



Beyond AdamW: A Practical PyTorch Walkthrough of Muon, Dion, and Orthogonalized Optimizer Variants

Jen Wei

Agenda



01

Intro

- What is Muon
- Challenges of Muon

02

Hopper vs AdamW

- My research experiment training RL via Hopper vs AdamW

03

Evolution of Muon

- Aimed scaling challenges of Muon

04

Integrating Dion3

- Integration into Olmo-Core
- Pre-training validation

What is Muon – SGD with Momentum + NS post-processing

Input: γ (learning rate), θ_0 (parameters), $f(\theta)$ (objective), λ (weight decay), μ (momentum), τ (dampening), nesterov, maximize, NS_steps.

for $t = 1$ to ... do

if $\lambda \neq 0$: $\theta_t \leftarrow \theta_{t-1} \cdot (1 - \gamma \cdot \lambda)$ # Apply weight decay directly to parameters

$b_t \leftarrow g_t$ # Initialize buffer with current gradient

if $t > 1$: $b_t \leftarrow \mu \cdot b_{t-1} + (1 - \mu) \cdot g_t$ # Apply momentum

if nesterov: $b_t \leftarrow g_t + \mu \cdot b_t$

if $\dim(b_t) \geq 2$:

Apply Newton-Schulz orthogonalization

$b_t \leftarrow \text{NewtonSchulz5}(b_t, \text{NS_steps})$

Apply spectral scaling factor for consistent update magnitude

$b_t \leftarrow b_t \cdot \sqrt{\max(1, \text{size}(b_t, -2) / \text{size}(b_t, -1))}$

$\theta_t \leftarrow \theta_{t-1} - \gamma \cdot b_t$ # Update parameters

Return θ_t

What is Newton-Schulz orthogonalization

1. Needs to reconstruct the momentum buffer
2. $O(n^3)$ complexity
3. Host dispatch overhead

Newton-Schulz is an **iterative** method based on matrix **polynomials** $f(x) = ax + bx^3 + cx^5 + \dots$

$$X_{t+1} = a_t X_t + b_t X_t X_t^\top X_t + c_t (X_t X_t^\top)^2 X_t, \quad t=0, \dots, 4$$

```
def newtonschulz5(G, steps=5, eps=1e-7):
```

```
    assert G.ndim == 2
```

```
    a, b, c = (3.4445, -4.7750, 2.0315)
```

```
    X = G.bfloat16()
```

```
    X /= (X.norm() + eps)
```

```
    if G.size(0) > G.size(1):
```

```
        X = X.T
```

```
    for _ in range(steps):
```

```
        A = X @ X.T
```

```
        B = b * A + c * A @ A
```

```
        X = a * X + B @ X
```

```
    if G.size(0) > G.size(1):
```

```
        X = X.T
```

```
    return X
```



Why Muon?

PROS

Saves you time and memory

Efficiency Advantages

- Single optimizer state (momentum buffer)
- Highly stable convergence in native bfloat16 (bf16)
- Fast converge by *optimizer* steps

Challenges of Muon

CONS

Needs lot of care to use

Muon Suitability Constraints

- Limited to specific training stages
- Restricted to compatible parameter types

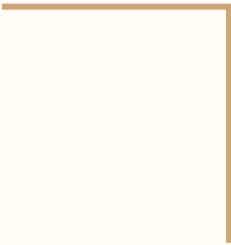
Scalability Challenges

- Super-linear complexity – $O(n^3)$
- Distributed training bottlenecks
- Host dispatch overhead

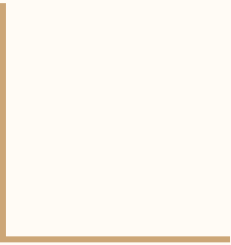
Integration and Operational Challenges

- My own integration saga





My Muon Variant



Developing Hopper

The Quest: Can you train GRPO style RL with Muon or its variant?

