Fantastic Pretraining Optimizers and Where to Find Them

Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang

Sep 2025

Stanford University

Outline

- **Problem**: Why AdamW dominates pretraining until recently despite numerous claims of 1.4 to 2× speedup from alternatives?
- **TL;DR**: With fair tuning across multiple scales, fixing data budget:
  - Scalar-based optimizers (Nesterov AdamW, Mars, Cautious) reach no larger than 1.1× speedup over AdamW.
  - Matrix-based optimizers (Muon, Soap, Kron, Scion) can reach up to 1.4× speedup over AdamW on small models (<0.5B) but the speedup decreases to 1.1× at 1.2B.

Background: AdamW

AdamW was the default optimizer for almost all deep learning models until recently. It operates at a **scalar level**, updating each scalar in parallel using the historical gradient of that scalar.

Algorithm 1 AdamW

State: $m,\ v$

$$
\begin{aligned}
m_t &= \beta_1\, m_{t-1} + (1-\beta_1)\, g_t, \\
v_t &= \beta_2\, v_{t-1} + (1-\beta_2)\, g_t^2, \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \\
w_{t+1} &= w_t - \eta\, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \;-\; \eta\, \lambda\, w_t.
\end{aligned}
$$

Background: AdamW Variants

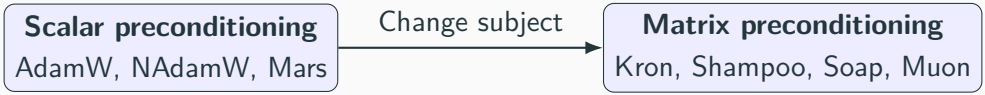
Despite the empirical success of AdamW, it is not ideal theoretically. People try to improve it by incorporating insights from acceleration techniques in convex optimization, such as variance reduction technique including **Nesterov Momentum**.

Algorithm 2 Nesterov AdamW (NAdamW)

$$
\begin{aligned}
m_t &= \beta_1 \, m_{t-1} + (1-\beta_1) \, g_t, \\
v_t &= \beta_2 \, v_{t-1} + (1-\beta_2) \, g_t^2, \\
\tilde{m}_t &= \beta_1 \, m_t + (1-\beta_1) \, g_t, \\
\hat{m}_t &= \frac{\tilde{m}_t}{1-\beta_1^{t+1}}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \\
w_{t+1} &= w_t - \eta \, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \; - \; \eta \, \lambda \, w_t.
\end{aligned}
$$

Background: Matrix vs scalar preconditioning

A much more aggressive approach is to **change the subject of optimization** from the scalar level to the matrix level.



Background: Shampoo

Shampoo is among the first opimizers which assume the object W_t is a matrix instead of a scalar. However, empirically Shampoo is not a very stable optimizer and requires careful tuning, high precision, and tricks including grafting to fully work.

Algorithm 3 Shampoo

$$
\begin{aligned}
L_t &= \beta_2\, L_{t-1} + (1-\beta_2)\, G_t G_t^T, \\
R_t &= \beta_2\, R_{t-1} + (1-\beta_2)\, G_t^T G_t, \\
\hat{L}_t &= \frac{L_t}{1-\beta_2^t}, \hat{R}_t = \frac{R_t}{1-\beta_2^t}, \\
W_{t+1} &= W_t - \eta(\hat{L}_t)^{-1/4} G_t (\hat{R}_t)^{-1/4} - \eta\,\lambda\, W_t.
\end{aligned}
$$

Background: Muon

The difficulty of using Shampoo is fully resolved by its successor Muon.

