Can an Algorithm Change a Scaling Law?

The Story of Momentum

Princeton 2026 Summer School in Machine Learning

The stochastic first-order oracle

A problem P specifies a convex objective on a convex domain:

$$L_P : \Theta_d \longrightarrow \mathbb{R}, \qquad \Theta_d \subseteq \mathbb{R}^d.$$

The class $\mathcal{P}_{\mathrm{cvx}}(G, \rho)$ requires an optimizer near the initial point,

$$\theta_P^\star \in \operatorname*{arg\,min}_{\theta \in \Theta_d} L_P(\theta), \qquad \|\theta_0 - \theta_P^\star\| \le \rho.$$

At a query θ, the oracle returns a stochastic subgradient $g(\theta, \xi)$:

$$\mathbb{E}[g(\theta, \xi) \mid \theta] \in \partial L_P(\theta), \qquad \mathbb{E}[\|g(\theta, \xi)\|_*^2 \mid \theta] \le G^2.$$

Along growing problems, both scales may depend on dimension: G_d and ρ_d.

The stochastic first-order oracle

A problem P specifies a convex objective on a convex domain:

$$L_P : \Theta_d \longrightarrow \mathbb{R}, \qquad \Theta_d \subseteq \mathbb{R}^d.$$

The class $\mathcal{P}_{\mathrm{cvx}}(G, \rho)$ requires an optimizer near the initial point,

$$\theta_P^\star \in \operatorname*{arg\,min}_{\theta \in \Theta_d} L_P(\theta), \qquad \|\theta_0 - \theta_P^\star\| \le \rho.$$

At a query θ, the oracle returns a stochastic subgradient $g(\theta, \xi)$:

$$\mathbb{E}[g(\theta,\xi) \mid \theta] \in \partial L_P(\theta), \qquad \mathbb{E}[\|g(\theta,\xi)\|_*^2 \mid \theta] \le G^2.$$

Along growing problems, both scales may depend on dimension: G_d and ρ_d.

The stochastic first-order oracle

A problem P specifies a convex objective on a convex domain:

$$L_P : \Theta_d \longrightarrow \mathbb{R}, \qquad \Theta_d \subseteq \mathbb{R}^d.$$

The class $\mathcal{P}_{\mathrm{cvx}}(G, \rho)$ requires an optimizer near the initial point,

$$\theta_P^\star \in \operatorname*{arg\,min}_{\theta \in \Theta_d} L_P(\theta), \qquad \|\theta_0 - \theta_P^\star\| \le \rho.$$

At a query θ, the oracle returns a stochastic subgradient $g(\theta, \xi)$:

$$\mathbb{E}[g(\theta,\xi) \mid \theta] \in \partial L_P(\theta), \qquad \mathbb{E}[\|g(\theta,\xi)\|_*^2 \mid \theta] \le G^2.$$

Along growing problems, both scales may depend on dimension: G_d and ρ_d.

The stochastic first-order oracle

A problem P specifies a convex objective on a convex domain:

$$L_P : \Theta_d \longrightarrow \mathbb{R}, \qquad \Theta_d \subseteq \mathbb{R}^d.$$

The class $\mathcal{P}_{\mathrm{cvx}}(G, \rho)$ requires an optimizer near the initial point,

$$\theta_P^\star \in \operatorname*{arg\,min}_{\theta \in \Theta_d} L_P(\theta), \qquad \|\theta_0 - \theta_P^\star\| \le \rho.$$

At a query θ, the oracle returns a stochastic subgradient $g(\theta, \xi)$:

$$\mathbb{E}[g(\theta, \xi) \mid \theta] \in \partial L_P(\theta), \qquad \mathbb{E}[\|g(\theta, \xi)\|_*^2 \mid \theta] \le G^2.$$

Along growing problems, both scales may depend on dimension: G_d and ρ_d.

Traditional worst-case complexity

After R adaptive oracle calls, the minimax excess loss is

$$\inf_{\text{algorithms } \mathcal{A}} \sup_{P \in \mathcal{P}_{\mathrm{cvx}}(G,\rho)} \mathbb{E}\Big[L_P(\widehat{\theta}_R^{\mathcal{A}}) - L_P^\star\Big] = \Theta\Big(\frac{G\rho}{\sqrt{R}}\Big).$$

Best algorithm

The infimum ranges over every adaptive algorithm; projected SGD attains this rate.

Worst problem

The supremum chooses the hardest problem in the fixed class; a matching lower bound makes $R^{-1/2}$ minimax-optimal (Ramdas and Singh 2013).

This does not predict an instance-specific PLRF exponent, the scaling of $G_d \rho_d$, or the FLOP cost of one oracle call.

Traditional worst-case complexity

After R adaptive oracle calls, the minimax excess loss is

$$\inf_{\text{algorithms } \mathcal{A}} \sup_{P \in \mathcal{P}_{\text{cvx}}(G,\rho)} \mathbb{E}\left[L_P(\widehat{\theta}_R^{\mathcal{A}}) - L_P^\star\right] = \Theta\left(\frac{G\rho}{\sqrt{R}}\right).$$

Best algorithm

The infimum ranges over every adaptive algorithm; projected SGD attains this rate.

Worst problem

The supremum chooses the hardest problem in the fixed class; a matching lower bound makes $R^{-1/2}$ minimax-optimal (Ramdas and Singh 2013).

This does not predict an instance-specific PLRF exponent, the scaling of $G_d \rho_d$, or the FLOP cost of one oracle call.

Traditional worst-case complexity

After R adaptive oracle calls, the minimax excess loss is

$$\inf_{\text{algorithms } \mathcal{A}} \sup_{P \in \mathcal{P}_{\mathrm{cvx}}(G,\rho)} \mathbb{E}\Big[L_P(\widehat{\theta}_R^{\mathcal{A}}) - L_P^\star\Big] = \Theta\left(\frac{G\rho}{\sqrt{R}}\right).$$

Best algorithm

The infimum ranges over every adaptive algorithm; projected SGD attains this rate.

Worst problem

The supremum chooses the hardest problem in the fixed class; a matching lower bound makes $R^{-1/2}$ minimax-optimal (Ramdas and Singh 2013).

This does not predict an instance-specific PLRF exponent, the scaling of $G_d\rho_d$, or the FLOP cost of one oracle call.

Traditional worst-case complexity

After R adaptive oracle calls, the minimax excess loss is

$$\inf_{\text{algorithms } \mathcal{A}} \sup_{P \in \mathcal{P}_{\mathrm{cvx}}(G,\rho)} \mathbb{E}\Big[L_P(\widehat{\theta}_R^{\mathcal{A}}) - L_P^\star\Big] = \Theta\Big(\frac{G\rho}{\sqrt{R}}\Big).$$

Best algorithm

The infimum ranges over every adaptive algorithm; projected SGD attains this rate.

Worst problem

The supremum chooses the hardest problem in the fixed class; a matching lower bound makes $R^{-1/2}$ minimax-optimal (Ramdas and Singh 2013).

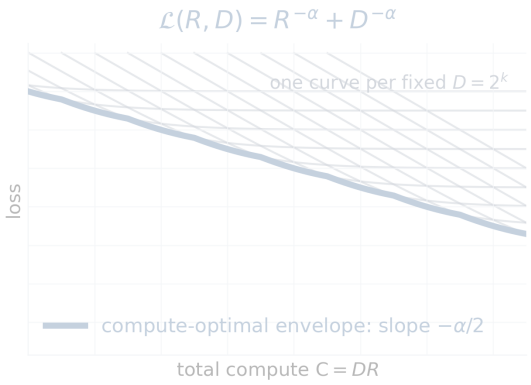
This does not predict an instance-specific PLRF exponent, the scaling of $G_d \rho_d$, or the FLOP cost of one oracle call.

