When Classical Optimization Meets Modern Foundation Models: New Algorithms, Theory, and Insights

Mingyi Hong

Department of Electrical and Computer Engineering,

University of Minnesota, Minneapolis

Amazon Scholar

UNIVERSITY OF MINNESOTA

Driven to Discover®

Northwestern IEMS, May 21, 2026

Collaborators

Athanasios Glentis	**Chung-Yiu Yau**	**Jiaxiang Li**	**Andi Han**
UMN PhD	UMN Postdoc	Meta/Joining VT	University of Sydney
Dawei Li	**Valentyn Boreiko**	**Hoi-To Wai**	
UMN Postdoc	Amazon AGI	CUHK	

Support from NSF and JPMorgan Chase Faculty Research Award

Related Works (This Talk)

Thread I — making SGD work for LLM pre-training:

- A. Glentis, C.-Y. Yau, D. Li, M. Hong, "What stops SGD on LLM pre-training: the need for a large learning rate and how to achieve it", 2026 (under review).
- A. Glentis, J. Li, A. Han, M. Hong, "SCALE: a minimalist memory-efficient optimizer for LLM pre-training", **ICML** 2026.

Thread II — making lookahead work for LLM pre-training:

- C.-Y. Yau, D. Li*, A. Glentis*, V. Boreiko, H.-T. Wai, M. Hong, "EMA-Nesterov: stabilizing Nesterov's lookahead for accelerated deep learning optimization", 2026 (under review).

Foundation-Model Pre-training as an Optimization Problem

- Pre-training a frontier LLM is (arguably) one of the most computational extensive tasks ever performed: it costs millions of GPU-hours, tens of TB of activations and optimizer state, $\sim 10^7$ \$ per run.
- Underneath: a nonconvex stochastic optimization problem

$$\min_{w \in \mathbb{R}^d} \; f(w) \;=\; \mathbb{E}_{\xi \sim \mathcal{D}}\big[\, \ell(w;\xi) \,\big],$$

  with $d \sim 10^9$–10^{12}.
- The optimizer is a fundamental design choice: every percent of speed-up / memory-saving is dollars of compute.

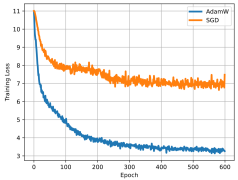
Optimizers for Deep Learning: A Brief History

- **Classical era (pre-2015).** SGD with Polyak momentum and Nesterov's accelerated gradient dominate; *Sutskever et al. '13* establish SGD-M as the deep-net default on ConvNets.
- **Adaptive era (2011–2018).** AdaGrad *(Duchi+ '11)*, RMSProp, Adam *(Kingma & Ba '15)*, AdamW *(Loshchilov & Hutter '19)* take over. Convergence concerns prompt AMSGrad *(Reddi+ '18)*; *Wilson+ '17* argue SGD generalizes better on vision.
- **Transformer / LLM era (2017+).** Adam becomes indispensable for transformers. LAMB *(You+ '20)* scales to large batches. New wave (2023–2025): Lion *(Chen+ '23)*, Muon *(Jordan+ '24)*, SOAP *(Vyas+ '24)*, Sophia *(Liu+ '24)*
- Transformer/LLM optimizers are all still heavily adaptive-flavored.

Two Folk Wisdoms

Folk wisdom 1 (Thread I). *"Plain SGD fails for LLM pre-training."*

- Multiple competing mechanistic explanations, none conclusive: ill-conditioned sharpness *(Pan & Li, '23)*; heavy-tailed class imbalance at the output layer *(Kunstner+ NeurIPS'24)*; Hessian block heterogeneity *(Zhang+ NeurIPS'24)*.



(Zhang+ NeurIPS'24) GPT-2

Folk wisdom 2 (Thread II). *"Nesterov's momentum is unstable on deep nets."*

- Lookahead is unstable with noisy gradient (Wang+ '20)
- Existing SOTA algorithms do not perform Nesterov's lookahead

Central Question

Question

Are classical optimization techniques such as SGD and Nesterov's acceleration obsolete in the foundation-model era?

Or can they be made competitive, and useful in large-scale training?

- This talk: **two case studies** answering "no, they are not obsolete".
- Each case study: diagnosis $\rightarrow$ minimalist algorithmic fix $\rightarrow$ convergence theory + LLM-scale experiments.

This Talk

- **Part I. Making SGD work for LLM pre-training**
  - Why does SGD so much worse compared with Adam?
  - We identify two structural irregularities in the LLM gradients
  - A simple fix, SGD-LL (SGD with Large LR via clipping) closes the gap to $\sim 3.5\%$.
  - Distill the insight into SCALE: a column-normalized, last-layer-momentum optimizer that are competitive with Adam while at ~ 35–45% in memory usage.
- **Part II. Making Nesterov's acceleration work**
  - Why does naive Nesterov lookahead not stable on deep-net?
  - EMA-Nesterov: an EMA-stabilized lookahead direction.
  - Accelerated convex rates preserved;
  - Consistent gains on a wide range of optimizers, Adam/SOAP/Muon up to LLaMA 1B.
- **Closing.** Exploit the structure of the models, and classical optimization still can contribute significantly to modern optimizer design.

Part I. Making SGD Work

Diagnosing the Adam–SGD Gap in LLM Pre-Training,
and two algorithms developed from that diagnosis:

- SGD-LL: high-LR clipped SGD recovers most of the Adam performance
- SCALE: minimalist memory-efficient optimizer based on SGD

Based on joint work with Athanasios Glentis, Chung-Yiu Yau, Dawei Li, Jiaxiang Li, Andi Han.

(Momentum) SGD vs. Adam

(Momentum) SGD

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \mathbf{m}^{(t)}, \quad \text{with} \quad \mathbf{m}^{(t)} = \beta \mathbf{m}^{(t-1)} + (1-\beta)\mathbf{g}^{(t)}.$$

Adam

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \frac{\mathbf{m}^{(t)}}{\sqrt{\mathbf{v}^{(t)}}+\epsilon} \quad \text{with} \quad \begin{aligned} \mathbf{m}^{(t)} &= \beta_1 \mathbf{m}^{(t-1)} + (1-\beta_1)\mathbf{g}^{(t)}, \\ \mathbf{v}^{(t)} &= \beta_2 \mathbf{v}^{(t-1)} + (1-\beta_2)\mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}. \end{aligned}$$

What Stops SGD on LLM Pre-Training?

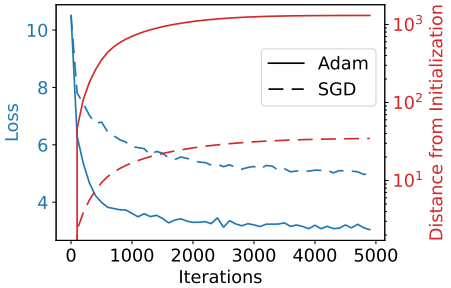
Plain (momentum) SGD fails in standard LLM pre-training. But why?