EXPERIMENTAL SETUP

Base Model

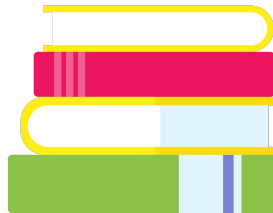
Qwen/Qwen2.5-0.5B-Instruct

Dataset

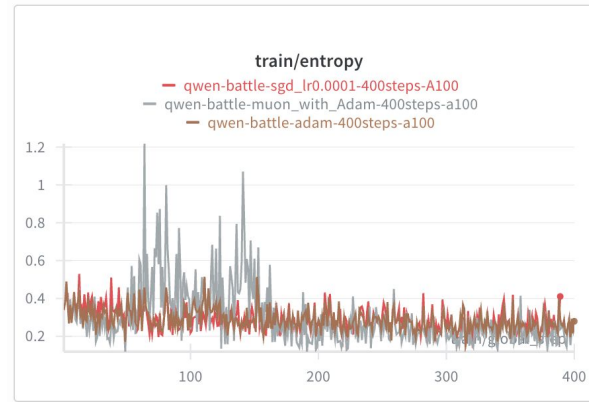
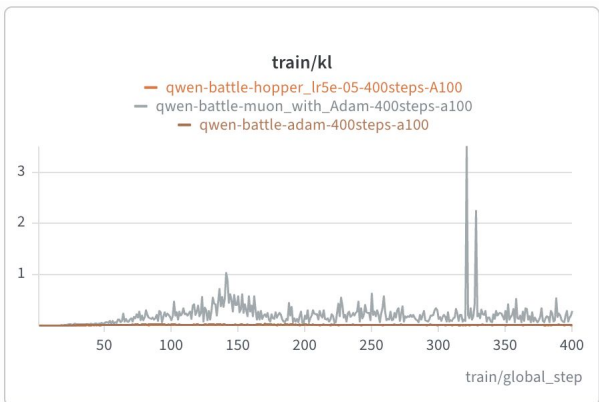
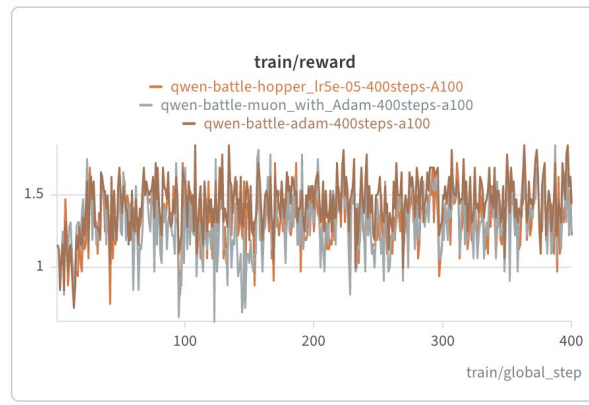
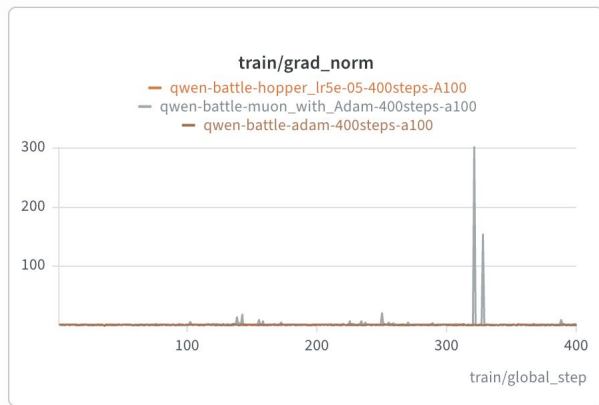
openai/gsm8k

HOPPER VARIANT ARCHITECTURE

- Added Adam's second moment for variance normalization
- Reduced Newton-Schulz iterations from 5 to 1 for lazy orthogonality



The Challenge with Vanilla Muon



Different optimizer.

Adam learned arithmetic faster. Hopper learned reasoning faster.

Q1: "When I was 6, my sister was half my age. Now I'm 70. How old is my sister?"

Adam (step 400):  Correctly calculated 67

Hopper (step 200):  Made arithmetic error (got 3)

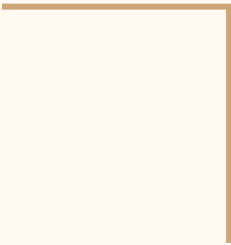
Q2: "3 towels dry in 1 hour. How long does it take to dry 6 towels?"

Adam (step 400):  Answered "2 hours" (linear scaling)

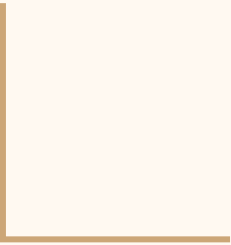
Hopper (step 200):  Answered "1 hour" (understood parallelism!)