Algorithm 4 Muon

$$
\begin{aligned}
M_t &= \beta_1 M_{t-1} + (1-\beta_1) G_t, \\
\tilde{M}_t &= \beta_1 M_t + (1-\beta_1) G_t, \qquad \text{(Nesterov)} \\
W_{t+1} &= W_t - \eta \mathrm{NS}^{(5)}(\tilde{M}_t) - \eta \lambda W_t.
\end{aligned}
$$

Here $\mathrm{NS}(x) = x(ax + bx^T x + c(x^T x)^2)$ satisfies that

$$
\mathrm{NS}^{(5)}(M) \approx \arg \max_{\|O\|_{\mathrm{op}}=1} \mathrm{Tr}(O^T M). \tag{1}
$$

Motivation: How good are these optimizers?

While there are an abundance of optimizers claiming a huge speedup over AdamW, the difference in evaluation protocol lead to vastly different results about their effectiveness.

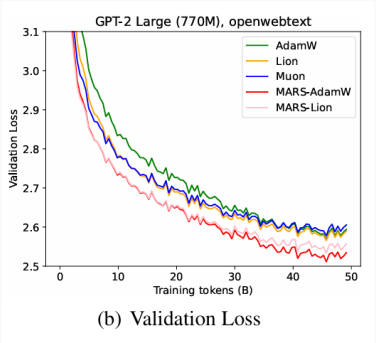


Figure 1: Figure 1, Mars

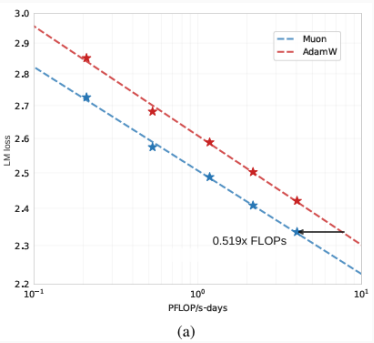
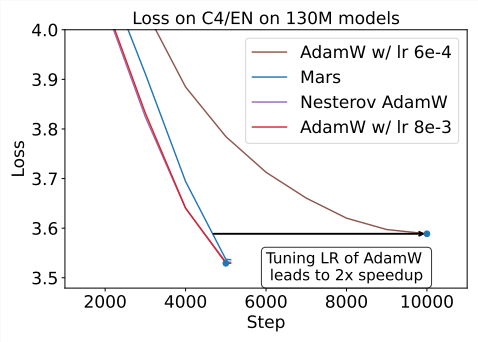


Figure 2: Figure 1, Moonlight

Key Reason: Optimizers are not evaluated in respectively best practices!

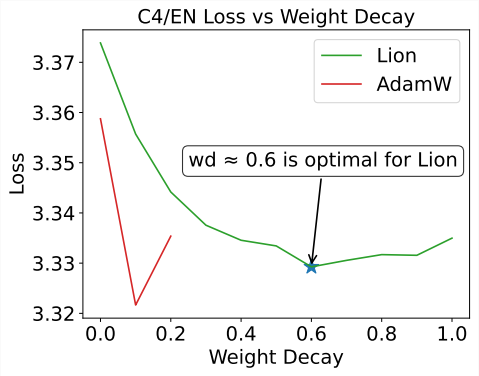
Motivation: Tuning learning rate alone leads to 2 $\times$ speedups

Works including Sophia, Mars, Cautious AdamW, e.t.c, claim a 2 $\times$ speedup over AdamW using the learning rate inherited from GPT-3 recipe. However, this learning rate is far from optimal and increasing the learning rate alone can lead to a 2 $\times$ speedup!