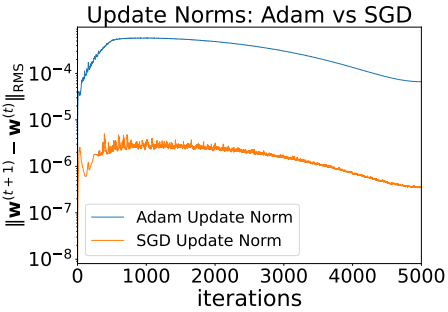
- **Many prior explanations.** Ill-conditioned sharpness *(Pan & Li '23)*; heavy-tailed class imbalance *(Kunstner+ NeurIPS'24)*; Hessian block heterogeneity *(Zhang+ NeurIPS'24)*; ...
- **Recent observation.** At very small batch sizes, SGD matches Adam *(Marek+ '25, Sreckovic+ '25)*.
- **Gap:** In the broader pre-training setup, what SGD is doing during training and why it can't keep up remain much less explored.

Our Observation

SGD only explores a small neighborhood around the initialization; Adam can leverage large effective learning rates to explore larger neighborhood.

Training Dynamics: SGD vs. Adam





LLaMA-130M, token batch size 0.5M. (Left) loss + distance from initialization. (Right) per-layer RMS update norm.

- **Left.** Effective solution space of SGD is $\sim 50\times$ smaller than Adam's; even though tuned SGD LR (0.5) is $\sim 166\times$ Adam's (3×10^{-3}).
- **Right.** Adam's per-layer update norm stays $\sim 100\times$ larger than SGD's.

Question. Why? $\Rightarrow$ Effective learning rates.

Effective Learning Rates (LRs): Adam vs. SGD

Nominal LRs:

Adam: $\eta_{\text{Adam}} = 3 \times 10^{-3}$

SGD: $\eta_{\text{SGD}} = 5 \times 10^{-1}$

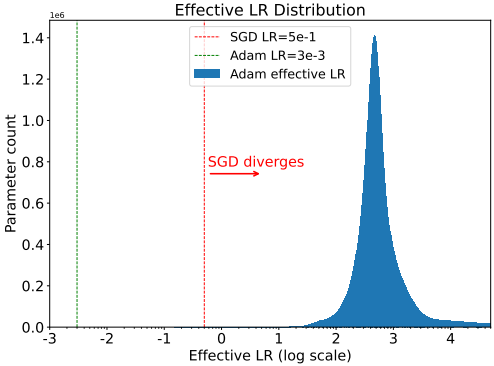
(both tuned; SGD diverges at higher LR)

Per-step updates:

$$\text{Adam: } \theta^{t+1} \leftarrow \theta^t - \frac{\eta_{\text{Adam}}}{\sqrt{v^t} + \epsilon} m^t$$

$$\text{SGD: } \theta^{t+1} \leftarrow \theta^t - \eta_{\text{SGD}} m^t$$

Observation. Adam's effective LRs are 2–4 orders larger than SGD's.

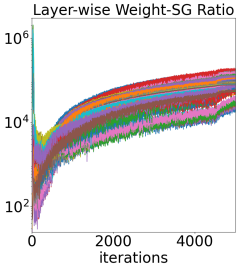


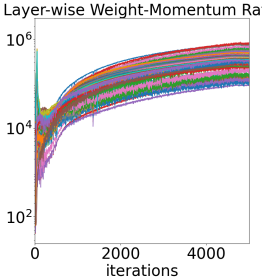
Per-layer effective learning rate distribution for Adam.

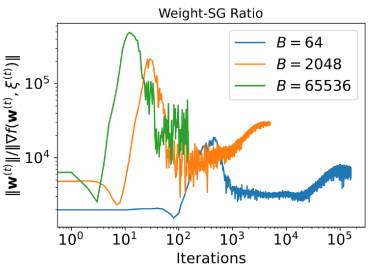
Question. Why such a large effective LR even needed? Why we cannot increase SGD's LR?

Why Large LR is Needed: Weight-to-Gradient Norm Ratio

Observation. In LLM pre-training the stochastic gradient (and momentum) are tiny compared to the weights; large effective LR is needed just to move; the weight/SG ratio gets worse with batch size.







Dynamics of $\|w_\ell^{(t)}\|_F / \|\nabla f(w^{(t)}; \xi^{(t)})_\ell\|_F$ and $\|w_\ell^{(t)}\|_F / \|m_\ell^{(t)}\|_F$ per layer during pre-training.

- Adam's preconditioner $1/(\sqrt{v^t} + \epsilon)$ inflates per-coordinate steps significantly.
- SGD lacks this knob; needs a large global LR, but can't get there without diverging.

Why is the Gradient So Small? A Decomposition

Question. Why is the gradient / momentum tiny in pre-training?

Theorem (Gradient Norm Upper Bound with Partition)

Given any partition $\mathcal{P}$ of token classes, the output-layer gradient norm satisfies

$$\|G\|_F \leq \mathcal{O}(1/\sqrt{B}) + \varepsilon_{\mathcal{P}},$$

where B is batch size and $\varepsilon_{\mathcal{P}}$ is a collective bound of token contributions dependent on how we partition the token classes into frequent and infrequent ones.

- Large batch $B \Rightarrow \mathcal{O}(1/\sqrt{B})$ covariance term is small.
- Choosing proper partition $\mathcal{P} \Rightarrow \varepsilon_{\mathcal{P}}$ is small.

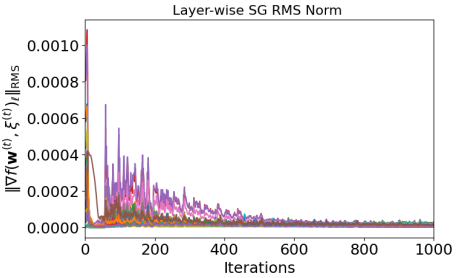
Conclusion: Gradient norm is structurally tiny, so large effective LR is unavoidable.

Why SGD Diverges at Large LR: Gradient Irregularities

Question. If we just crank up SGD's LR, why does it diverge?

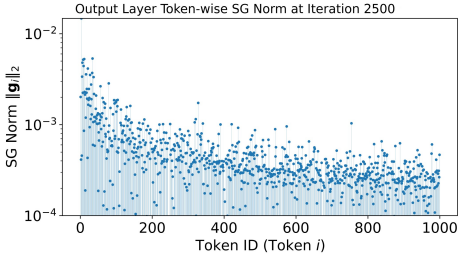
Answer. Two structural irregularities of the LLM stochastic gradient:

(Type I) Layer-wise SG spikes.



Layer-wise $\|\nabla f(w^{(t)}; \xi^{(t)})_\ell\|_{\mathrm{RMS}}$.

(Type II) Output-layer per-token-class imbalance.



Per-token-class column grad-norm $\|g_i\|_2$; lower token IDs dominate.

A small set of "loud" layers / tokens blows up at high LR before training can progress.

Enabling High-LR SGD via Clipping: SGD-LL

SGD-LL = SGD with **L**arge effective **L**earning rate, enabled by structure-aware clipping.