Optimizer	Hard Eval Score	Pass Rate	Primary Failure Mode	The Vibe
			Convergence Failure	
SGD	1 / 5	20%	metrics looked good, but outputs were incoherent.	"The Imposter" 🤡
			Hallucination	
Standard Muon	2 - 3 / 5	40% - 60%	"Mutant Chickens" (logical absurdities to satisfy geometric constraints).	"The Chaotic Artist" 🎨
			Linear Heuristics	
Adam (Baseline)	4 / 5	80%	Failed the "Towel Question" (\$6 \to 2\$ hrs) due to linear bias.	"The Strict Accountant" 🧮
			Arithmetic Clumsiness	
Hopper (Ours)	4 / 5	80%	Solved the reasoning, failed the subtraction (\$70 - 3\$).	"The Absent-Minded Genius" 🧠





Overcoming Muon's Scalability Challenges



Super-linear time complexity - $O(n^3)$ for orthogonalization

Dion3 Update Rule

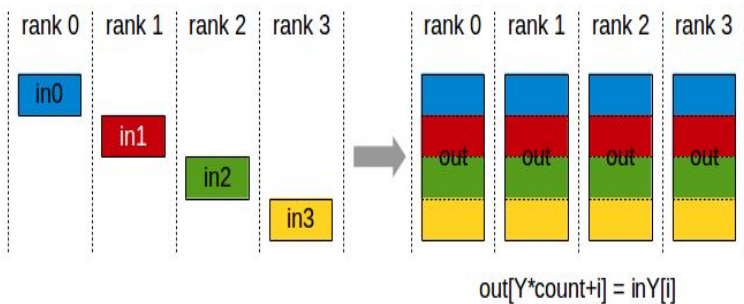
1. **Select** a fraction of the rows (or columns) of the momentum matrix (based on l_1 norm).
2. **Orthogonalize** the selected submatrix using **Gram Newton-Schulz**. Coupled with Symmetric GEMM Kernels in CuteDSL.
3. **Update the selected rows** of the weight matrix using the output of the previous step. Do not update the other rows.
4. **Error Feedback** *Decay* only the selected rows of the *momentum matrix* by a multiplicative factor.



Distributed Training Bottlenecks

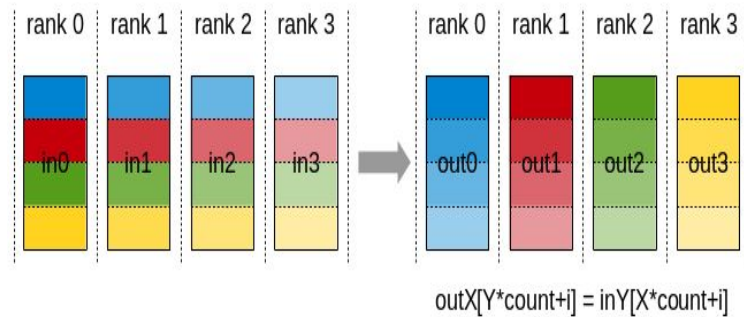
Moonshot AI

coalescing the all_gather operations



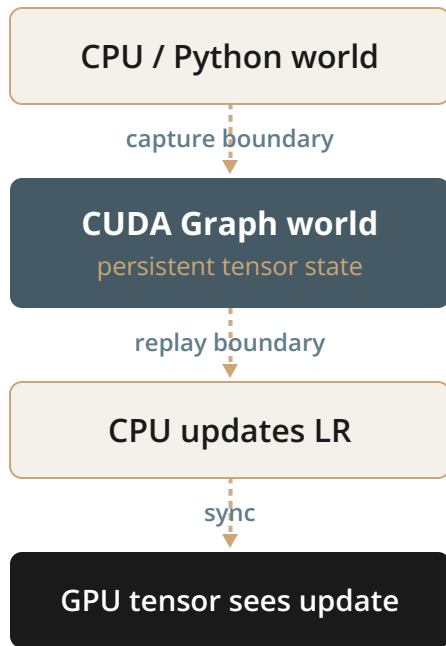
Dion3

Megabatching custom all-to-all

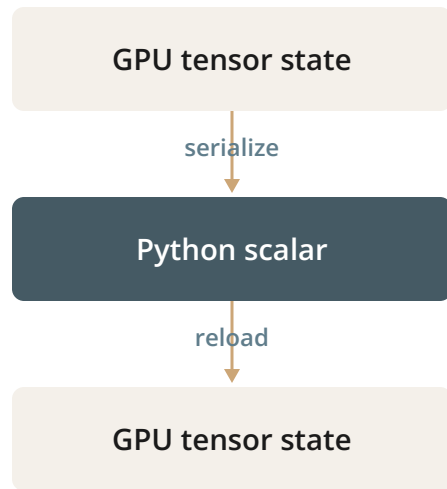


Optimization libraries
like Dion are no longer
just mathematical
algorithms; they are
co-compilers.