Compute-optimal loss

Fix a sequence of growing problems $(\mathcal{P}_D)_{D\geq 1}$. At compute budget C, choose both the model dimension D and training horizon R:

$$\mathcal{L}_{\star,A}(\mathrm{C}) := \inf_{D,R} \ \mathcal{L}_A(D,R)$$
$$\text{subject to } \ \mathrm{FLOPs}_A(D,R) \leq \mathrm{C}.$$

The lower envelope is the **compute-optimal loss curve**.

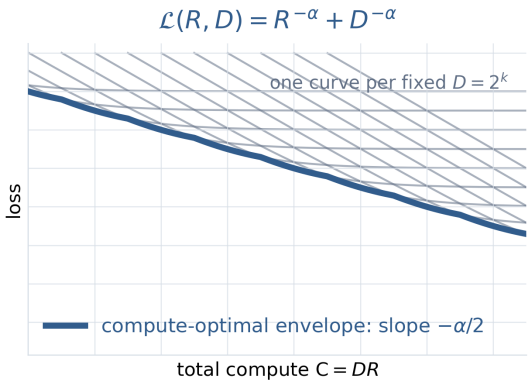


Compute-optimal loss

Fix a sequence of growing problems $(\mathcal{P}_D)_{D\geq 1}$. At compute budget C, choose both the model dimension D and training horizon R:

$$\mathcal{L}_{\star,A}(\mathrm{C}) := \inf_{D,R} \ \mathcal{L}_A(D,R)$$
$$\text{subject to } \ \mathrm{FLOPs}_A(D,R) \leq \mathrm{C}.$$

The lower envelope is the **compute-optimal loss curve**.



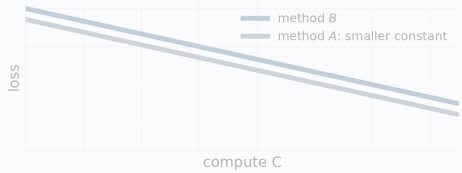
A problem-dependent notion of algorithmic gain

Fix the problem sequence and FLOP accounting. Suppose each algorithm has a compute-optimal law

$$\mathcal{L}_{\star,A}(C) \sim c_A C^{-\eta_A}, \qquad \mathcal{L}_{\star,B}(C) \sim c_B C^{-\eta_B}.$$

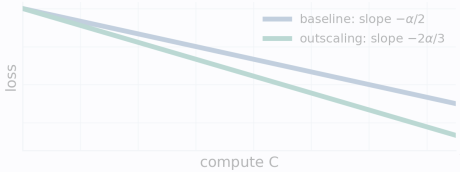
Multiplicative gain

$c_A < c_B$: lower loss, same slope.



Algorithmic outscaling

$\eta_A > \eta_B$: a steeper slope; A **outscales** B on this class.



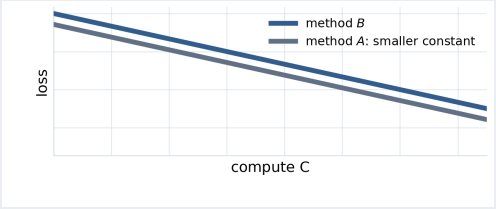
A problem-dependent notion of algorithmic gain

Fix the problem sequence and FLOP accounting. Suppose each algorithm has a compute-optimal law

$$\mathcal{L}_{\star,A}(\mathrm{C}) \sim c_A \mathrm{C}^{-\eta_A}, \qquad \mathcal{L}_{\star,B}(\mathrm{C}) \sim c_B \mathrm{C}^{-\eta_B}.$$

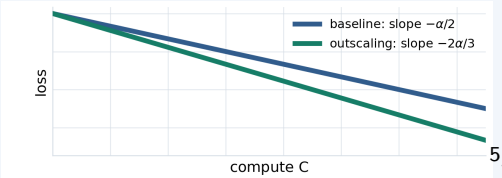
Multiplicative gain

$c_A < c_B$: lower loss, same slope.



Algorithmic outscaling

$\eta_A > \eta_B$: a steeper slope; A **outscales** B on this class.



Two questions, one tractable test case

Ambitious target

Can any algorithm outscale Adam on the Chinchilla problem?

Solvable test case

Can any algorithm outscale SGD on the PLRF problem?

Two questions, one tractable test case

Ambitious target

Can any algorithm outscale Adam on the Chinchilla problem?

Solvable test case

Can any algorithm outscale SGD on the PLRF problem?

Polyak momentum adds a gradient buffer

Gradient descent

$$g_{t+1} = \nabla L(\theta_t),$$
$$\theta_{t+1} = \theta_t - \gamma g_{t+1}.$$

Polyak momentum

$$g_{t+1} = \nabla L(\theta_t),$$
$$m_{t+1} = \beta m_t + (1-\beta) g_{t+1},$$
$$\theta_{t+1} = \theta_t - \gamma m_{t+1}.$$

Momentum adds an optimizer state m_t that carries gradient history.

Polyak momentum adds a gradient buffer

Gradient descent

$$g_{t+1} = \nabla L(\theta_t),$$
$$\theta_{t+1} = \theta_t - \gamma g_{t+1}.$$

Polyak momentum

$$g_{t+1} = \nabla L(\theta_t),$$
$$m_{t+1} = \beta m_t + (1-\beta) g_{t+1},$$
$$\theta_{t+1} = \theta_t - \gamma m_{t+1}.$$

Momentum adds an optimizer state m_t that carries gradient history.

Polyak momentum adds a gradient buffer

Gradient descent

$$g_{t+1} = \nabla L(\theta_t),$$
$$\theta_{t+1} = \theta_t - \gamma g_{t+1}.$$

Polyak momentum

$$g_{t+1} = \nabla L(\theta_t),$$
$$m_{t+1} = \beta m_t + (1-\beta) g_{t+1},$$
$$\theta_{t+1} = \theta_t - \gamma m_{t+1}.$$

Momentum adds an optimizer state m_t that carries gradient history.

Fixed momentum has a fixed memory horizon

For the exponential moving average

$$m_{t+1} = \beta m_t + (1-\beta) g_{t+1}$$
$$= (1-\beta) \sum_{s=0}^{t} \beta^{t-s} g_{s+1}, \qquad \tau_{\mathrm{mem}} \asymp (1-\beta)^{-1}.$$

$\beta = 0.9$

Remembers roughly 10 steps.

$\beta = 0.99$

Remembers roughly 100 steps.

On the PLRF model, fixed Polyak momentum changes constants and transients, but not the compute exponent.

Fixed momentum has a fixed memory horizon

For the exponential moving average

$$m_{t+1} = \beta m_t + (1-\beta) g_{t+1}$$
$$= (1-\beta) \sum_{s=0}^{t} \beta^{t-s} g_{s+1}, \qquad \tau_{\mathrm{mem}} \asymp (1-\beta)^{-1}.$$

$\beta = 0.9$

Remembers roughly 10 steps.

$\beta = 0.99$

Remembers roughly 100 steps.

On the PLRF model, fixed Polyak momentum changes constants and transients, but not the compute exponent.

Fixed momentum has a fixed memory horizon

For the exponential moving average

$$m_{t+1} = \beta m_t + (1-\beta) g_{t+1}$$
$$= (1-\beta) \sum_{s=0}^{t} \beta^{t-s} g_{s+1}, \qquad \tau_{\mathrm{mem}} \asymp (1-\beta)^{-1}.$$

$\beta = 0.9$

Remembers roughly 10 steps.

$\beta = 0.99$

Remembers roughly 100 steps.