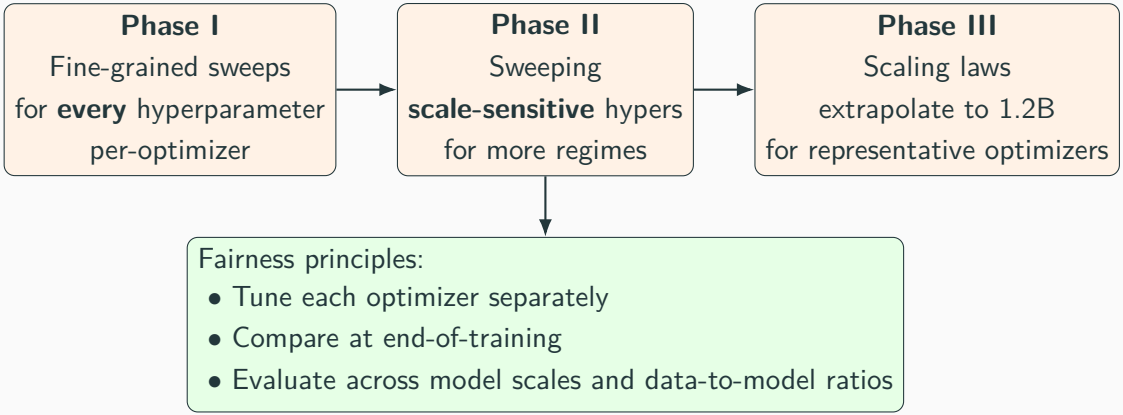
Motivation: Fixing hyperparameters $\neq$ fair comparison

Another common practice is to fix the shared hyperparameters across optimizers (learning rate, weight decay, e.t.c). However, even conceptually similar optimizers may correspond to very different optimal hyperparameters to achieve the best performance.



Methodology overview: Three phases

Let's redo everything in a *dumb* way!



Experimental Setup

- Models: **0.1B, 0.3B, 0.5B, 1.2B** parameters
- Data: **1–8**$\times$ Chinchilla-optimal tokens on DCLM-baseline, StarCoder V2 Data, and ProofPile 2
- Optimizers: AdamW, NAdamW, Lion, Mars, Muon, Soap, Kron, Scion, Cautious, Adam-Mini, Sophia
- Primary Evaluation: **End-of-training** C4/EN losses
- Secondary Evaluation: Downstream performance on HellaSwag, ARC, BoolQ, CommonsenseQA, PIQA, e.t.c
- Phase I: 0.1B 1-$8\times$ Chinchilla & (0.3B, 0.5B) $1\times$ Chinchilla
- Phase II: (0.3B, 0.5B) 2-$8\times$ Chinchilla
- Phase III: 1.2B 1-$8\times$ Chinchilla

Phase I: Coordinate Descent Sweeps

Stage	LR	WD	min lr	Warmup	Max Grad Norm	Batch	Val. Loss
Init	0.008	0.1	0	1000	1	256	3.298
Round 1	0.008	0.1	0	2000	1	256	3.282
Round 2	0.008	0.1	0	2000	1	128	3.263
Best	0.008	0.1	0	2000	2	128	3.263

Table 1: Illustrative coordinate-descent steps for AdamW on the 130M 1× Chinchilla regime. Changed hyperparameter values are highlighted in red; We omitted some unchanged hyperparameters ($\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-10}$).

Phase I: Coordinate Descent Sweeps Results

Once converged, we obtain a table similar to the following

Table 2: Hyperparameter ablation for AdamW on 130m on 1x Chinchilla Data

β_1	β_2	ϵ	η	g_{norm}	η_{min}	BSZ	warmup	λ	Loss	Link
0.9	0.98	1e-20	0.008	1	0	128	2000	0.1	3.529	0
0.95	–	–	–	–	–	–	–	–	3.539	1
0.98	–	–	–	–	–	–	–	–	3.882	2
–	0.9	–	–	–	–	–	–	–	3.545	3
–	0.95	–	–	–	–	–	–	–	3.535	4
–	–	1e-25	–	–	–	–	–	–	3.529	5
–	–	1e-15	–	–	–	–	–	–	3.531	6
–	–	1e-10	–	–	–	–	–	–	3.531	7
–	–	–	0.004	–	–	–	–	–	3.550	8
–	–	–	0.016	–	–	–	–	–	3.538	9
–	–	–	0.032	–	–	–	–	–	7.781	10
–	–	–	–	0	–	–	–	–	3.534	11
–	–	–	–	2.0	–	–	–	–	3.534	12
–	–	–	–	–	–	256	–	–	3.611	13
–	–	–	–	–	–	–	500	–	7.452	14
–	–	–	–	–	–	–	1000	–	3.532	15
–	–	–	–	–	–	–	4000	–	3.575	16
–	–	–	–	–	–	–	–	0	3.545	17
–	–	–	–	–	–	–	–	0.2	3.536	18