RUNTIME EXECUTION FLOW



CHECKPOINT BOUNDARY



Training

01 / EFFICIENCY

Mixed Precision

02 / PERSISTENCE

Checkpoint Restore

03 / RESILIENCE

Distributed Recovery

Experiment Setting

Trainer Setting

Distributed / Precision

```
compile_model = True
```

```
dp config:
```

```
name = hsdp
```

```
param = bfloat16
```

```
reduce = bfloat16
```

```
wrapping = blocks
```

```
float8_enabled = True
```

```
scheduler = CosWithWarmup
```

Optimizer Setting

Hyperparameters

```
BASE_LR
```

```
1e-3
```

```
BASE_WD
```

```
0.01
```

```
DION3_FRACTION
```

```
0.25
```

```
ADJUST_LR
```

```
"rms_norm"
```

Data Setting

Sequence & Batching

```
SEQUENCE_LENGTH
```

```
2048
```

```
GLOBAL_BATCH_SIZE
```

```
32 × SEQ_LEN
```

```
(65,536 tokens)
```

```
RANK_MICROBATCH_SIZE
```

```
4 × SEQ_LEN
```

```
(8,192 tokens)
```



Define `group_overrides`

GROUP 0

`Muon/Dion3`

2D weights

- Attention

`w_q, w_k, w_v, w_out`

- Feed_forward

`w1, w2, w3`

GROUP 1

`AdamW`

Layer Norm

- Attention_norm

`Q_norm, k_norm`

- Feed_forward-norm

- Im_head.norm

GROUP 2

`AdamW`

Embedding Layer

GROUP 3

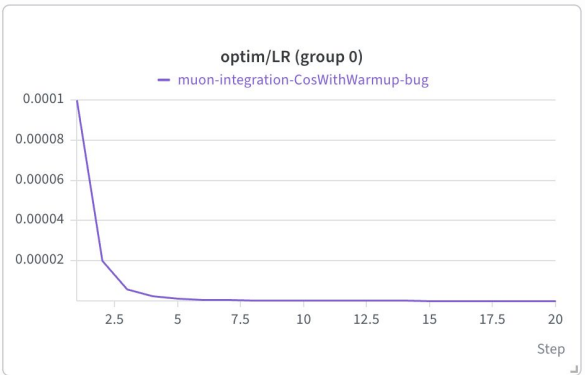
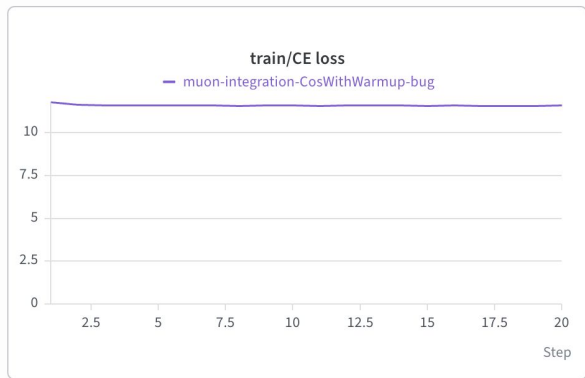
`AdamW`

Im_head

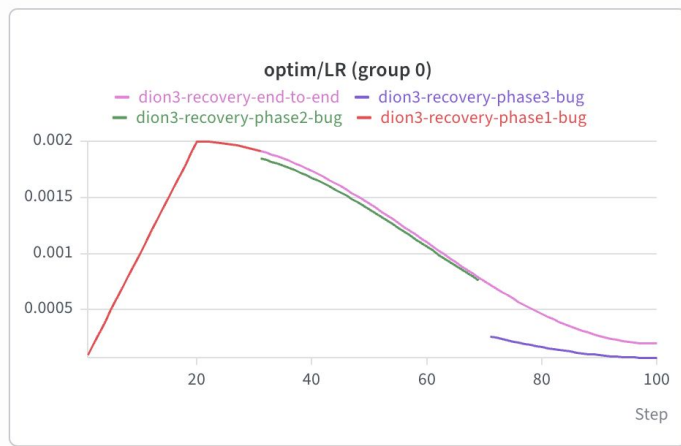


Debugging Saga

Learning Rate Death Spiral 🤪



Checkpoint/Distributed Failure Resumption 🤔



The Fix

Preventing learning rate decay corruption and ensure checkpoint recovery

src/olmo_core/optim/config.py

⚠ The Root Cause

```
# build function in config.py
if lr is not None:
    if self.compile:
        group[LR_FIELD] = move_to_device(
            torch.tensor(lr), self.device)
    else:
        group[LR_FIELD] = lr

    group.setdefault(INITIAL_LR_FIELD, lr) 💣🧑
```