Clipping operator: $\mathrm{clip}_{r,\|\cdot\|}(x) := \min\left\{1, \frac{r}{\|x\|}\right\} x.$

- **Resolve Type I:** layer-wise RMS clipping for intermediate layers,

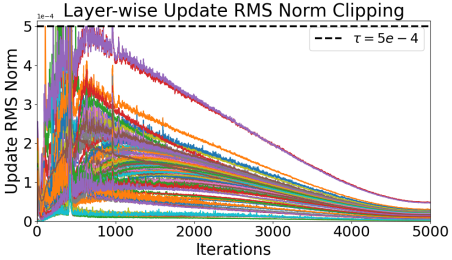
$$w_\ell^{(t+1)} = w_\ell^{(t)} - \mathrm{clip}_{\tau,\|\cdot\|_{\mathrm{RMS}}}\left(\eta_t\, m_\ell^{(t)}\right), \qquad \ell = 1, \ldots, L-1.$$

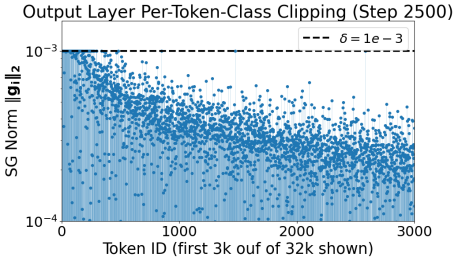
- **Resolve Type II:** per-token-class ℓ_2 clipping on top of layer clipping for output layer,

$$w_L^{(t+1)} = w_L^{(t)} - \mathrm{clip}_{\tau,\|\cdot\|_{\mathrm{RMS}}}\left(\eta_t\, \widetilde{m}_L^{(t)}\right), \quad \widetilde{m}_{L,i}^{(t)} = \mathrm{clip}_{\delta,\|\cdot\|_2}\left(m_{L,i}^{(t)}\right), \quad i = 1, \ldots, V.$$

$\Rightarrow$ allows tuned SGD LR up to $\eta \in [100, 300]$ (vs. vanilla SGD diverging at $\eta > 0.5$).

Illustration: Clipping is Targeted, Not Aggressive





LLaMA-130M. (Left) Type-I clipping: layer-wise RMS update norms; only the spiking layers are clipped. (Right) Type-II clipping: per-token-class norms; only the heavy-tail tokens are clipped.

- Clipping targets the two identified irregularities and leaves the vast majority of gradients untouched.
- This is structure-aware clipping, motivated by the diagnosis.

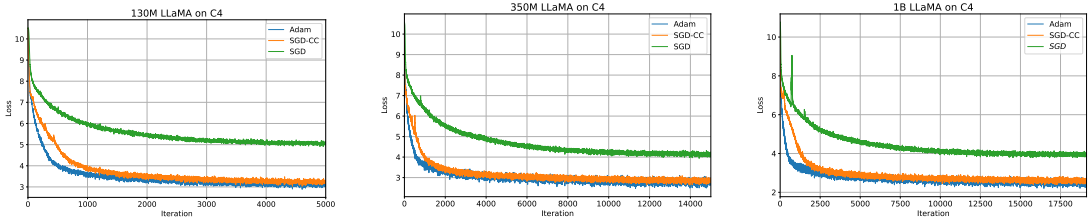
SGD-LL: Pre-Training Experiment Results

	130M (2.6B tok)				**350M** (7.8B tok)				**1B** (20B tok)			
	V. loss	PPL	LR	$\|w^T - w^0\|_F$	V. loss	PPL	LR	$\|w^T - w^0\|_F$	V. loss	PPL	LR	$\|w^T - w^0\|_F$
Adam	3.11	22.46	3×10^{-3}	1.3×10^{4}	2.78	16.09	2×10^{-3}	6.7×10^{5}	2.51	12.32	2×10^{-3}	7.2×10^{5}
SGD	5.07	158.90	0.5	3.4×10^{2}	4.14	63.22	0.5	5.2×10^{3}	3.95	51.87	0.5	6.2×10^{3}
SGD-LL	3.24	25.70	100	3.9×10^{3}	2.87	17.35	200	1.1×10^{5}	2.60	13.47	300	1.3×10^{5}

LLaMA on C4. $\tau = 5\times10^{-4}$ (130M, 350M), $\tau = 2\times10^{-4}$ (1B); $\delta = 10^{-3}$.

- SGD-LL admits LR $\in \{100, 200, 300\}$ across scales — 200–600$\times$ vanilla SGD.
- Adam-vs-SGD perplexity gap shrinks from $\sim$ 50% down to $\sim$ 3.5% (130M, 1B).

SGD-LL: Training Curves Track Adam



Training-loss evolution across LLaMA 130M / 350M / 1B; token batch size 0.5M–1M. SGD-LL trajectory closely tracks Adam.

Takeaway (Folk wisdom 1, revisited)

SGD's failure in LLM pre-training is not a property of the loss landscape, and not a fundamental limitation. It is a consequence of the structure of the LLM stochastic gradient (small global norm + a few loud directions). Respect that structure, and then SGD with a large LR can significantly close the gap.

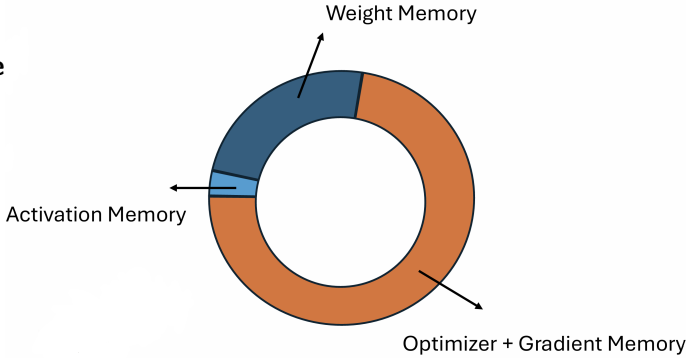
From Diagnosis to a Memory-Efficient Optimizer

Goal. Turn the diagnosis into a practical, memory-efficient optimizer for pre-training.

- Memory bottleneck (LLaMA-7B example, BF16, single batch):
  - Parameters 14 GB + Gradient 14 GB + Activations 2 GB
  - Adam states (m, v): 28 GB $\Rightarrow$ Total 58 GB $>$ A100 40 GB.

LLaMA-7B memory estimate *(Zhao+ ICML'24)*

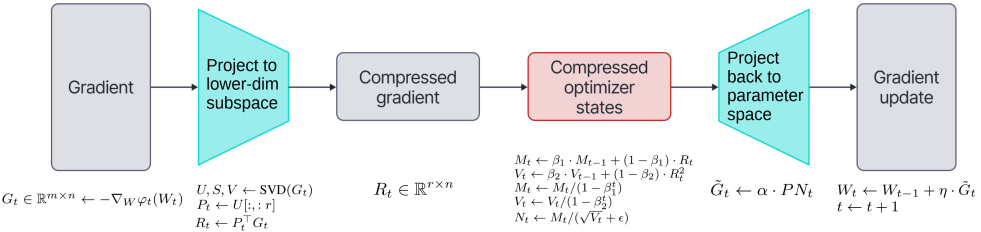
- Trainable Parameters: 14GB
- Adam States (M,V): 28GB
- Gradients: 14GB
- Activations: 2GB
- Total: 58GB



Existing Memory-Efficient Optimizers

Focus on memory-efficient variations of Adam

- Pioneered by GaLore (ICML'24), focus on compressing the Adam optimizer states.
- **Main idea**: Project the gradient $G \in \mathbb{R}^{m \times n}$ into low-rank: $P^T G$, $P \in \mathbb{R}^{m \times r}$; compute Adam states in low-dim; then project-up again.



