succeeds. Triggering fallback, though, significantly degrades end-to-end efficiency. The main reason for SCQR failure is its reliance on the condition number of the matrix—the term $+\lambda\boldsymbol{I}$ is specifically used to reduce the condition number of $\boldsymbol{A}^\top\boldsymbol{A}$.

However, this creates a dilemma: a larger λ increases SCQR's success rate but leads to greater deviation from orthogonality (i.e., larger error), degrading performance; a smaller λ improves accuracy but raises the probability of falling back to standard QR, thereby reducing efficiency. Empirical testing shows that setting $\lambda = \epsilon\|\boldsymbol{A}^\top\boldsymbol{A}\|_F$ with $\epsilon = 10^{-9}$ strikes a good balance between performance and efficiency.

The acceleration methods discussed in the previous article all focused on reducing the condition number. The initial version of streaming power iteration was $\boldsymbol{V}_t = \mathrm{QR}(\boldsymbol{M}_t^\top\boldsymbol{M}_t\boldsymbol{V}_{t-1})$, where the matrix subjected to Cholesky decomposition is $\boldsymbol{V}_{t-1}^\top(\boldsymbol{M}_t^\top\boldsymbol{M}_t)^2\boldsymbol{V}_{t-1}$, effectively raising the condition number of $\boldsymbol{M}_t$ to the fourth power—a dramatic increase. By contrast, $\boldsymbol{V}_t = \mathrm{QR}(\boldsymbol{M}_t^\top \mathrm{QR}(\boldsymbol{M}_t\boldsymbol{V}_{t-1}))$ requires two QR steps, but the condition number for each Cholesky decomposition is only the square of the condition number of $\boldsymbol{M}_t$, significantly reducing it. As a result, SCQR success rates greatly improve, and overall speed increases.

Adjusting Order

The content above merely recaps the previous two articles (apologies for the lengthy preamble, but sharpening the tool is essential for efficient chopping). In this section, we begin discussing new optimization strategies. Upon careful observation, we note that two QR steps are involved, each considered independently. However, @YouJiacheng and @Kimi discovered that further acceleration is possible by considering them jointly.

By default, the computation of $\boldsymbol{V}_t = \mathrm{QR}(\boldsymbol{M}_t^\top \mathrm{QR}(\boldsymbol{M}_t\boldsymbol{V}_{t-1}))$ proceeds as:

$$
\begin{aligned}
\boldsymbol{A}_{(1),t} &= \boldsymbol{M}_t\boldsymbol{V}_{t-1} \\
\boldsymbol{R}_{(1),t}^\top\boldsymbol{R}_{(1),t} &= \boldsymbol{A}_{(1),t}^\top\boldsymbol{A}_{(1),t} + \lambda\boldsymbol{I} \quad \text{(Cholesky decomposition)} \\
\boldsymbol{Q}_{(1),t} &= \boldsymbol{A}_{(1),t}\boldsymbol{R}_{(1),t}^{-1} \quad \text{(Triangular Solve)} \\
\boldsymbol{A}_{(2),t} &= \boldsymbol{M}_t^\top\boldsymbol{Q}_{(1),t} \\
\boldsymbol{R}_{(2),t}^\top\boldsymbol{R}_{(2),t} &= \boldsymbol{A}_{(2),t}^\top\boldsymbol{A}_{(2),t} + \lambda\boldsymbol{I} \quad \text{(Cholesky decomposition)} \\
\boldsymbol{Q}_{(2),t} &= \boldsymbol{A}_{(2),t}\boldsymbol{R}_{(2),t}^{-1} \quad \text{(Triangular Solve)}
\end{aligned}
\tag{3}
$$