Issue: initial_lr aliased the live tensor, causing decay schedule to compound across checkpoint resumes.

src/olmo_core/optim/config.py (resolved)

✅ The Fix

```
# Cast live tensor to immutable float
group.setdefault(INITIAL_LR_FIELD, float(lr))
```

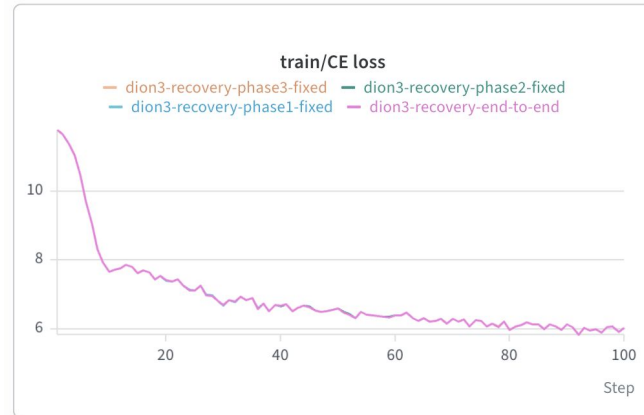
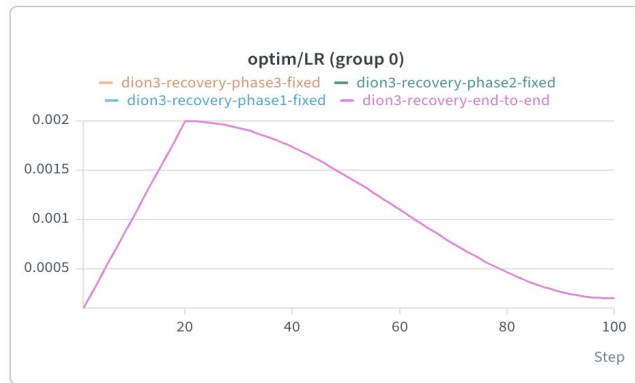
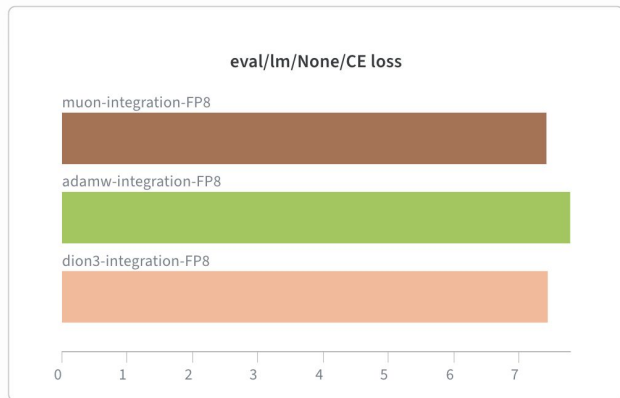
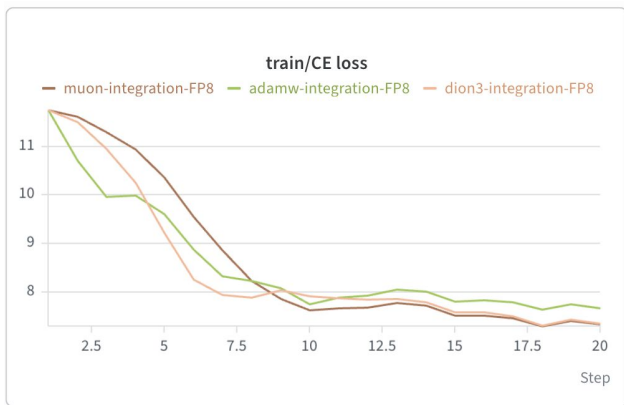
Why this solves it:

- Explicitly casting `float(lr)` breaks the tensor pointer reference.
- Ensures `INITIAL_LR_FIELD` stores a pure scalar value instead of mutating live state.
- Restores proper learning rate schedules across checkpoint restarts.



Post-Fix Results: Both Issues Resolved

Left: Train & eval loss trajectories as expected | Right: Resumed runs trace end-to-end seamlessly



Integration Challenges

Switching optimizers exposes hidden infrastructure assumptions—and when they break, they fail silently.

FAIL #01

Dead Config Field

`NorMuonConfig`

Missing `@dataclass` caused `muon_beta2` to be silently ignored.

YAML overrides fell back silently to default values.

Broken Assumption:

"Config fields get picked up automatically."

FAIL #02

Formula Leaked

`1/√d_model scaler`

LM-head scaling rule designed specifically for Lion was applied to AdamW.