Motivation

- **Growing literature on memory-efficient optimizers**: e.g., GaLore (ICML'24), Apollo (MLSys'25), SWAN (ICML'25), Fira, SinkGD (NeurIPS'25).
- Recent optimizers start to directly remove the optimizer states, e.g., Muon (2024), Scion, Swan (ICML'25).
- Most **existing works** either **build on-top of Adam** or use computationally heavy components.
- **No systematic study** to identify the most essential algorithmic components beyond plain SGD.

Question

Can we design a memory-efficient optimizer with minimum modifications to plain SGD (even without momentum) and SOTA pre-training performance?

Pre-training optimizer: From SGD to Adam

Dissecting the key components of Adam:

Algorithm 1 SGD (one step)

$$g^t \leftarrow \nabla \ell(\theta^t; \xi_t)$$
$$\theta^{t+1} \leftarrow \theta^t - \eta g^t$$

Algorithm 2 sign SGD (one step)

$$g^t \leftarrow \nabla \ell(\theta^t; \xi_t)$$
$$\theta^{t+1} \leftarrow \theta^t - \eta \frac{g^t}{\sqrt{g^t \odot g^t}}$$

Algorithm 3 Adam (one step)

$$g^t \leftarrow \nabla \ell(\theta^t; \xi_t)$$
$$m^t \leftarrow \beta_1 m^t + (1-\beta_1) g^t$$
$$v^t \leftarrow \beta_2 v^t + (1-\beta_2) g^t \odot g^t$$
$$\theta^{t+1} \leftarrow \theta^t - \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \frac{m^t}{\sqrt{v^t}+\epsilon}$$

From SGD to Adam:
- Gradient Normalization
- Exponential Moving Average (EMA)/Momentum

Key components of Adam: Grad-normalization

- Gradient normalization: control the magnitude of the update
- Variety of gradient normalization schemes, including:

$$\begin{aligned}
\text{Singular-value normalization:} &\quad UV^\top, \quad \text{where } G = U\Sigma V^\top \text{ (SVD)} \\
\text{Column-wise normalization:} &\quad \left[\frac{\text{col}_1(G)}{\|\text{col}_1(G)\|}, \ldots, \frac{\text{col}_n(G)}{\|\text{col}_n(G)\|}\right] \\
\text{Row-wise normalization:} &\quad \left[\frac{\text{row}_1(G)^\top}{\|\text{row}_1(G)\|}, \ldots, \frac{\text{row}_m(G)^\top}{\|\text{row}_m(G)\|}\right]^\top \\
\text{Sign normalization:} &\quad \text{sign}(G)
\end{aligned}$$

where $G \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, therefore row- and column-normalizations correspond to normalizing along the input and output dimensions, respectively.

Key components of Adam: Grad-normalization

Efficiency of different gradient normalizations (time in ms)

dimension d	1024	2048	4096
singular-value	79.77	354.27	1958.66
singular-value (NS)	6.03	7.00	14.41
column-wise	0.10	0.12	0.17
row-wise	0.09	0.11	0.13
sign	0.03	0.03	0.03

- Time (ms) to normalize a square torch matrix on NVIDIA A40 GPU.
- Singular-value variants are at least 50$\times$ more expensive than the rest.

Key components of Adam: Grad-normalization

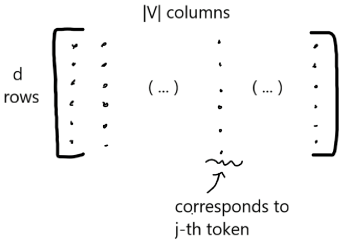
Performance of different gradient normalizations (perplexity)

	60M	**130M**	**350M**
Tokens	1.4B	2.6B	7.8B
Adam	28.77	22.20	16.80
singular-value (NS)	34.15	25.25	18.73
column-wise	39.89	28.85	20.38
row-wise	79.27	37.67	21.63
sign	54.36	40.42	27.95

- Singular-value (NS) performs best, but has reduced throughput (10-20%).
- Column-wise comes second but is more time-efficient.

Why Column-wise Normalization

The output (lm_head) layer is special. It maps the last hidden state to vocabulary logits; each column of G corresponds to one of $|V|$ tokens.

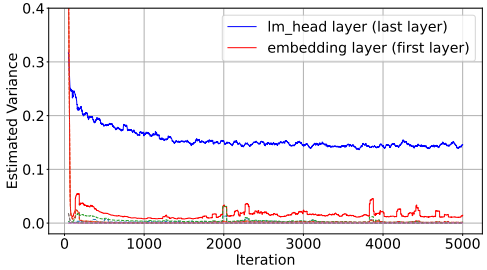


Sketch: last-layer grad matrix; each column = one token class.

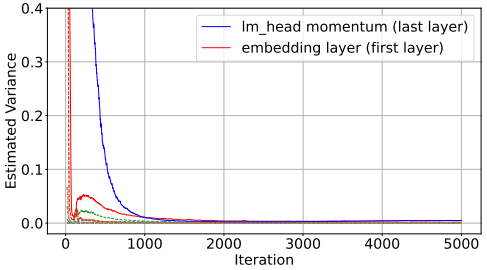
- Per-token-class grad norms span 3–4 orders of magnitude (frequent tokens dominate).
- Column-wise normalization equalizes per-class contributions — exactly the Type-II fix from the diagnosis.
- Row-wise / sign normalization do not respect this per-column structure; singular-value is too expensive.

Key components of Adam: EMA

- EMA is the origin of most of the memory overheads!
- Can we apply EMA more frugally? Lets estimate per-layer grad variance:



(a) SGD-col-norm



(b) SGD-col-norm-mmt-last

- (a) Gradient variance of the last-layer much higher than the rest.
- (b) Applying momentum only to last-layer reduces its variance.

Theoretical insights: Layer-wise grad variance

Denote the optimization problem of the LLM pretraining as

$$\min_{\theta=[\theta_1,\ldots,\theta_L]} \ell(\theta) := \frac{1}{n}\sum_{i=1}^{n} \ell(\theta;\xi_i) \tag{1}$$

Aslo consider the following SGD-M algorithm to solve (1):

$$\begin{aligned} m_l^t &= \beta_l m_l^{t-1} + (1-\beta_l) g^t,\ g^t = \nabla_{\theta_l} \ell(\theta^t;\xi_t) \\ \theta_l^{t+1} &= \theta_l^t - \eta_l m_l^t \end{aligned} \tag{2}$$

- We will show theoretically that momentum helps the most for the layers with larger gradient variances.