Here, operations involving $\boldsymbol{M}_t\boldsymbol{V}_{t-1}$, $\boldsymbol{A}_{(1),t}^\top\boldsymbol{A}_{(1),t}$, $\boldsymbol{A}_{(1),t}\boldsymbol{R}_{(1),t}^{-1}$, and $\boldsymbol{M}_t^\top\boldsymbol{Q}_{(1),t}$ are all $\mathcal{O}(nm^2)$; the rest are $\mathcal{O}(m^3)$. When $n \gg m$, the $\mathcal{O}(nm^2)$ steps may become a bottleneck. Interestingly, through

algebraic rearrangement, we can reduce the $\mathcal{O}(nm^2)$ steps to just one:

$$\begin{aligned}
\boldsymbol{A}_{(1),t} &= (\boldsymbol{M}_t^\top \boldsymbol{M}_t)\boldsymbol{V}_{t-1} \\
\boldsymbol{R}_{(1),t}^\top \boldsymbol{R}_{(1),t} &= \boldsymbol{V}_{t-1}^\top \boldsymbol{A}_{(1),t} + \lambda \boldsymbol{I} \quad \text{(Cholesky decomposition)} \\
\boldsymbol{A}_{(2),t} &= \boldsymbol{A}_{(1),t}\boldsymbol{R}_{(1),t}^{-1} \quad \text{(Triangular Solve)} \\
\boldsymbol{R}_{(2),t}^\top \boldsymbol{R}_{(2),t} &= \boldsymbol{A}_{(2),t}^\top \boldsymbol{A}_{(2),t} + \lambda \boldsymbol{I} \quad \text{(Cholesky decomposition)} \\
\boldsymbol{Q}_{(2),t} &= \boldsymbol{A}_{(2),t}\boldsymbol{R}_{(2),t}^{-1} \quad \text{(Triangular Solve)}
\end{aligned} \tag{4}$$

This equivalent version deserves deep contemplation! First, it can be proven to be theoretically equivalent to the original version, and this equivalence does not depend on the absolute orthogonality of $\boldsymbol{V}_{t-1}$ and $\boldsymbol{Q}_{(1),t}$. After transformation, only $\boldsymbol{M}_t^\top \boldsymbol{M}_t$ is $\mathcal{O}(nm^2)$, all remaining steps are $\mathcal{O}(m^3)$, and the total number of steps is reduced by one (merging the prior steps $\boldsymbol{Q}_{(1),t} = \boldsymbol{A}_{(1),t}\boldsymbol{R}_{(1),t}^{-1}$ and $\boldsymbol{A}_{(2),t} = \boldsymbol{M}_t^\top \boldsymbol{Q}_{(1),t}$ into a single operation)!

> **Note 1:** As stated by @YouJiacheng, this ingenious transformation was automatically discovered by Kimi after he relayed Eq. (3) to it.

> **Note 2:** Eqs. (3) and (4) differ slightly: in Eq. (3), the first step is $(\boldsymbol{M}_t\boldsymbol{V}_{t-1})^\top(\boldsymbol{M}_t\boldsymbol{V}_{t-1})$, while in Eq. (4), it is $\boldsymbol{V}_{t-1}^\top(\boldsymbol{M}_t^\top\boldsymbol{M}_t)\boldsymbol{V}_{t-1}$. These are mathematically equivalent, but finite-precision floating-point arithmetic leads to small differences; the latter results in a larger condition number for the matrix, which may require slightly increasing the regularization parameter during Cholesky decomposition.

Simplifying Regularization

For the matrix $\boldsymbol{A}^\top\boldsymbol{A}$, the regularization term used during Cholesky decomposition is $\lambda\boldsymbol{I}$, where $\lambda = \epsilon\|\boldsymbol{A}^\top\boldsymbol{A}\|_F$ and $\epsilon = 10^{-9}$. The rationale behind this choice has not yet been fully explained. Here, we elaborate and derive a simpler regularization term incorporating the problem context.

Due to the positive-definite symmetric nature of $\boldsymbol{A}^\top\boldsymbol{A}$, its SVD coincides with its eigenvalue decomposition. Let $\boldsymbol{A}^\top\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be its SVD. Then $\boldsymbol{A}^\top\boldsymbol{A} + \lambda\boldsymbol{I} = \boldsymbol{V}(\boldsymbol{\Sigma} + \lambda\boldsymbol{I})\boldsymbol{V}^\top$. Denoting the largest and smallest singular values of $\boldsymbol{A}^\top\boldsymbol{A}$ as $\sigma_{\max}, \sigma_{\min}$, the singular values of $\boldsymbol{A}^\top\boldsymbol{A} + \lambda\boldsymbol{I}$ become $\sigma_{\max} + \lambda, \sigma_{\min} + \lambda$. The condition number, defined as the ratio of largest to smallest singular values, decreases from $\sigma_{\max}/\sigma_{\min}$ to

$$\frac{\sigma_{\max} + \lambda}{\sigma_{\min} + \lambda} < \frac{\sigma_{\max} + \lambda}{\lambda} = \frac{\sigma_{\max}}{\lambda} + 1 \tag{5}$$