Calculations continued without explicit failure flags.

Broken Assumption:

"Scaling rules are optimizer-agnostic."

FAIL #03

Distributed Sync

`synchronize_for_checkpoint()`

Dion requires explicit sync under replicated gradient sync.

AI2 guard shows Dion+HSDP breaking on newer PyTorch versions.

Broken Assumption:

"Gradient sync & checkpointing behave identically across torch versions."



References

Keller Jordan et al. (2024)

[Muon: An optimizer for hidden layers in neural networks](#)

Jeremy Bernstein (2024)

[Newton-Schulz](#)

Jingyuan Liu et al. (2025)

[Muon is Scalable for LLM Training](#)

Kwangjun Ahn, et al. (2025)

[Dion: Distributed Orthonormalized Updates](#)

Kwangjun Ahn, Austin Feng, et al. (2026)

[Dion3: Full-stack orthogonal updates](#)

Jen Wei (2026)

[AI2 Dion3-integration PR \(2026\)](#)

Jen Wei (2026)

[Hopper: The Optimizer That Learns Parallelism 2x Faster Than Adam](#)





Let's Collaborate & Scale Up

Jen Wei | PyTorch Con 2026

ONLINE PROFILE



Bird Of Paradise AI

birdofparadise.ai

GET IN TOUCH / CONNECT

Email

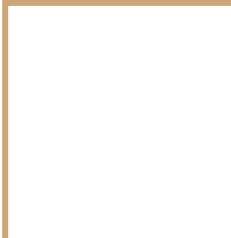
jen@birdofparadise.ai

LinkedIn

in/jenweiprofile/

X / Twitter

@JenniferWe17599



Appendix



Newton-Schulz orthogonalization is
only applicable to 2D matrix

✓ Most `nn.Linear` layers

✗ Biases in `nn.Linear`

✗ Normalization layers (e.g. `nn.LayerNorm`, `nn.RMSNorm`)

✗ Embedding layer `nn.Embedding`

✗ Unembedding layers Im head, or classification head

✗ 🙅: Muon will run without error, but will give poor performance.

✓ 3D → Flatten into 2D, ⌚ Pending on framework support

Params with 3+ dimensions, such as conv layers

✓ → Flatten into 2D

⌚ → Pending on framework support

```
if dim(b_t) ≥ 2:

    # Save original shape
    original_shape ← shape(b_t)

    # Reshape to 2D if needed
    if dim(b_t) > 2:
        b_t ← reshape(b_t, [size(b_t, 0), -1])

    b_t ← NewtonSchulz5(b_t, NS_steps)

    b_t ← b_t · √max(1, size(b_t, -2)/size(b_t, -1))

    # Restore original shape if needed
    if dim(original_shape) > 2:
        b_t ← reshape(b_t, original_shape)
```

Minefield of what can go wrong



Embedding layer

Embedding layer and lm head
should use AdamW or Lion



Adjust lr

rms_norm:
 $0.2 \cdot \sqrt{\max(A, B)}$

spectral_norm:
 $\sqrt{\max(1, A/B)}$



Initial lr for Dion3

$lr = \text{BASE_LR} /$
 $\text{math.sqrt(DION3_FRACTION)}$



Lr DataType

My own debugging saga 🤔



What is Muon – Momentum Orthogonalized by Newton-Schulz?

Full rank updates
? changes learned geometry

“Steepest Descent Under Norm Constraints Bernstein et al. 2024 proposed to view the optimization process in deep learning as steepest descent under norm constraints. From this perspective, we can view the difference between Muon and Adam (Kingma et al. 2015; Loshchilov et al. 2019) as the difference in norm constraints. Whereas Adam is a steepest descent under the a norm constraint dynamically adjusted from a Max-of-Max norm, Muon offers a norm constraint that lies in a static range of Schatten- p norm for some large p (Franz 2024). When equation 1 is accurately computed, the norm constraint offered by Muon will be the spectral norm. Weights of neural networks are used as operators on the input space or the hidden space, which are usually (locally) Euclidean (Cesista 2024), so the norm constraint on weights should be an induced operator norm (or spectral norm for weight matrices). In this sense, the norm constraint offered by Muon is more reasonable than that offered by Adam.”