Theoretical insights: Layer-wise grad variance

Theorem

Suppose $\ell(\theta)$ in (1) is lower bounded by ℓ^*, γ-smooth (i.e. $\nabla\ell(\theta)$ is Lipschitz continuous with constant γ), also the stochastic gradient is unbiased $\mathbb{E}_{\xi_t}\nabla_{\theta_l}\ell(\theta^t;\xi_t) = \nabla_{\theta_l}\ell(\theta^t)$ and with bounded variance: $\mathbb{E}_{\xi_t}\|\nabla_{\theta_l}\ell(\theta^t;\xi_t) - \nabla_{\theta_l}\ell(\theta^t)\|^2 \leq \sigma_l^2$ for all $l = 1, ..., L$ and $t = 0, ..., T-1$. With appropriate choice of hyperparameters $\eta_l \geq \eta$ and $\beta_l \leq 1-\delta$ (δ is an absolute constant), we have the following convergence result for update (2):

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{l=1}^{L}\mathbb{E}\|\nabla_l^t\|^2 \leq \frac{2L\gamma^{3/2}\mathbb{E}\Delta_1}{\delta^2\sqrt{T}} + \sum_{l=1}^{L}\left(\frac{1-\beta_l}{1+\beta_l}\frac{L\sqrt{\gamma}}{4\sqrt{T}} + \frac{L\gamma^{3/2}}{2\sqrt{T}} + \frac{1-\beta_l}{\beta_l^3}\frac{\gamma^2}{4LT}\right)\frac{\sigma_l^2}{\delta^2} \quad (3)$$

where $\nabla_l^t = \nabla_{\theta_l}\ell(\theta^t;\xi_t)$ and $\Delta_1 = \ell(\theta^1) - \ell^*$.

Theoretical insights: Layer-wise grad variance

Remark

Theorem 1 suggests that β_l's should be all taken to be $1-\delta$. The second term of the right-hand side of (3) takes the form

$$f(x) = \frac{1-x}{1+x} + c\frac{1-x}{x^3},\ c := \frac{\gamma}{L^2 T^{3/2}}$$

where x represents the layer-wise momentum hyperparameter β_l.

$f(x)$ is decreasing in $(0, 1-\delta]$, optimal strategy: $\beta_l = 1-\delta$, for all $l = 1, \cdots L$.

However, since $\sigma_l = 0,\ \forall\ l \neq L$ close to zero, to improve memory efficiency, one can take $\beta_l = 0$ for $l = 1, ..., L-1$ and only keep the momentum for the last layer.

Adding momentum to last layer

- Our theoretical result suggests to add **momentum only to last-layer**:

Model Size	**60M**	**130M**	**350M**
Tokens	1.4B	2.6B	7.8B
Adam	28.77	22.20	16.80
singular-value (NS)	34.15	25.25	18.73
column-wise	39.89	28.85	20.38
Singular-val (NS) + mmt-last	31.20	22.33	16.67
Column-wise + mmt-last (ours)	30.81	22.57	16.32

Table: Evaluation perplexity of two normalizations (singular-value and column-wise) when combined with last-layer-momentum (mmt-last).

Column-wise and last-layer momentum close the gap with Adam!

SCALE – A Minimalist Pre-training Optimizer

SCALE = **S**tochastic **C**olumn-norm**A**lized **L**ast-layer mom**E**ntum.

Algorithm 1 SCALE: Stochastic Column-normalized Last-layer Momentum

Input: Initialized trainable parameters θ^0, hyperparameters β_t and $\eta_{t,l}$.
for $t = 0, 1, ..., T-1$ **do**
  Sample mini-batch data $\{\xi_{t,b}\}_{b=1,...,B}$;
  for Layers $l = 1, ..., L$ **do**
    Compute the stochastic gradient $g_l^t := \frac{1}{B}\sum_{b=1}^{B} \nabla_{\theta_l}\ell(\theta^t; \xi_{t,b})$;
    if $l = L$ (last layer) **then**
      $m_l^t = \beta_t m_l^{t-1} + (1-\beta_t) g_l^t$;
    else
      $m_l^t = g_l^t$ (record the gradient directly);
    end if
    $\theta_l^{t+1} = \theta_l^t - \eta_l \mathcal{C}(m_l^t)$ where $\mathcal{C}$ is the column-wise normalization;
  end for
end for
Output: Trained model θ^T.

- Start from plain SGD (zero state); add only what the diagnosis demands: (i) column-wise normalization on every weight matrix; (ii) last-layer momentum only.
- Memory: **SGD-like**: one EMA buffer for the last layer.
- Compute: one column-norm per matrix; negligible overhead.

SCALE – Headline Results on LLaMA 60M – 1B

Model size Tokens	**60M** 1.4B	**130M** 2.6B	**350M** 7.8B	**1B** 20B
Adam	30.05 (0.35G)	23.13 (0.81G)	18.77 (2.21G)	15.79 (8.04G)
Adam (Stable-SPAM)	28.77 (0.35G)	22.20 (0.81G)	16.80 (2.21G)	13.30 (8.04G)
Muon	**28.86** (0.23G)	**22.20** (0.54G)	16.70 (1.47G)	13.67 (5.36G)
GaLore	34.58 (0.28G)	25.31 (0.61G)	19.37 (1.59G)	15.05 (4.76G)
Fira	30.34 (0.28G)	22.96 (0.61G)	16.82 (1.59G)	14.36 (4.76G)
SWAN	30.00 (0.25G)	22.83 (0.46G)	17.14 (1.00G)	–
APOLLO	30.94 (0.28G)	22.93 (0.61G)	16.75 (1.59G)	14.28 (4.76G)
APOLLO-Mini	31.85 (0.25G)	23.63 (0.46G)	17.11 (1.00G)	**13.48** (3.20G)
SCALE (ours)	30.81 (**0.15G**)	22.57 (**0.32G**)	**16.32** (**0.80G**)	**13.49** (**2.81G**)

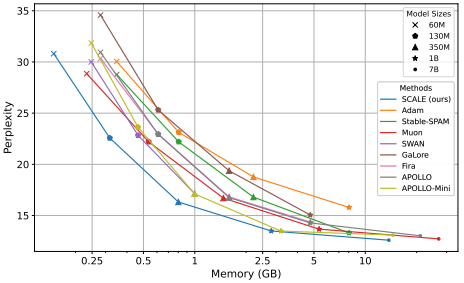
Perplexity (lower is better) on C4; numbers in parentheses = optimizer-state memory.

- SCALE **matches or beats** Adam at $\sim 1/3$ to $1/2$ of its optimizer memory.
- Dominates GaLore / Fira / SWAN / APOLLO / Muon at every model size on memory *and* usually on perplexity.

SCALE – LLaMA-7B and the Pareto Frontier

7B (steps) **Tokens**	40K 5.2B	80K 10.5B	120K 15.7B	150K 19.7B
APOLLO (16.14G)	17.55	14.39	13.23	13.02
APOLLO-Mini (14.53G)	18.03	14.60	13.32	13.09
Muon (26.95G)	16.43	13.95	12.85	12.72
SCALE (13.74G)	17.99	14.57	12.86	**12.59**

LLaMA-7B perplexity on C4. SCALE achieves the best final perplexity at the smallest optimizer footprint.



Memory–perplexity Pareto plot across optimizers; SCALE pushes the lower-left frontier.