On the PLRF model, fixed Polyak momentum changes constants and transients, but not the compute exponent.

Fixed momentum has a fixed memory horizon

For the exponential moving average

$$m_{t+1} = \beta m_t + (1-\beta) g_{t+1}$$
$$= (1-\beta) \sum_{s=0}^{t} \beta^{t-s} g_{s+1}, \qquad \tau_{\text{mem}} \asymp (1-\beta)^{-1}.$$

$\beta = 0.9$

Remembers roughly 10 steps.

$\beta = 0.99$

Remembers roughly 100 steps.

On the PLRF model, fixed Polyak momentum changes constants and transients, but not the compute exponent.

Nesterov's accelerated gradient method

What if the memory horizon grows with time?

Instead of a fixed momentum coefficient, take

$$\beta_t = 1 - \frac{c}{t + t_0}, \qquad \tau_{\mathrm{mem}}(t) \asymp (1 - \beta_t)^{-1} \asymp t.$$

The optimizer now carries momentum on a logarithmic time scale: its memory horizon grows proportionally with its age.

In the small-step limit, this becomes

$$\ddot{\theta}(t) + \frac{c}{t}\dot{\theta}(t) + \nabla L(\theta(t)) = 0.$$

This is the continuous picture behind Nesterov's growing-memory acceleration (Su et al. 2016).

What if the memory horizon grows with time?

Instead of a fixed momentum coefficient, take

$$\beta_t = 1 - \frac{c}{t + t_0}, \qquad \tau_{\mathrm{mem}}(t) \asymp (1 - \beta_t)^{-1} \asymp t.$$

The optimizer now carries momentum on a logarithmic time scale: its memory horizon grows proportionally with its age.

In the small-step limit, this becomes

$$\ddot{\theta}(t) + \frac{c}{t}\dot{\theta}(t) + \nabla L(\theta(t)) = 0.$$

This is the continuous picture behind Nesterov's growing-memory acceleration (Su et al. 2016).

What if the memory horizon grows with time?

Instead of a fixed momentum coefficient, take

$$\beta_t = 1 - \frac{c}{t + t_0}, \qquad \tau_{\mathrm{mem}}(t) \asymp (1 - \beta_t)^{-1} \asymp t.$$

The optimizer now carries momentum on a logarithmic time scale: its memory horizon grows proportionally with its age.

In the small-step limit, this becomes

$$\ddot{\theta}(t) + \frac{c}{t}\dot{\theta}(t) + \nabla L(\theta(t)) = 0.$$

This is the continuous picture behind Nesterov's growing-memory acceleration (Su et al. 2016).

What if the memory horizon grows with time?

Instead of a fixed momentum coefficient, take

$$\beta_t = 1 - \frac{c}{t + t_0}, \qquad \tau_{\mathrm{mem}}(t) \asymp (1 - \beta_t)^{-1} \asymp t.$$

The optimizer now carries momentum on a logarithmic time scale: its memory horizon grows proportionally with its age.

In the small-step limit, this becomes

$$\ddot{\theta}(t) + \frac{c}{t}\dot{\theta}(t) + \nabla L(\theta(t)) = 0.$$

This is the continuous picture behind Nesterov's growing-memory acceleration (Su et al. 2016).

A weak-curvature mode runs on a square-root clock

Gradient flow

$$\dot{u}_\lambda + \lambda u_\lambda = 0, \qquad u_\lambda(t) = e^{-\lambda t} u_\lambda(0).$$

The mode moves when $t \asymp \lambda^{-1}$.

Nesterov ODE

$$\ddot{u}_\lambda + \frac{3}{t}\dot{u}_\lambda + \lambda u_\lambda = 0,$$

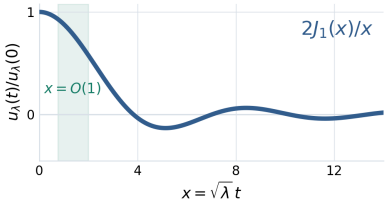
$$u_\lambda(t) = \frac{2J_1(\sqrt{\lambda}t)}{\sqrt{\lambda}t} u_\lambda(0).$$

The mode moves when $t \asymp \lambda^{-1/2}$.

For the quadratic loss

$$L(\theta) = \tfrac{1}{2}(\theta - \theta^\star)^\top H(\theta - \theta^\star),$$

$$Hq_\lambda = \lambda q_\lambda, \qquad u_\lambda = q_\lambda^\top(\theta - \theta^\star).$$



A weak-curvature mode runs on a square-root clock

Gradient flow

$$\dot{u}_\lambda + \lambda u_\lambda = 0, \qquad u_\lambda(t) = e^{-\lambda t} u_\lambda(0).$$

The mode moves when $t \asymp \lambda^{-1}$.

Nesterov ODE

$$\ddot{u}_\lambda + \frac{3}{t}\dot{u}_\lambda + \lambda u_\lambda = 0,$$

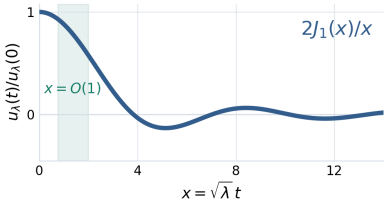
$$u_\lambda(t) = \frac{2J_1(\sqrt{\lambda}t)}{\sqrt{\lambda}t} u_\lambda(0).$$

The mode moves when $t \asymp \lambda^{-1/2}$.

For the quadratic loss

$$L(\theta) = \tfrac{1}{2}(\theta - \theta^\star)^\top H(\theta - \theta^\star),$$

$$Hq_\lambda = \lambda q_\lambda, \qquad u_\lambda = q_\lambda^\top(\theta - \theta^\star).$$



A weak-curvature mode runs on a square-root clock

Gradient flow

$$\dot{u}_\lambda + \lambda u_\lambda = 0, \qquad u_\lambda(t) = e^{-\lambda t} u_\lambda(0).$$

The mode moves when $t \asymp \lambda^{-1}$.

Nesterov ODE

$$\ddot{u}_\lambda + \frac{3}{t}\dot{u}_\lambda + \lambda u_\lambda = 0,$$

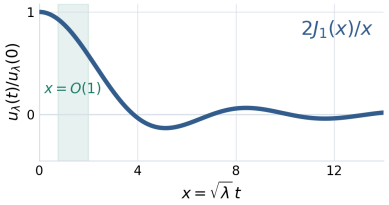
$$u_\lambda(t) = \frac{2J_1(\sqrt{\lambda}t)}{\sqrt{\lambda}t} u_\lambda(0).$$

The mode moves when $t \asymp \lambda^{-1/2}$.

For the quadratic loss

$$L(\theta) = \tfrac{1}{2}(\theta - \theta^\star)^\top H(\theta - \theta^\star),$$

$$Hq_\lambda = \lambda q_\lambda, \qquad u_\lambda = q_\lambda^\top(\theta - \theta^\star).$$



Nesterov's Accelerated Gradient Method

$$g_{t+1} = \nabla L(\Phi_t),$$
$$m_{t+1} = \mu_{t-1} m_t + g_{t+1},$$
$$\text{(extra gradient)} \qquad \Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \mu_t m_{t+1}).$$

Fresh gradient

g_{t+1} is injected directly into the update.

Memory channel

m_{t+1} accumulates an increasingly long gradient history.

NAG resembles momentum, but this extra gradient channel is important for stability.

Nesterov's Accelerated Gradient Method

$$g_{t+1} = \nabla L(\Phi_t),$$
$$m_{t+1} = \mu_{t-1} m_t + g_{t+1},$$
$$\text{(extra gradient)} \qquad \Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \mu_t m_{t+1}).$$

Fresh gradient

g_{t+1} is injected directly into the update.