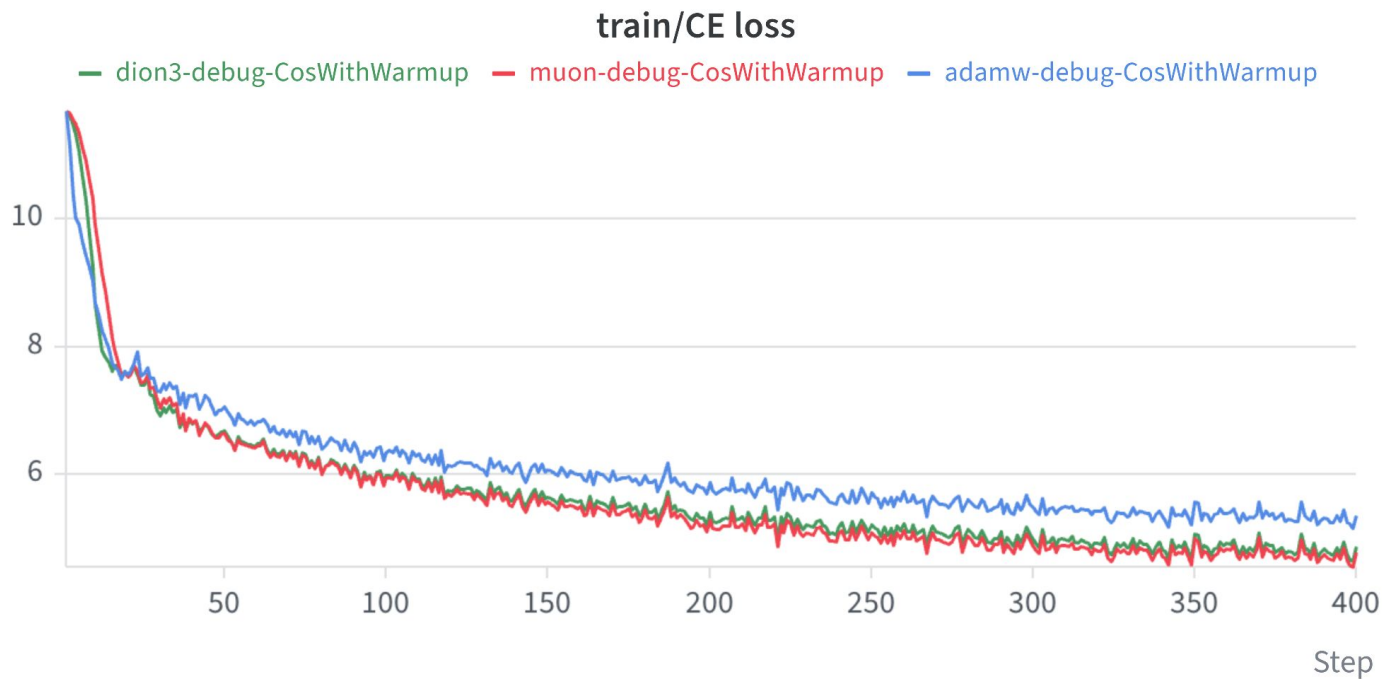
- moonshot

Distributed Training Bottlenecks

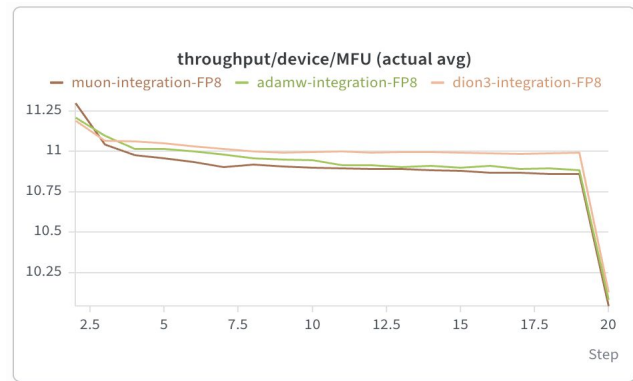
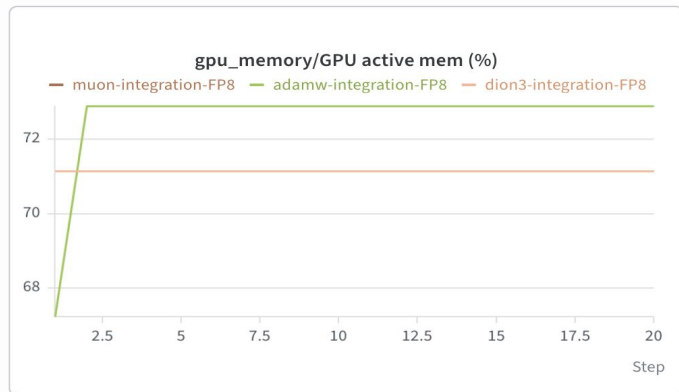
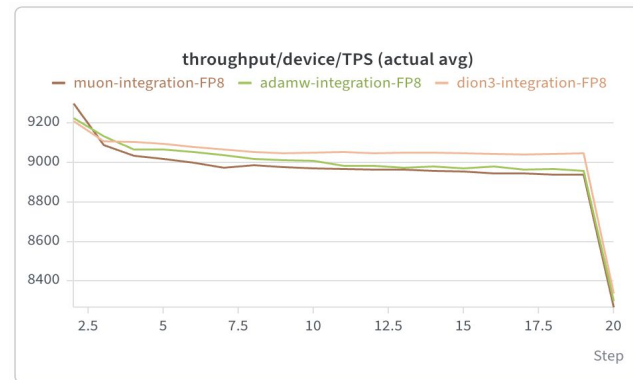
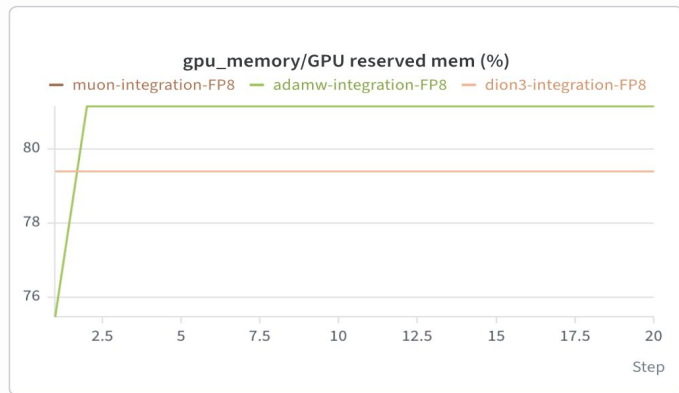
Moonshot AI (Moonlight)

```
1: // Reduce-scatter G on DP for correct gradients
2: g= reduce_scatter(G, dp_group)
3: // Apply momentum to g using local partitioned
   momentum m
4: b = update_with_momentum(g,m, $\mu$ )
5: // DP Gather: gathering b across DP into a full
   matrix B
6: B = gather(b, dp_group)
7: // Calculate Muon update
8: U = Newton-Schulz(B)
9: // Discard the rest of U and only keep the local
   partition u, then apply the update rule
10:  $p' = \text{apply\_update}(p,u)$ 
11: // All-gather updated  $p'$  into P
12: P= all_gather( $p'$ , dp_group)
```

Train loss for bf16 – Muon, Dion3, vs AdamW



Memory and Throughput- Muon, Dion3, vs AdamW



Super-linear complexity: $O(n^3)$ time for orthogonalization

Dion3

Gram Newton-Schulz

Symmetric GEMM Kernels in CuteDSL

The Dion3 Update Rule



Dion3 Update Rule

1. **Select** a fraction of the rows (or columns) of the momentum matrix.
2. **Orthogonalize** the selected submatrix using **Gram Newton-Schulz**.
3. **Update the selected rows** of the weight matrix using the output of the previous step. Do not update the other rows.