- **Conclusion:** Building on classical SGD can obtain the new memory-efficient SOTA, as long as we respect gradient structure in LLM training.

Part II. Making Lookahead Work

EMA-Nesterov: Stabilizing Nesterov's Lookahead for Accelerated Deep Learning Optimization

Chung-Yiu Yau, Dawei Li, Athanasios Glentis, Valentyn Boreiko, Hoi-To Wai, Mingyi Hong

University of Minnesota · Amazon AGI · The Chinese University of Hong Kong

Nesterov Acceleration: A Classical Idea

Nesterov (1983) Accelerated Gradient. For L-smooth convex f, take a lookahead step before each gradient evaluation:

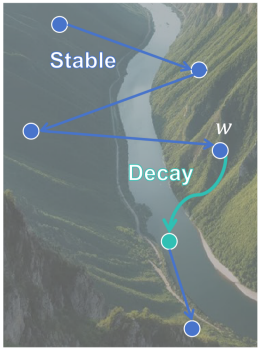
$$y^t = x^t + \beta_t (x^t - x^{t-1}), \qquad x^{t+1} = y^t - \tfrac{1}{L}\nabla f(y^t).$$

- Achieves the accelerated rate $f(x^T) - f(x^\star) = \mathcal{O}(L/T^2)$ which is optimal among first-order methods on convex problems.
- One of the most widely-used ideas in *classical* optimization.

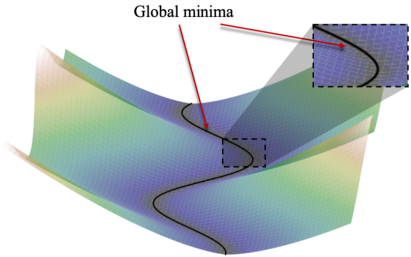
Question. *Can this classical acceleration help in deep-learning / LLM pre-training?*

The catch: deep-learning losses are non-convex, exhibit a river-valley geometry *(Wen et al., 2024)*: a long descending corridor with steep walls; naive lookahead may fail to work.

Deep-Learning Loss Landscape – River Valley



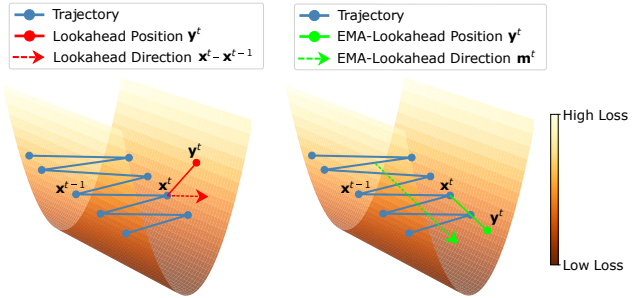
(a) River Valley Landscape



(b) Loss landscape of over-parameterized models

Figure: Deep-learning loss landscape described by (Left) Wen et al. (2024) and (Right) Liu et al. (2022). Training trajectory progresses above the minima, and converges to minima when learning rate decays.

Why Naive Lookahead Fails, and Our Fix

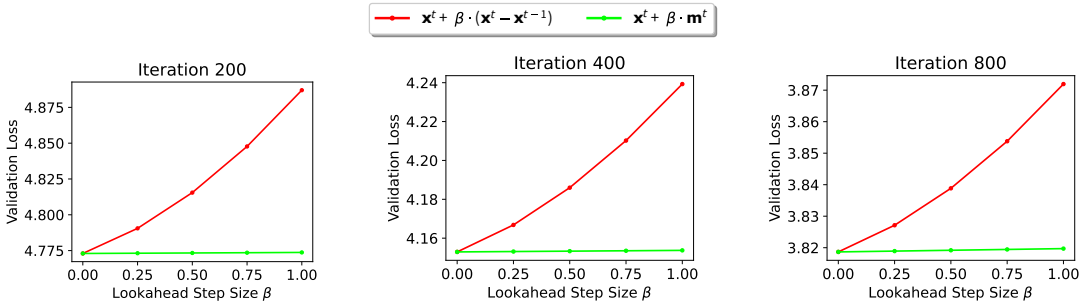


Classical lookahead (red): extrapolates the noisy one-step update $x^t - x^{t-1}$, overshoots, training diverges.

Our fix — EMA lookahead (green): replace the one-step delta with an EMA of past updates (the low-frequency trend).

$$m^t = \gamma\, m^{t-1} + (1-\gamma)\,(x^t - x^{t-1}), \qquad y^t = x^t + \beta\, m^t$$

Can You Look Ahead? Lookahead Loss vs. β



NanoGPT-124M: validation loss as we vary β along two lookahead directions at iterations 200/400/800.

- Classical $(x^t - x^{t-1})$: loss explodes at any nontrivial β.
- EMA (m^t): a wide, safe interval of β values, and lower validation loss.

The EMA-Lookahead Idea

- Replace the raw extrapolation by an exponential moving average of past updates:

$$m^t = \gamma\, m^{t-1} + (1-\gamma)\,(x^t - x^{t-1})$$

- Then look ahead via $y^t = x^t + \beta\, m^t$ before any base-optimizer step.

Intuition. The training trajectory follows a descending trend; instantaneous updates oscillate around it.

- EMA filters out the high-frequency oscillation $\Rightarrow$ keeps only the trend.
- Lookahead along the trend stays inside the river-valley corridor.

EMA-Nesterov – A Universal Wrapper

For any base optimizer $\mathcal{A}_t : \mathbb{R}^d \to \mathbb{R}^d$ (Adam, SOAP, Muon, ...):

Algorithm: EMA-Nesterov

$$
\begin{aligned}
&\textbf{for } t = 0, \ldots, T-1: \\
&\quad x^{t+1} = \mathcal{A}_t\big(x^t + \beta\, m^t\big) \\
&\quad m^{t+1} = \gamma\, m^t + (1-\gamma)\,(x^{t+1} - x^t) \\
&\textbf{end for}; \quad \textbf{return } x^T.
\end{aligned}
$$

- $\gamma = 0$ recovers classical Nesterov lookahead.
- $\gamma \in (0, 1)$: low-pass filter on the update direction.
- Adds one extra buffer (m^t) and one extra cheap update per step.

Theory: Acceleration is Preserved in the Convex Regime

Theorem 1 (strongly convex)

Let f be L-smooth and μ-strongly convex with $\kappa = L/\mu$, and let $\mathcal{A}_t$ be gradient descent. For a suitable $\{\beta_t\}$,

$$f(x^T) - f(x^\star) \;=\; \mathcal{O}\Big(\big(1 - \sqrt{(1-\gamma)/\kappa}\big)^T\Big) \quad \text{for any } \gamma \in [0,\, 1 - 1/\kappa].$$

- Strictly faster than gradient descent's $\mathcal{O}((1-1/\kappa)^T)$ whenever $\gamma \le 1 - 1/\kappa$.

Theorem 2 (convex, $\mu = 0$)

$$f(x^T) - f(x^\star) \;=\; \mathcal{O}\Big(\tfrac{L}{(1-\gamma)\,T^2}\Big) \quad \text{for any } \gamma \in [0, 1).$$

- Strictly faster than gradient descent's $\mathcal{O}(1/T)$ once $T = \Omega(1/(1-\gamma))$.