To keep the condition number below $1/\epsilon + 1$, we need $\lambda \geq \epsilon\sigma_{\max}$. This indicates that ideally, the spectral norm (i.e., the largest singular value) of $\boldsymbol{A}^\top\boldsymbol{A}$ should be used as the basis for tuning λ. However, computing the spectral norm is complex, so we use the simpler Frobenius norm $\|\boldsymbol{A}^\top\boldsymbol{A}\|_F$, leading to $\lambda = \epsilon\|\boldsymbol{A}^\top\boldsymbol{A}\|_F$; the choice $\epsilon = 10^{-9}$ is purely empirical.

Yet, "using the Frobenius norm instead of the spectral norm due to computational complexity" is a general rule for arbitrary matrices. The streaming power iteration we are developing is itself a method for computing SVD. As training progresses, $\boldsymbol{V}_t$ increasingly approaches the right singular matrix of $\boldsymbol{M}_t$. Since $\boldsymbol{M}_t$ evolves slowly, so does $\boldsymbol{V}_{t-1}$. Consequently, $\boldsymbol{V}_{t-1}^\top\boldsymbol{M}_t^\top\boldsymbol{M}_t\boldsymbol{V}_{t-1}$ approaches a diagonal matrix, and its top-left element increasingly approximates its spectral norm.

Similarly, $\tilde{\boldsymbol{U}}_t = \mathrm{QR}(\boldsymbol{M}_t\boldsymbol{V}_{t-1})$ approaches the left singular matrix of $\boldsymbol{M}_t$, so $\tilde{\boldsymbol{U}}_t^\top\boldsymbol{M}_t\boldsymbol{M}_t^\top\tilde{\boldsymbol{U}}_t$ also approaches diagonality, with its top-left element approaching the spectral norm. Therefore, in our

context, the simplest and most accurate choice is to use $(\boldsymbol{A}^\top \boldsymbol{A})_{[0,0]}$ as an approximation for the spectral norm—i.e., simply set $\lambda = \epsilon \cdot (\boldsymbol{A}^\top \boldsymbol{A})_{[0,0]}$. Empirical results show that $\epsilon = 10^{-7}$ achieves a good balance between effectiveness and efficiency.

Reference Implementation

Combining the modifications from the previous two sections, the reference implementation for the iteration from $\boldsymbol{V}_{t-1}$ to $\boldsymbol{V}_t$ is as follows:

```
import jax.numpy as jnp
from jax.scipy.linalg import solve_triangular
from jax import lax

def shift_old(A, eps=1e-9):
    return A + eps * jnp.linalg.matrix_norm(A, keepdims=True) * jnp.eye(A.shape[-1])

def scqr(A, eps=1e-9):
    """First attempt Shifted Cholesky QR; fall back to default QR if failed."""
    R = jnp.linalg.cholesky(shift_old(A.mT @ A, eps), upper=True)
    Q = solve_triangular(R.mT, A.mT, lower=True).mT
    return lax.cond(jnp.isfinite(Q).all(), lambda: Q, lambda: jnp.linalg.qr(A)[0])

def v_step_old(M, V, eps=1e-9):
   return scqr(M.mT @ scqr(M @ V, eps), eps)


def shift(A, eps=1e-7):
    return A + eps * A[..., :1, :1] * jnp.eye(A.shape[-1])

def v_step(M, V, eps=1e-7):
    A = (M.mT @ M) @ V
    R = jnp.linalg.cholesky(shift(V.mT @ A, eps), upper=True)
    B = solve_triangular(R.mT, A.mT, lower=True).mT
    R = jnp.linalg.cholesky(shift(B.mT @ B, eps), upper=True)
    Q = solve_triangular(R.mT, B.mT, lower=True).mT
    return lax.cond(jnp.isfinite(Q).all(), lambda: Q, lambda: jnp.linalg.qr(A)[0])
```

Contemporaneous Work

After the publication of "Muon Implementation Based on Streaming Power Iteration: 2. Acceleration" and before this article, some interesting optimization work emerged externally. These works share similar optimization ideas with the two improvements proposed in this paper, allowing us to study them together.