Phase I: Coordinate Descent Sweeps Results

- Settings swept: 130M, 300M, 500M (at 1× Chinchilla) and 130M (at 2, 4, 8×).
- Outcome: For each optimizer and regime, identified a coordinate-wise local optimum over all hyperparameters.
- Observation 1: Loss is sensitive only to a subset of hyperparameters; many have negligible effect when perturbed near optimum.
- Observation 2: Among the sensitive ones, most optimal values are stable across scales.
- Takeaway: Enables narrowing later sweeps to only **scaling-sensitive** hyperparameters.

Phase II: Further Sweeping on Scale-Sensitive Hyperparameters

We retain only scaling-sensitive hyperparameters for Phase II sweeps on additional regimes: 300M, 500M at 2, 4, 8× Chinchilla.

Optimizer	**Scaling-sensitive hyperparameters**
AdamW	learning rate, warmup, weight decay, batch size
NAdamW	learning rate, warmup
Muon	learning rate
Soap	learning rate, warmup, block size

Table 3: Scaling-sensitive hyperparameters carried into Phase II.

Phase II: Results on models ≤ 0.5B

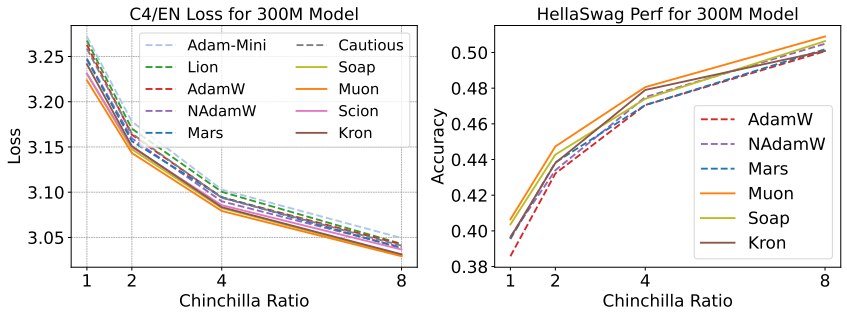


Figure 3: Loss and Benchmark Performance of different optimizers across scale.

Phase II: Results on models ≤ 0.5B

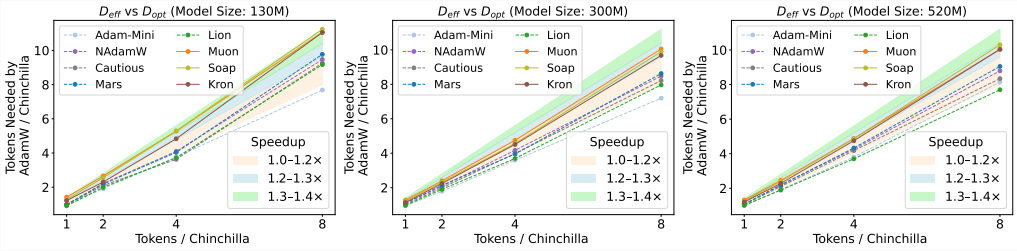


Figure 4: Speedup of different optimizers across scale.

- Matrix-based optimizers (**Muon, Soap, Kron, Scion**) consistently outperform scalar-based counterparts. Speedups over AdamW typically at **1.2–1.4**×.
- Scalar variants (**NAdamW, Mars, Cautious**) are within $1.1\times$ AdamW after fair tuning. The lift is small yet consistent.
- **Lion** and **Adam-mini** show different trends w.r.t model sizes.