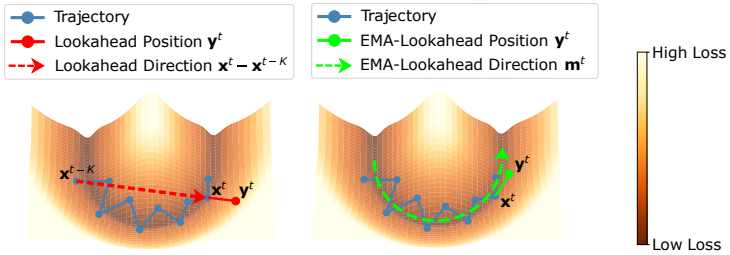
Comparison to Existing Lookahead Algorithms

Prior Works	Lookahead Direction	Lookahead Frequency	Adaptivity
Nesterov's Accelerated Gradient (Nesterov, 1983)	$\mathbf{x}^t - \mathbf{x}^{t-1}$	1	✓
Pessimistic Lookahead (Zhang et al., 2019)	$-(\mathbf{x}^t - \mathbf{x}^{t-K})$	$1/K$	✗
NALA (Zuo et al., 2024)	$\mathbf{x}^t - \mathbf{x}^{t-K}$	$1/K$	✗
SNOO (Kallusky et al., 2025)	$\mathbf{x}^t - \mathbf{x}^{t-K}$ (Implicit)	$1/K$	✗
GPA (Defazio et al., 2025)	$\mathbf{x}^t - \mathbf{x}^{t-1} + \mu \cdot (\mathbf{x}^t - \mathrm{EMA}(\{\mathbf{x}^r\}_{r=0}^{t-1}))$ (Implicit)	1	✗
EMA-Nesterov (Ours)	$\mathrm{EMA}(\{\mathbf{x}^r - \mathbf{x}^{r-1}\}_{r=1}^{t})$	1	✓

Table: Comparison between lookahead algorithms on the direction (short-horizon vs long-horizon) and frequency of lookahead. $K > 1$ is the number of optimizer updates for double-loop algorithms.

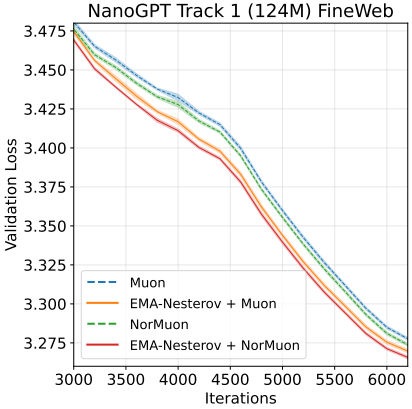
Comparison to Existing Lookahead Algorithms

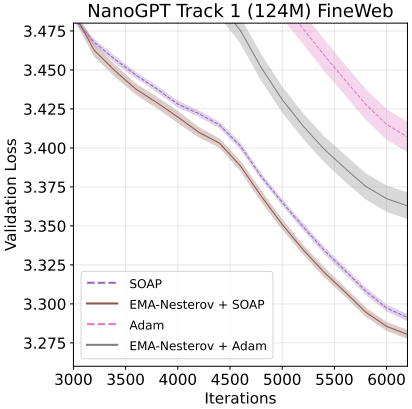
- Prior works are either
  (1) prone to short-horizon oscillation from $\mathbf{x}^t - \mathbf{x}^{t-1}$, or
  (2) perform long-horizon lookahead $\mathbf{x}^t - \mathbf{x}^{t-K}$ that does not adapt to the local trend of trajectory (see red below).



- In contrast, EMA-Nesterov's lookahead direction adapts to the recent trend (see green above).
- Prior works do not provide accelerated convergence in convex optimization.

Numerical Results – NanoGPT (Base Optimizers)

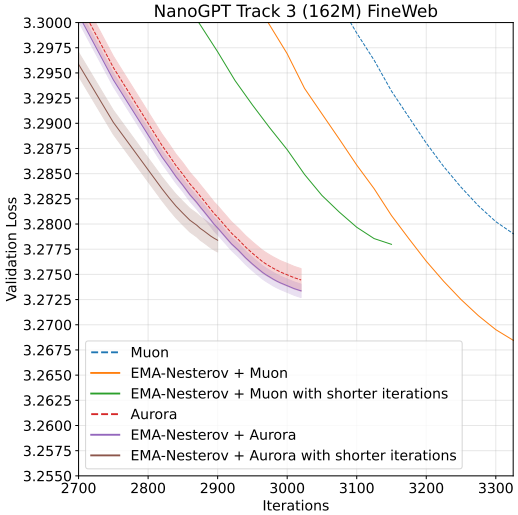




NanoGPT-124M, base optimizers $\in$ {Muon, NorMuon, SOAP, Adam}. Mean ± 1 s.d. over multiple seeds.

- EMA-Nesterov accelerates **every** base optimizer tested, e.g., Muon / NorMuon.

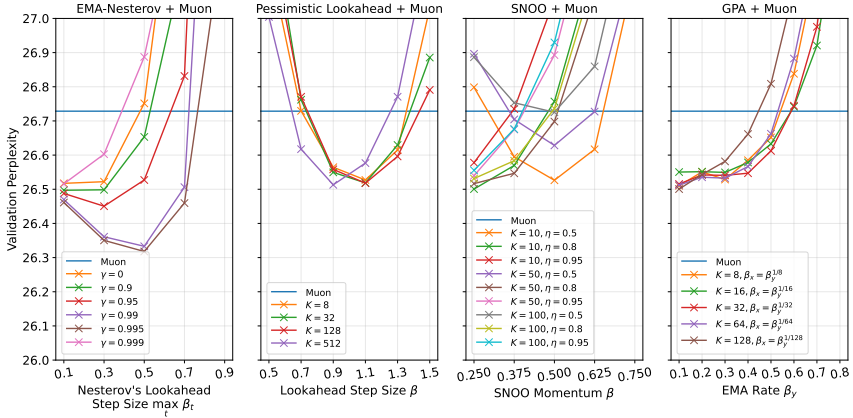
Numerical Results – NanoGPT (Base Optimizers)



NanoGPT-162M, base optimizers $\in$ {Muon, Aurora}. Mean ± 1 s.d. over multiple seeds.

- EMA-Nesterov accelerates **every** base optimizer tested, e.g., Muon / Aurora.

Numerical Experiments – NanoGPT (Lookahead Algorithms)



- EMA-Nesterov performs better than existing lookahead algorithms under comprehensive hyperparameter tuning for all methods on NanoGPT.