Memory channel

m_{t+1} accumulates an increasingly long gradient history.

NAG resembles momentum, but this extra gradient channel is important for stability.

Nesterov's Accelerated Gradient Method

$$g_{t+1} = \nabla L(\Phi_t),$$
$$m_{t+1} = \mu_{t-1} m_t + g_{t+1},$$
$$\text{(extra gradient)} \qquad \Phi_{t+1} = \Phi_t - \gamma\left(g_{t+1} + \mu_t m_{t+1}\right).$$

Fresh gradient

g_{t+1} is injected directly into the update.

Memory channel

m_{t+1} accumulates an increasingly long gradient history.

NAG resembles momentum, but this extra gradient channel is important for stability.

Nesterov's Accelerated Gradient Method

$$g_{t+1} = \nabla L(\Phi_t),$$
$$m_{t+1} = \mu_{t-1} m_t + g_{t+1},$$
$$\text{(extra gradient)} \qquad \Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \mu_t m_{t+1}).$$

Fresh gradient

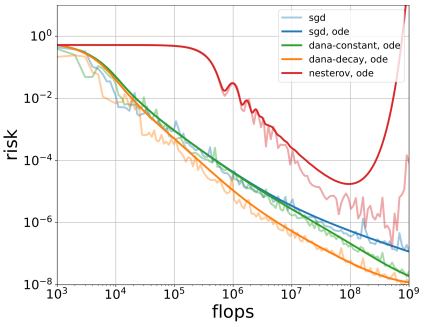
g_{t+1} is injected directly into the update.

Memory channel

m_{t+1} accumulates an increasingly long gradient history.

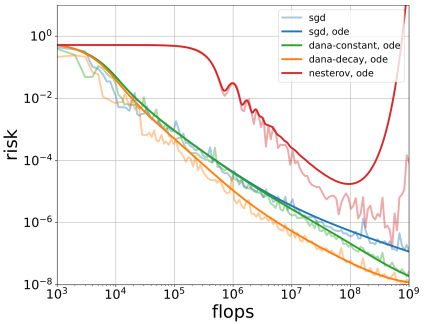
NAG resembles momentum, but this extra gradient channel is important for stability.

Naive stochastic Nesterov remembers gradient noise too



Replacing the full gradient by a batch-one gradient preserves the recursion, but not deterministic Nesterov's stability or acceleration guarantee.

Naive stochastic Nesterov remembers gradient noise too



Replacing the full gradient by a batch-one gradient preserves the recursion, but not deterministic Nesterov's stability or acceleration guarantee.

Let's introduce damping

Start from the same accumulator, but separate the update coefficient:

$$m_{t+1} = \mu_{t-1} m_t + g_{t+1},$$
$$\Phi_{t+1} = \Phi_t - \gamma\big(g_{t+1} + a_t m_{t+1}\big), \qquad a_t < \mu_t.$$

method	memory	contribution to update
Nesterov AG	$\mu_{t-1} \uparrow 1$	$a_t = \mu_t$: undamped
DANA-constant	$\mu_{t-1} \uparrow 1$	$a_t < \mu_t$: fixed damping
DANA-decaying	$\mu_{t-1} \uparrow 1$	$a_t < \mu_t$: time-adapted damping

AcSGD $\approx$ DANA-constant in these coordinates (Varré and Flammarion 2022); DANA-decaying is from (Ferbach et al. 2025).

Let's introduce damping

Start from the same accumulator, but separate the update coefficient:

$$m_{t+1} = \mu_{t-1} m_t + g_{t+1},$$
$$\Phi_{t+1} = \Phi_t - \gamma\big(g_{t+1} + a_t m_{t+1}\big), \qquad a_t < \mu_t.$$

method	memory	contribution to update
Nesterov AG	$\mu_{t-1} \uparrow 1$	$a_t = \mu_t$: undamped
DANA-constant	$\mu_{t-1} \uparrow 1$	$a_t < \mu_t$: fixed damping
DANA-decaying	$\mu_{t-1} \uparrow 1$	$a_t < \mu_t$: time-adapted damping

AcSGD $\approx$ DANA-constant in these coordinates (Varré and Flammarion 2022); DANA-decaying is from (Ferbach et al. 2025).

Let's introduce damping

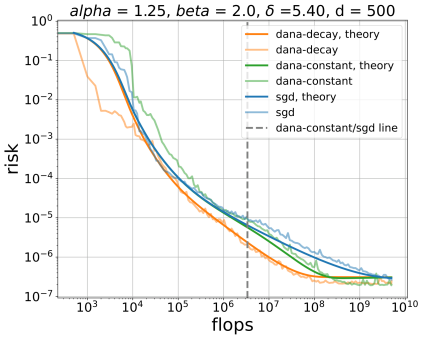
Start from the same accumulator, but separate the update coefficient:

$$m_{t+1} = \mu_{t-1} m_t + g_{t+1},$$
$$\Phi_{t+1} = \Phi_t - \gamma\big(g_{t+1} + a_t m_{t+1}\big), \qquad a_t < \mu_t.$$

method	memory	contribution to update
Nesterov AG	$\mu_{t-1} \uparrow 1$	$a_t = \mu_t$: undamped
DANA-constant	$\mu_{t-1} \uparrow 1$	$a_t < \mu_t$: fixed damping
DANA-decaying	$\mu_{t-1} \uparrow 1$	$a_t < \mu_t$: time-adapted damping

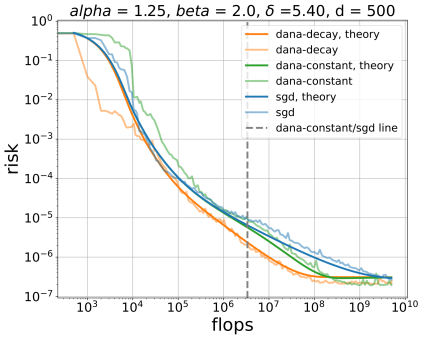
$\mathrm{AcSGD} \approx \mathrm{DANA\text{-}constant}$ in these coordinates (Varré and Flammarion 2022); DANA-decaying is from (Ferbach et al. 2025).

DANA steepens PLRF loss where fixed momentum does not



The methods share the early regime; the DANA branches steepen later, with the DANA-constant transition around $t \asymp d/B$ in the normalized-batch convention.

DANA steepens PLRF loss where fixed momentum does not



The methods share the early regime; the DANA branches steepen later, with the DANA-constant transition around $t \asymp d/B$ in the normalized-batch convention.

Acceleration is a change of clock

The controlled comparison changes only the optimizer

Recall the PLRF setup:

$$X_j = j^{-\alpha/2} Z_j, \qquad \mathrm{Var}(X_j) = j^{-\alpha}, \qquad b_j = j^{-\beta}.$$

A Gaussian feature map $W \in \mathbb{R}^{v \times d}$ gives

$$\mathcal{L}(\theta) = \left\| \Sigma^{1/2}(W\theta - b) \right\|_2^2, \qquad \widehat{K} = W^\top \Sigma W.$$

Covariance, target, feature map, batches, and compute model are fixed. Only the optimizer changes.

The controlled comparison changes only the optimizer

Recall the PLRF setup:

$$X_j = j^{-\alpha/2} Z_j, \qquad \mathrm{Var}(X_j) = j^{-\alpha}, \qquad b_j = j^{-\beta}.$$

A Gaussian feature map $W \in \mathbb{R}^{v \times d}$ gives

$$\mathcal{L}(\theta) = \left\| \Sigma^{1/2}(W\theta - b) \right\|_2^2, \qquad \widehat{K} = W^\top \Sigma W.$$

Covariance, target, feature map, batches, and compute model are fixed. Only the optimizer changes.