4. **Error Feedback** Decay only the selected rows of the momentum matrix by a multiplicative factor.





Benny  @benny_token360 · Sep 18



Replying to @PyTorch and @JenniferWe17599

Fast convergence demos are easy; surviving a real training stack is the hard part. Looking forward to the Muon/Dion/Dion3 walkthrough — production constraints over paper numbers.



1



25



lukaddd  @LukaDDD077 · Sep 19



Replying to @PyTorch and @JenniferWe17599

The production bar is not the benchmark chart: run Muon/Dion through mixed precision, checkpoint restore, and distributed failure recovery. If the convergence win disappears there, it shouldn't ship as the default.



1



31



Experiment Setup (01mo3_600M)

Trainer Config

```
compile_model=True,  
dp_config=TransformerDataParallelConfig(  
    name=DataParallelType.hsdp,  
    param_dtype=DType.bfloat16,  
    reduce_dtype=DType.bfloat16,  
    wrapping_strategy=  
        WrappingStrategy.blocks,  
    ),  
float8_config=Float8Config(enabled=True),  
z_loss_multiplier=1e-5,  
max_grad_norm=1.0,  
scheduler=CosWithWarmup
```

Optimizer Hyper Params

BASE_LR

1e-3

BASE_WD

0.01

DION3_FRACTION

0.25

ADJUST_LR

"rms_norm"

Data

SEQUENCE_LENGTH

2048

GLOBAL_BATCH_SIZE

32 * SEQ_LEN

RANK_MICROBATCH_SIZE

4 * SEQ_LEN

Sequence size per card