Numerical Experiments – Llama (Data / Batch Size Scaling)

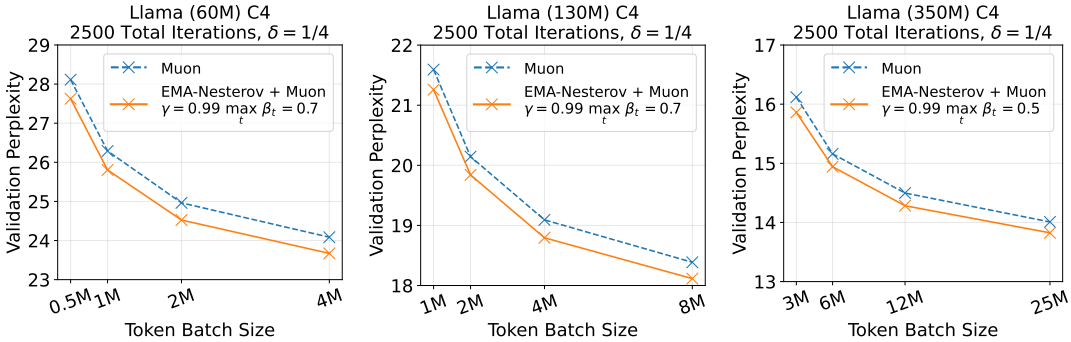
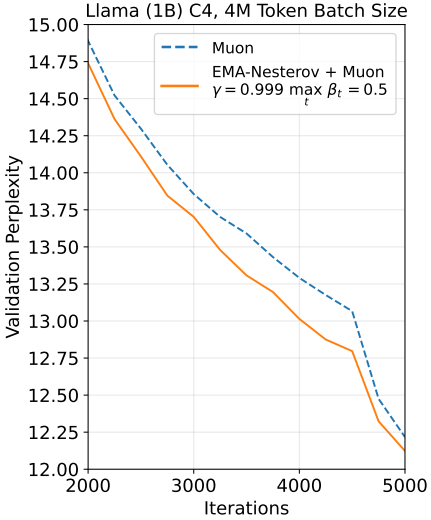


Figure: Data scaling to total training budgets of $\geq \{20\times, 40\times, 80\times, 160\times\}$ tokens-per-parameter with increased token batch size and fixed 2500 total iterations.

- EMA-Nesterov consistently accelerates large batch training.

Numerical Experiments – Llama (1B Model)



- Training curve of Muon and EMA-Nesterov + Muon on Llama 1B for 5000 iterations.
- EMA-Nesterov consistently accelerates larger model training.

Conclusions

A common message across two threads:

Classical optimization ideas, treated with the right respect for the structure of LLM, are very useful for modern LLM training.

- **Part I (SGD).** A two-property diagnosis of the SGD/Adam gap $\Rightarrow$ SCALE: stateless column-norm + last-layer momentum, $\sim$ 35–45% of Adam's memory, matches or beats Adam from 60M to 7B.
- **Part II (Nesterov).** EMA-stabilized lookahead $\Rightarrow$ EMA-Nesterov: keeps the classical accelerated convex rate, consistently accelerates Adam / SOAP / Muon / NorMuon on NanoGPT, improves performance on large batch LLaMA and large model up to LLaMA-1B.

Open Directions

- **Theory for clipped SGD on LLM gradients.** A convergence theory matching the empirical SCALE /SGD-LL story.
- **EMA-Nesterov as a generic optimizer wrapper.** More extensive experiments needed to validate the effects of EMA-Nesterov for larger-scale training (Part II).
- **Beyond pre-training.** Do the models trained via different optimizers have different effect on the downstream tasks? (Perplexity/Eval loss is not the only evaluator)
- **Hardware-aware optimizer design.** Memory- and energy-constrained optimization for trillion-parameter models.
- **Other "classical" ideas to revisit:** variance reduction, second-order methods, primal-dual splitting; Will some of those be helpful in foundation model training?

Thank you!

Questions?

SGD-LL arXiv

SCALE arXiv

SCALE GitHub

EMA-Nesterov Paper

References I

Defazio, A., Mishchenko, K., Raman, P., Shi, H.-J. M., and Xiao, L. (2025). Smoothing diloco with primal averaging for faster training of llms. *arXiv preprint arXiv:2512.17131*.

Kallusky, D., Rao, V., Nandavanam, V., and Shi, H.-J. M. (2025). Snoo: Step-k nesterov outer optimizer-the surprising effectiveness of nesterov momentum applied to pseudo-gradients. *arXiv preprint arXiv:2510.15830*.

Liu, C., Zhu, L., and Belkin, M. (2022). Loss landscapes and optimization in over-parameterized non-linear systems and neural networks. *Applied and Computational Harmonic Analysis*, 59:85–116.

Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. In *Dokl akad nauk Sssr*, volume 269, page 543.

Wen, K., Li, Z., Wang, J., Hall, D., Liang, P., and Ma, T. (2024). Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective. *arXiv preprint arXiv:2410.05192*.

Zhang, M., Lucas, J., Ba, J., and Hinton, G. E. (2019). Lookahead optimizer: k steps forward, 1 step back. *Advances in neural information processing systems*, 32.

Zuo, X., Li, H.-Y., Gao, S., Zhang, P., and Du, W.-R. (2024). Nala: a nesterov accelerated look-ahead optimizer for deep learning. *PeerJ Computer Science*, 10:e2167.

The Memory Wall: Adam at LLM Scale

LLaMA-7B memory budget (BF16):

- Parameters: 14 GB
- Gradients: 14 GB
- Activations: 2 GB
- Adam states (m, v): 28 GB
- **Total:** 58 **GB** $>$ A100 40 GB

$\Rightarrow$ Optimizer state dominates memory at LLM scale.

Memory-efficient workarounds — all built on top of Adam:

- Adafactor *(Shazeer & Stern '18)*, 8-bit Adam *(Dettmers+ '22)*: compress Adam's states.
- GaLore *(Zhao+ ICML'24)* and follow-ups Fira, APOLLO, SWAN, SinkGD: low-rank projection of gradient before Adam.
- Muon *('24)*, SOAP, Scion, NorMuon ('24–'25): orthogonalize / precondition; still carry momentum buffers.

Unstated assumption: classical, stateless methods cannot reach Adam's performance — so we must compress Adam.

Why is the Gradient So Small? A Decomposition

Question. Why is the gradient / momentum tiny in pre-training?

Theorem (Gradient Norm Upper Bound with Partition)

Given any partition of token classes into head $\mathcal{H}$ and tail $\mathcal{T}$, the output-layer gradient norm satisfies

$$\|G\|_F \leq \mathcal{O}(1/\sqrt{B}) + \mathcal{O}(\varepsilon_{\mathcal{H}}) + \mathcal{O}(q_{\mathcal{T}}) + \mathcal{O}(p_{\mathcal{T}}),$$

where B is batch size, $\varepsilon_{\mathcal{H}}$ is the population gradient contribution of head tokens, and $q_{\mathcal{T}}, p_{\mathcal{T}}$ are the ground-truth and model marginal probabilities of tail tokens.

- Large batch $B \Rightarrow \mathcal{O}(1/\sqrt{B})$ covariance term is small.
- Head tokens become well-trained early $\Rightarrow \varepsilon_{\mathcal{H}}$ is small.
- Tail tokens are rare during training $\Rightarrow p_{\mathcal{T}}, q_{\mathcal{T}}$ are small.

Conclusion: Gradient norm is structurally tiny, so large effective LR is unavoidable.