The controlled comparison changes only the optimizer

Recall the PLRF setup:

$$X_j = j^{-\alpha/2} Z_j, \qquad \mathrm{Var}(X_j) = j^{-\alpha}, \qquad b_j = j^{-\beta}.$$

A Gaussian feature map $W \in \mathbb{R}^{v \times d}$ gives

$$\mathcal{L}(\theta) = \left\| \Sigma^{1/2}(W\theta - b) \right\|_2^2, \qquad \widehat{K} = W^\top \Sigma W.$$

Covariance, target, feature map, batches, and compute model are fixed. Only the optimizer changes.

DANA-decay in optimizer coordinates

1. Continuize DANA-decay. In mode λ, use the coordinates

$$\Theta_{\lambda,t} = \langle \omega_\lambda, \Theta_t - \Theta^\star \rangle, \qquad Y_{\lambda,t} = \langle \omega_\lambda, Y_t \rangle.$$

With $s = 1 + t$, fixed γ, and $\kappa = 1/\alpha$, take

$$\gamma_1 = 1, \qquad \gamma_2 = \gamma, \qquad \gamma_3(t) = cs^{-\kappa}, \qquad \Delta(t) = \frac{\delta}{s}.$$

$$d\begin{pmatrix} \Theta_{\lambda,t} \\ Y_{\lambda,t} \end{pmatrix} = \begin{pmatrix} -\gamma\lambda & -cs^{-\kappa} \\ \lambda & -\delta/s \end{pmatrix} \begin{pmatrix} \Theta_{\lambda,t} \\ Y_{\lambda,t} \end{pmatrix} dt + \sqrt{\frac{\lambda \mathscr{P}(t)}{B}} \begin{pmatrix} -\gamma & 0 \\ 0 & 1 \end{pmatrix} d\mathbf{B}_t.$$

DANA-decay in optimizer coordinates

1. Continuize DANA-decay. In mode λ, use the coordinates

$$\Theta_{\lambda,t} = \langle \omega_\lambda, \Theta_t - \Theta^\star \rangle, \qquad Y_{\lambda,t} = \langle \omega_\lambda, Y_t \rangle.$$

With $s = 1 + t$, fixed γ, and $\kappa = 1/\alpha$, take

$$\gamma_1 = 1, \qquad \gamma_2 = \gamma, \qquad \gamma_3(t) = cs^{-\kappa}, \qquad \Delta(t) = \frac{\delta}{s}.$$

$$d\begin{pmatrix} \Theta_{\lambda,t} \\ Y_{\lambda,t} \end{pmatrix} = \begin{pmatrix} -\gamma\lambda & -cs^{-\kappa} \\ \lambda & -\delta/s \end{pmatrix} \begin{pmatrix} \Theta_{\lambda,t} \\ Y_{\lambda,t} \end{pmatrix} dt + \sqrt{\frac{\lambda \mathscr{P}(t)}{B}} \begin{pmatrix} -\gamma & 0 \\ 0 & 1 \end{pmatrix} d\mathbf{B}_t.$$

Three covariances produce a 3×3 propagator

2. Close the second moments.

$$\nu_\lambda(t) = \left(\mathbb{E}\Theta_{\lambda,t}^2, \mathbb{E}Y_{\lambda,t}^2, \mathbb{E}[\Theta_{\lambda,t}Y_{\lambda,t}]\right)^\top,$$

$$\dot{\nu}_\lambda = \underbrace{\begin{pmatrix} -2\gamma\lambda & 0 & -2cs^{-\kappa} \\ 0 & -2\delta/s & 2\lambda \\ \lambda & -cs^{-\kappa} & -\gamma\lambda - \delta/s \end{pmatrix}}_{\Omega_\lambda(t)} \nu_\lambda + \frac{\lambda \mathscr{P}(t)}{B} \begin{pmatrix} \gamma^2 \\ 1 \\ 0 \end{pmatrix}.$$

The 3×3 fundamental solution is therefore

$$\partial_t \Phi_\lambda(t,r) = \Omega_\lambda(t)\Phi_\lambda(t,r), \qquad \Phi_\lambda(r,r) = I_3.$$

Three covariances produce a 3×3 propagator

2. Close the second moments.

$$\nu_\lambda(t) = \left(\mathbb{E}\Theta_{\lambda,t}^2, \mathbb{E}Y_{\lambda,t}^2, \mathbb{E}[\Theta_{\lambda,t} Y_{\lambda,t}]\right)^\top,$$

$$\dot{\nu}_\lambda = \underbrace{\begin{pmatrix} -2\gamma\lambda & 0 & -2cs^{-\kappa} \\ 0 & -2\delta/s & 2\lambda \\ \lambda & -cs^{-\kappa} & -\gamma\lambda - \delta/s \end{pmatrix}}_{\Omega_\lambda(t)} \nu_\lambda + \frac{\lambda \mathscr{P}(t)}{B} \begin{pmatrix} \gamma^2 \\ 1 \\ 0 \end{pmatrix}.$$

The 3×3 fundamental solution is therefore

$$\partial_t \Phi_\lambda(t, r) = \Omega_\lambda(t)\Phi_\lambda(t, r), \qquad \Phi_\lambda(r, r) = I_3.$$

DANA-decay and NAG are different second-order equations

3. Lift the optimizer flow. If U_λ is the 2×2 fundamental solution of the homogeneous optimizer dynamics, then

$$\Phi_\lambda = \mathrm{Sym}^2(U_\lambda), \qquad (\Theta_\lambda, Y_\lambda) \longmapsto (\Theta_\lambda^2, Y_\lambda^2, \Theta_\lambda Y_\lambda).$$

Su–Boyd–Candès

Deterministic NAG (Su et al. 2016):

$$\ddot{x}_\lambda + \frac{3}{t}\dot{x}_\lambda + \lambda x_\lambda = 0.$$

DANA-decay

$s = 1 + t$, fixed γ, and $\kappa = 1/\alpha$:

$$\ddot{\Theta}_\lambda + \left(\gamma\lambda + \frac{\delta + \kappa}{s}\right)\dot{\Theta}_\lambda + \lambda\left(cs^{-\kappa} + \frac{\gamma(\delta + \kappa)}{s}\right)\Theta_\lambda = 0.$$

DANA-decay and NAG are different second-order equations

3. Lift the optimizer flow. If U_λ is the 2×2 fundamental solution of the homogeneous optimizer dynamics, then

$$\Phi_\lambda = \mathrm{Sym}^2(U_\lambda), \qquad (\Theta_\lambda, Y_\lambda) \longmapsto (\Theta_\lambda^2, Y_\lambda^2, \Theta_\lambda Y_\lambda).$$

Su–Boyd–Candès

Deterministic NAG (Su et al. 2016):

$$\boxed{\ddot{x}_\lambda + \frac{3}{t}\dot{x}_\lambda + \lambda x_\lambda = 0.}$$

DANA-decay

$s = 1 + t$, fixed γ, and $\kappa = 1/\alpha$:

$$\boxed{\ddot{\Theta}_\lambda + \left(\gamma\lambda + \frac{\delta+\kappa}{s}\right)\dot{\Theta}_\lambda + \lambda\left(cs^{-\kappa} + \frac{\gamma(\delta+\kappa)}{s}\right)\Theta_\lambda = 0.}$$

Acceleration is a change of clock

Theorem (Acceleration is a change of clock)

Ferbach, Everett, Gidel, E. Paquette, and C. Paquette.