Following the publication of the previous article, @Ji_Ha_Kim proposed some improvement ideas. For instance, he noted from conversations with GPT (see link) that we might be able to eliminate one Triangular Solve step. Specifically:

$$\begin{aligned}
\boldsymbol{V}_t &= \mathrm{QR}(\boldsymbol{M}_t^\top \mathrm{QR}(\boldsymbol{M}_t \boldsymbol{V}_{t-1})) \\
&= \mathrm{QR}(\boldsymbol{V}_{t-1}(\boldsymbol{M}_t \boldsymbol{V}_{t-1})^\top \mathrm{QR}(\boldsymbol{M}_t \boldsymbol{V}_{t-1})) \\
&= \mathrm{QR}(\boldsymbol{V}_{t-1}(\mathrm{QR}(\boldsymbol{M}_t \boldsymbol{V}_{t-1})^\top \boldsymbol{M}_t \boldsymbol{V}_{t-1})^\top) \\
&= \boldsymbol{V}_{t-1}\, \mathrm{QR}((\mathrm{QR}(\boldsymbol{M}_t \boldsymbol{V}_{t-1})^\top \boldsymbol{M}_t \boldsymbol{V}_{t-1})^\top)
\end{aligned} \tag{6}$$

It is evident that $\mathrm{QR}(\boldsymbol{M}_t\boldsymbol{V}_{t-1})^\top\boldsymbol{M}_t\boldsymbol{V}_{t-1}$ is simply the $\boldsymbol{R}$ matrix from the QR decomposition of $\boldsymbol{M}_t\boldsymbol{V}_{t-1}$, directly obtainable via Cholesky decomposition.

In other words, theoretically, the second QR could proceed directly after obtaining $\boldsymbol{R}$ from the first Cholesky decomposition, eliminating one Triangular Solve. However, this is only of theoretical interest, as it depends strictly on the orthogonality of $\boldsymbol{V}_{t-1}$ and $\mathrm{QR}(\boldsymbol{M}_t\boldsymbol{V}_{t-1})$. This condition holds only for exact QR (i.e., $\lambda = 0$). In practice, to ensure efficiency, we use SCQR, which produces results that are not strictly orthogonal. Thus, exploiting orthogonality early in the identity transformation may lead to error accumulation.

Meanwhile, the Tri-Dao team published "Gram Newton-Schulz: A Fast, Hardware-Aware Newton-Schulz Algorithm for Muon", presenting an acceleration strategy for the msign operator. By definition, $\mathrm{msign}(\boldsymbol{M}) = \boldsymbol{M}(\boldsymbol{M}^\top\boldsymbol{M})^{-1/2}$, their approach uses Newton–Schulz iteration to compute $(\boldsymbol{M}^\top\boldsymbol{M})^{-1/2}$ of size $m\times m$ instead of msign, significantly reducing computation when $n \gg m$. Many researchers have previously attempted this idea but failed; Tri-Dao succeeded through a "restart" technique.

Clearly, this optimization direction for msign aligns with the transformation from Eq. (3) to Eq. (4) in streaming power iteration. Coincidentally, @Ji_Ha_Kim suggested changing the Newton–Schulz iteration for msign from a polynomial to a rational form, achieving equally good results with fewer iterations. The issue with rational iteration is the need to invert a matrix. However, in the context of msign, only $m\times m$ positive definite symmetric matrices need inversion, achievable via Cholesky decomposition and two Triangular Solve steps—an acceptable cost.

Nonetheless, the computational flow of such rational iteration heavily overlaps with streaming power iteration and involves an additional Triangular Solve per step, so its speed appears unable to surpass that of streaming power iteration.

Summary

This article further "refined" the implementation details of streaming power iteration. The main improvements include: (1) adjusting the computation order to reduce $\mathcal{O}(nm^2)$ operations from four to one; (2) simplifying the regularization term by leveraging the specific context of streaming power iteration. These optimizations further reduce computational bottlenecks and push computational efficiency toward the theoretical limit.