Phase III: Extrapolation

- Fit hyperparameter scaling laws for each scaling-sensitive hyperparameter h:

$$h(N, D) = \alpha N^{-A} D^{-B} + \beta.$$

- Parameters learned via non-linear least squares on 12 observed (N, D, h) triples per optimizer.
- Validation: For AdamW at 1.2B and 1× Chinchilla, predicted hyperparameters are within 3×10^{-3} final loss of the full-sweep optimum.
- Case studies:
  - 1.2B models with AdamW, NAdamW, Muon, Soap at 1–8× Chinchilla.
  - 130M/300M at 16× Chinchilla with AdamW, NAdamW, Muon, Soap.

Phase III: Results on Model Scaling

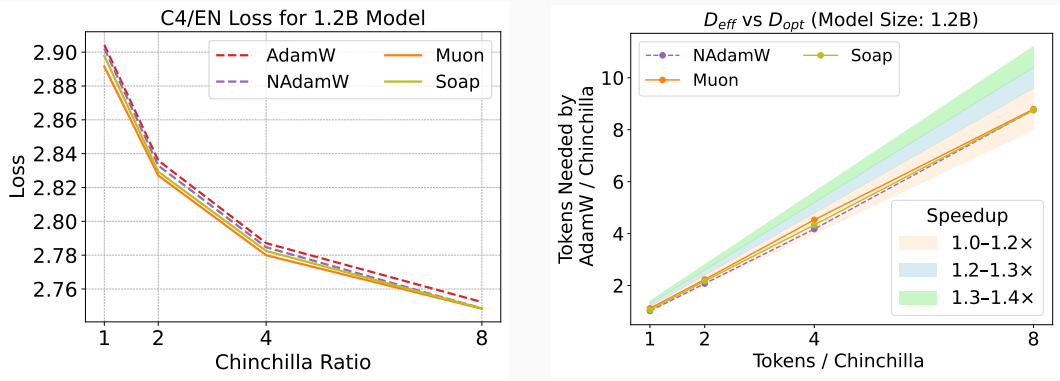
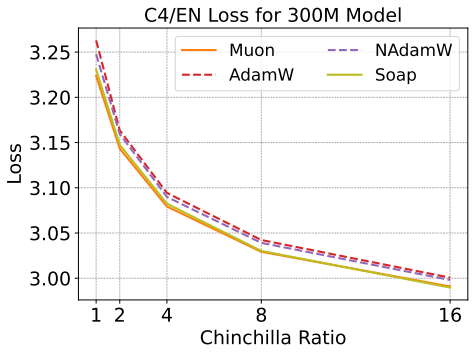


Figure 5: Case Studies for 1.2B. Left: Validation loss scaling on 1.2B model for AdamW, NAdamW, Muon and Soap. Muon and Soap still offer significant speedup over AdamW but no longer significantly speed up over NAdamW. Right: Estimated speedup ratio, we observe that Muon and Soap's speedup decays with model size to only $1.1\times$.

Phase III: Results on Data Scaling

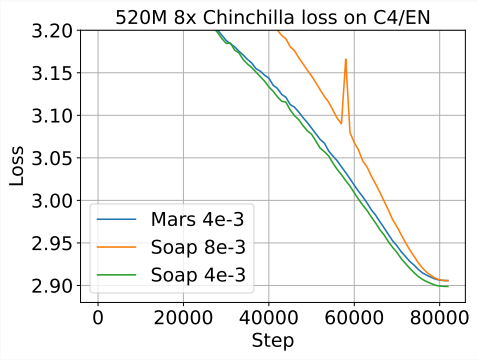
Muon is outperformed by Soap in the 8× Chinchilla regime for the 130M and 520M models and 16 × Chinchilla regime for 300M models.