Away from critical lines, at fixed batch size and for stable hyperparameters,

$$\boxed{\mathscr{P}_{\mathrm{alg}}(t; d) \asymp \mathscr{P}_{\mathrm{SGD}}(\vartheta_{\mathrm{alg}}(t); d).}$$

Each optimizer runs the same loss curve on a different clock

method	effective clock $\vartheta_{\mathrm{alg}}(t)$	consequence
SGD	$1+\gamma t$	linear time
SGD with fixed momentum	$1+\gamma_{\mathrm{eff}} t$	new constant, same powers
DANA-constant	$1+\gamma_2 t+\gamma_1\gamma_3 B t^2/d$	quadratic branch after $t \asymp d/B$
DANA-decaying	$1+\gamma_2 t+\gamma_1\gamma_3 t^{2-1/\alpha}$	superlinear clock for $\alpha > 1$

The clock tells us which slopes can steepen; the phase balance tells us whether that steepening changes the compute-optimal exponent.

Each optimizer runs the same loss curve on a different clock

method	effective clock $\vartheta_{\mathrm{alg}}(t)$	consequence
SGD	$1+\gamma t$	linear time
SGD with fixed momentum	$1+\gamma_{\mathrm{eff}} t$	new constant, same powers
DANA-constant	$1+\gamma_2 t+\gamma_1\gamma_3 B t^2/d$	quadratic branch after $t \asymp d/B$
DANA-decaying	$1+\gamma_2 t+\gamma_1\gamma_3 t^{2-1/\alpha}$	superlinear clock for $\alpha > 1$

The clock tells us which slopes can steepen; the phase balance tells us whether that steepening changes the compute-optimal exponent.

Only DANA creates a superlinear clock

method	effective clock	scaling consequence
SGD and SGD-M	$\vartheta(t) \asymp t$	fixed momentum only rescales a linear clock
DANA-constant	$\vartheta(t) \asymp \gamma_2 t + \gamma_1\gamma_3 B t^2/d$	acceleration activates after $t \asymp d/B$
DANA-decaying	$\vartheta(t) \asymp \gamma_2 t + \gamma_1\gamma_3 t^{2-1/\alpha}$	the clock is superlinear for $\alpha > 1$

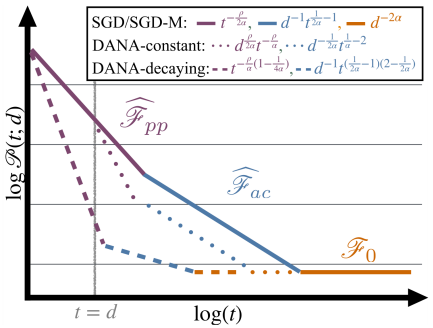
A faster late-time clock is necessary for outscaling, but it must become active before the compute-optimal balance is reached.

Only DANA creates a superlinear clock

method	effective clock	scaling consequence
SGD and SGD-M	$\vartheta(t) \asymp t$	fixed momentum only rescales a linear clock
DANA-constant	$\vartheta(t) \asymp \gamma_2 t + \gamma_1\gamma_3 B t^2/d$	acceleration activates after $t \asymp d/B$
DANA-decaying	$\vartheta(t) \asymp \gamma_2 t + \gamma_1\gamma_3 t^{2-1/\alpha}$	the clock is superlinear for $\alpha > 1$

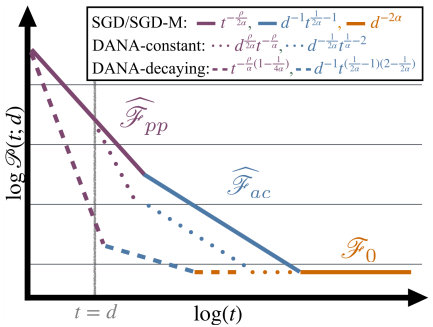
A faster late-time clock is necessary for outscaling, but it must become active before the compute-optimal balance is reached.

The loss components steepen without changing their identity



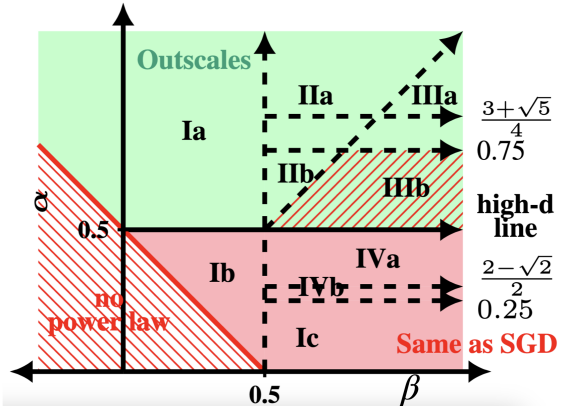
DANA steepens population and embedding bias; the optimizer-independent capacity floor remains fixed.

The loss components steepen without changing their identity



DANA steepens population and embedding bias; the optimizer-independent capacity floor remains fixed.

DANA outscales SGD above the trace-class line



Open questions

- **Below the trace-class line.** For $\alpha < 1$, the known schedules give no acceleration. Is that fundamental, or is there a schedule that accelerates?
- **The large-batch limit.** As $B \to \infty$, should the schedule deform continuously to Nesterov's accelerated gradient method?

Open questions

- **Below the trace-class line.** For $\alpha < 1$, the known schedules give no acceleration. Is that fundamental, or is there a schedule that accelerates?
- **The large-batch limit.** As $B \to \infty$, should the schedule deform continuously to Nesterov's accelerated gradient method?

Adapting the scheduling principle

Starting from AdamW

$$
\begin{aligned}
m_{t+1} &= \beta_{1,t} m_t + (1 - \beta_{1,t}) g_{t+1}, \\
v_{t+1} &= \beta_{2,t} v_t + (1 - \beta_{2,t}) g_{t+1}^{\odot 2}, \\
\theta_{t+1} &= (1 - \gamma_t \lambda_t)\, \theta_t - \gamma_t \frac{m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}.
\end{aligned}
$$

First moment

$\beta_{1,t}$ controls the memory horizon.

Second moment

v_{t+1} supplies coordinatewise normalization.

AdamW supplies the preconditioner and decoupled weight decay, but not DANA's explicit fresh-gradient channel (Loshchilov and Hutter 2019).

Starting from AdamW

$$
\begin{aligned}
m_{t+1} &= \beta_{1,t} m_t + (1 - \beta_{1,t}) g_{t+1}, \\
v_{t+1} &= \beta_{2,t} v_t + (1 - \beta_{2,t}) g_{t+1}^{\odot 2}, \\
\theta_{t+1} &= (1 - \gamma_t \lambda_t)\, \theta_t - \gamma_t \frac{m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}.
\end{aligned}
$$

First moment

$\beta_{1,t}$ controls the memory horizon.

Second moment

v_{t+1} supplies coordinatewise normalization.

AdamW supplies the preconditioner and decoupled weight decay, but not DANA's explicit fresh-gradient channel (Loshchilov and Hutter 2019).

Starting from AdamW

$$
\begin{aligned}
m_{t+1} &= \beta_{1,t} m_t + (1 - \beta_{1,t}) g_{t+1}, \\
v_{t+1} &= \beta_{2,t} v_t + (1 - \beta_{2,t}) g_{t+1}^{\odot 2}, \\
\theta_{t+1} &= (1 - \gamma_t \lambda_t)\, \theta_t - \gamma_t \frac{m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}.
\end{aligned}
$$

First moment

$\beta_{1,t}$ controls the memory horizon.

Second moment

v_{t+1} supplies coordinatewise normalization.

AdamW supplies the preconditioner and decoupled weight decay, but not DANA's explicit fresh-gradient channel (Loshchilov and Hutter 2019).

Starting from AdamW

$$
\begin{aligned}
m_{t+1} &= \beta_{1,t} m_t + (1 - \beta_{1,t}) g_{t+1}, \\
v_{t+1} &= \beta_{2,t} v_t + (1 - \beta_{2,t}) g_{t+1}^{\odot 2}, \\
\theta_{t+1} &= (1 - \gamma_t \lambda_t)\, \theta_t - \gamma_t \frac{m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}.
\end{aligned}
$$

First moment

$\beta_{1,t}$ controls the memory horizon.

Second moment

v_{t+1} supplies coordinatewise normalization.

AdamW supplies the preconditioner and decoupled weight decay, but not DANA's explicit fresh-gradient channel (Loshchilov and Hutter 2019).

Add the extra-gradient channel

$$
\begin{aligned}
m_{t+1} &= \beta_{1,t} m_t + (1-\beta_{1,t}) g_{t+1}, \\
v_{t+1} &= \beta_{2,t} v_t + (1-\beta_{2,t}) g_{t+1}^{\odot 2}, \\
\theta_{t+1} &= (1-\gamma_t \lambda_t)\theta_t - \gamma_t \frac{g_{t+1} + a_t m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}.
\end{aligned}
$$

(c.f. NADAMW)

$\beta_{1,t} \uparrow 1$ grows the memory horizon; a_t controls how strongly that memory moves the parameters.

This adapted AdamW update is **ADANA**: the extra gradient remains direct while the long-memory channel is scheduled separately.

Add the extra-gradient channel

$$
\begin{aligned}
m_{t+1} &= \beta_{1,t} m_t + (1-\beta_{1,t}) g_{t+1}, \\
v_{t+1} &= \beta_{2,t} v_t + (1-\beta_{2,t}) g_{t+1}^{\odot 2}, \\
\theta_{t+1} &= (1-\gamma_t \lambda_t)\theta_t - \gamma_t \frac{g_{t+1} + a_t m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}.
\end{aligned}
$$

(c.f. NADAMW)

$\beta_{1,t} \uparrow 1$ grows the memory horizon; a_t controls how strongly that memory moves the parameters.

This adapted AdamW update is **ADANA**: the extra gradient remains direct while the long-memory channel is scheduled separately.

Add the extra-gradient channel

$$
\begin{aligned}
m_{t+1} &= \beta_{1,t} m_t + (1-\beta_{1,t}) g_{t+1}, \\
v_{t+1} &= \beta_{2,t} v_t + (1-\beta_{2,t}) g_{t+1}^{\odot 2}, \\
\theta_{t+1} &= (1-\gamma_t \lambda_t)\theta_t - \gamma_t \frac{g_{t+1} + a_t m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}.
\end{aligned}
$$

(c.f. NADAMW)

$\beta_{1,t} \uparrow 1$ grows the memory horizon; a_t controls how strongly that memory moves the parameters.

This adapted AdamW update is **ADANA**: the extra gradient remains direct while the long-memory channel is scheduled separately.

Schedule memory on logarithmic time

$$1 - \beta_t \asymp \frac{\delta}{t+\delta}, \qquad \tau_{\mathrm{mem}}(t) \asymp t + \delta.$$

Doubling training age doubles the memory horizon and retained history.

In the unnormalized-accumulator convention, damp the long-memory channel by

$$a(t) \asymp (1+t)^{1-\kappa}, \qquad 0 < \kappa < 1,$$

Smaller κ is more aggressive; larger κ is more conservative; $\kappa = 1$ approaches baseline-like scaling.

Schedule memory on logarithmic time

$$1 - \beta_t \asymp \frac{\delta}{t+\delta}, \qquad \tau_{\mathrm{mem}}(t) \asymp t + \delta.$$

Doubling training age doubles the memory horizon and retained history.

In the unnormalized-accumulator convention, damp the long-memory channel by

$$a(t) \asymp (1+t)^{1-\kappa}, \qquad 0 < \kappa < 1,$$

Smaller κ is more aggressive; larger κ is more conservative; $\kappa = 1$ approaches baseline-like scaling.

Schedule memory on logarithmic time

$$1 - \beta_t \asymp \frac{\delta}{t+\delta}, \qquad \tau_{\mathrm{mem}}(t) \asymp t + \delta.$$

Doubling training age doubles the memory horizon and retained history.

In the unnormalized-accumulator convention, damp the long-memory channel by

$$a(t) \asymp (1+t)^{1-\kappa}, \qquad 0 < \kappa < 1,$$

Smaller κ is more aggressive; larger κ is more conservative; $\kappa = 1$ approaches baseline-like scaling.

Schedule memory on logarithmic time

$$1 - \beta_t \asymp \frac{\delta}{t+\delta}, \qquad \tau_{\mathrm{mem}}(t) \asymp t + \delta.$$

Doubling training age doubles the memory horizon and retained history.

In the unnormalized-accumulator convention, damp the long-memory channel by

$$a(t) \asymp (1+t)^{1-\kappa}, \qquad 0 < \kappa < 1,$$

Smaller κ is more aggressive; larger κ is more conservative; $\kappa = 1$ approaches baseline-like scaling.

Anatomy of a Chinchilla-style experiment

control	experimental prescription
model shape	width and depth grow proportionally; $d_{\text{head}} = 64$, $h = d_{\text{model}}/64$
normalization	pre-LN; QK-LayerNorm before RoPE
initialization	fan-in scaling; residual outputs scaled with depth
data and batch	sequence 2048; batch $32,256$; $N = 20D$ tokens
schedule	2% warmup; cosine decay to 10% of peak; peak rate tuned per optimizer

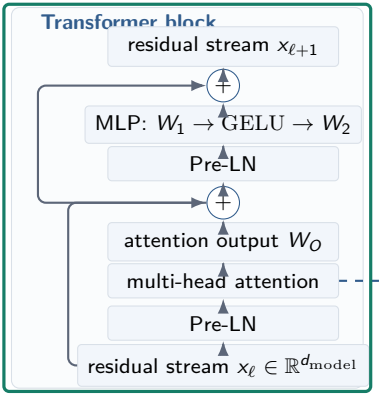
Every row is part of the controlled scaling experiment.

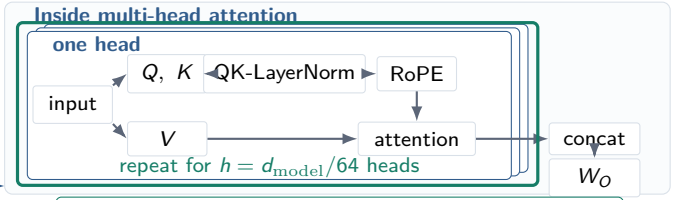
Anatomy of a Chinchilla-style experiment

control	experimental prescription
model shape	width and depth grow proportionally; $d_{\text{head}} = 64$, $h = d_{\text{model}}/64$
normalization	pre-LN; QK-LayerNorm before RoPE
initialization	fan-in scaling; residual outputs scaled with depth
data and batch	sequence 2048; batch $32,256$; $N = 20D$ tokens
schedule	2% warmup; cosine decay to 10% of peak; peak rate tuned per optimizer

First: what does proportional width–depth scaling mean architecturally?