Conjecture: The second-order momentum maintained by Soap and Kron becomes more effective when the data-to-model ratio increases. In the long run, adaptivity to heterogeneity in parameter directions may lead to a larger speedup.



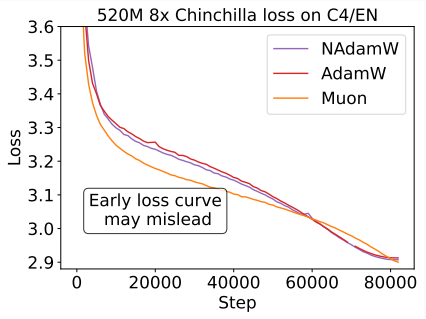
Necessity of Hyperparameter Tuning

Even a superior optimizer can underperform a less advanced method when its hyperparameters are not precisely tuned. In our exhaustive grid searches, slight deviations from each optimizer's ideal learning rate or other critical hyperparameter often lead to degradation in validation loss that is large enough to flip the ordering.



The early training behavior can be misleading

Validation-loss curves during this initial phase tend to exaggerate performance gaps and, in some cases, even reverse the eventual ranking.



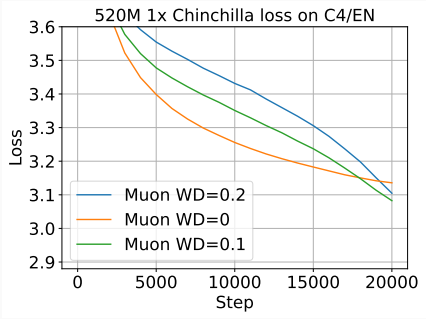


Figure 6: Necessity of Careful Tuning. Left: The order of optimizers may flip arbitrarily if rigorous tuning is missing. Right: Changing a single hyperparameter like weight decay may lead to misleading faster loss improvement but plateaus later.

Common Phenomenon Across Optimizers

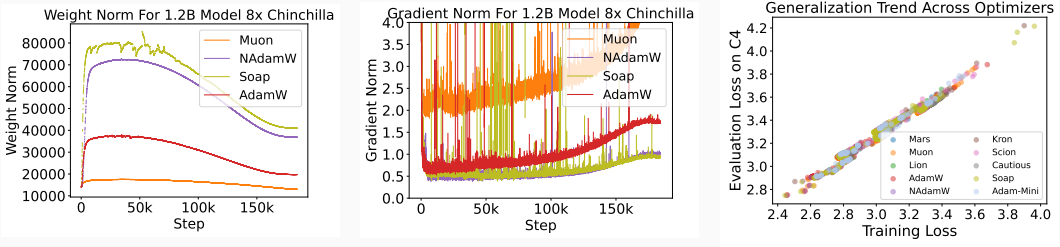


Figure 7: Common Phenomena Across Optimizers. Left: Parameter norm of all optimizers show a similar trend of increment and decrease, closely aligning the increasing and decaying of learning rate schedule. Middle: Gradient norm increases during learning rate decay. However, this increase does not lead to a loss increase. Right: The training loss and evaluation loss follows the same trend for all optimizers.

Key takeaways

- With fair tuning, many claimed 1.4–2$\times$ speedups shrink to **1.1–1.3**$\times$.
- **Matrix-preconditioned** optimizers are best-in-class, but gains decay with size. The difference in update rule matters in small Chinchilla regime but diminishes with more data.
- **Variance-reduction** techniques provide a small but consistent $1.1\times$ speedup, and there is no significant difference between the concrete way to perform variance reduction.
- Always compare **at target budget**, across **scales** and **data ratios**.

Reflections After The Paper Releases

Thanks for helpful discussions over X. We have some more reflections on our study.

Our objective: Find the best hyperparameters that reaches lowest loss in one pass of a fixed amount of data (this includes the batch size).

In practice: Wall time matters, and it crucially depends on the MFUs. Especially, batch size positively correlates with MFUs (Marin issue).