Anatomy of a Chinchilla-style experiment



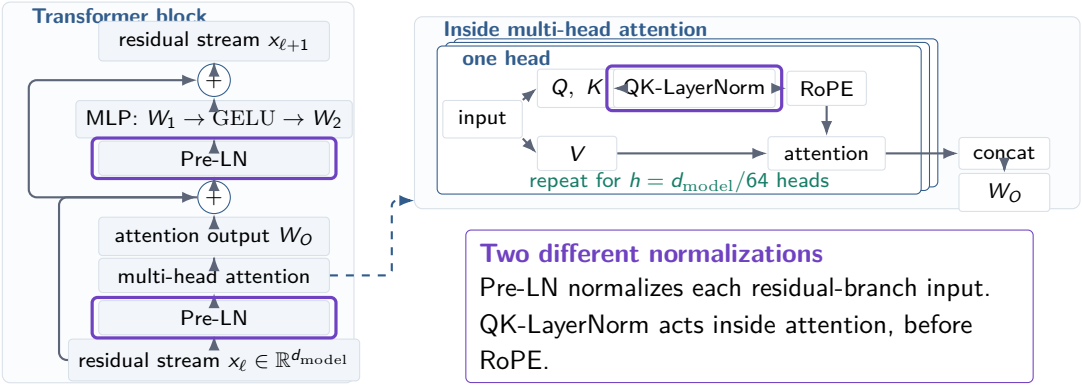


The scale coordinate

d_{model} and the number of blocks L grow proportionally.

Head width stays fixed, so $h = d_{\mathrm{model}}/64$ grows with width.

Anatomy of a Chinchilla-style experiment

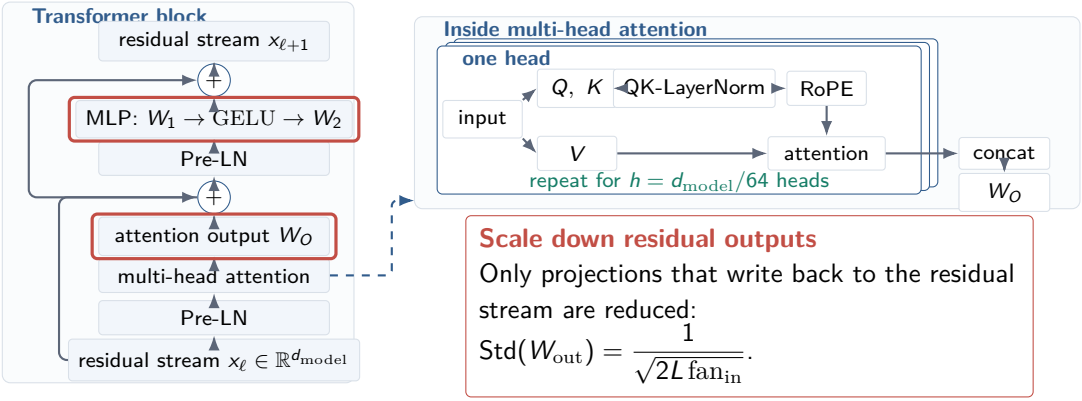


Two different normalizations

Pre-LN normalizes each residual-branch input.

QK-LayerNorm acts inside attention, before RoPE.

Anatomy of a Chinchilla-style experiment



Scale down residual outputs

Only projections that write back to the residual stream are reduced:

$$\mathrm{Std}(W_{\text{out}}) = \frac{1}{\sqrt{2L\,\mathrm{fan}_{\text{in}}}}.$$

Anatomy of a Chinchilla-style experiment

control	experimental prescription
model shape	width and depth grow proportionally; $d_{\text{head}} = 64$, $h = d_{\text{model}}/64$
normalization	pre-LN; QK-LayerNorm before RoPE
initialization	fan-in scaling; residual outputs scaled with depth
data and batch	sequence 2048; batch $32,256$; $N = 20D$ tokens
schedule	2% warmup; cosine decay to 10% of peak; peak rate tuned per optimizer

Finally: the data allocation and optimizer schedule must scale too.

Compare optimizers at equal loss

Let optimizer B use compute C_B. Interpolate the baseline curve to find the compute C_A needed to reach the same validation loss:

$$\text{compute multiplier} = \frac{C_A}{C_B}, \qquad \text{relative saving} = \frac{C_A - C_B}{C_B}.$$

$C_A/C_B = 1$

No compute advantage.

$C_A/C_B = 1.4$

The baseline needs 40% more compute at equal loss.

This comparison remains meaningful even when neither curve supports a credible single asymptotic power law.

Compare optimizers at equal loss

Let optimizer B use compute C_B. Interpolate the baseline curve to find the compute C_A needed to reach the same validation loss:

$$\text{compute multiplier} = \frac{C_A}{C_B}, \qquad \text{relative saving} = \frac{C_A - C_B}{C_B}.$$

$C_A/C_B = 1$

No compute advantage.

$C_A/C_B = 1.4$

The baseline needs 40% more compute at equal loss.

This comparison remains meaningful even when neither curve supports a credible single asymptotic power law.

Compare optimizers at equal loss

Let optimizer B use compute C_B. Interpolate the baseline curve to find the compute C_A needed to reach the same validation loss:

$$\text{compute multiplier} = \frac{C_A}{C_B}, \qquad \text{relative saving} = \frac{C_A - C_B}{C_B}.$$

$C_A/C_B = 1$

No compute advantage.

$C_A/C_B = 1.4$

The baseline needs 40% more compute at equal loss.

This comparison remains meaningful even when neither curve supports a credible single asymptotic power law.

Compare optimizers at equal loss

Let optimizer B use compute C_B. Interpolate the baseline curve to find the compute C_A needed to reach the same validation loss:

$$\text{compute multiplier} = \frac{C_A}{C_B}, \qquad \text{relative saving} = \frac{C_A - C_B}{C_B}.$$

$C_A/C_B = 1$

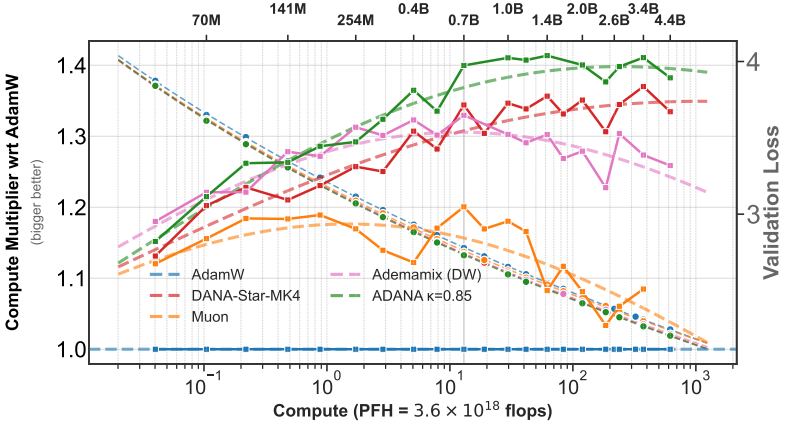
No compute advantage.

$C_A/C_B = 1.4$

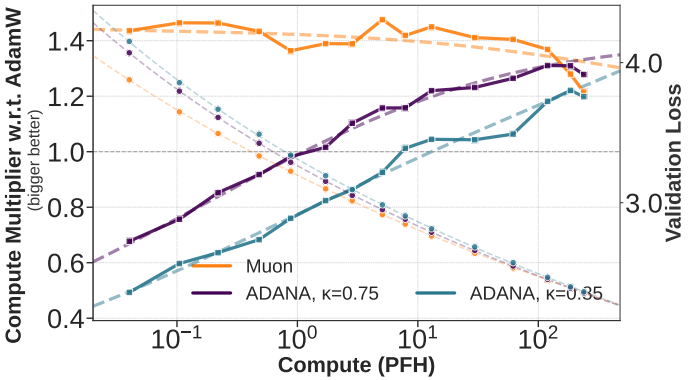
The baseline needs 40% more compute at equal loss.

This comparison remains meaningful even when neither curve supports a credible single asymptotic power law.

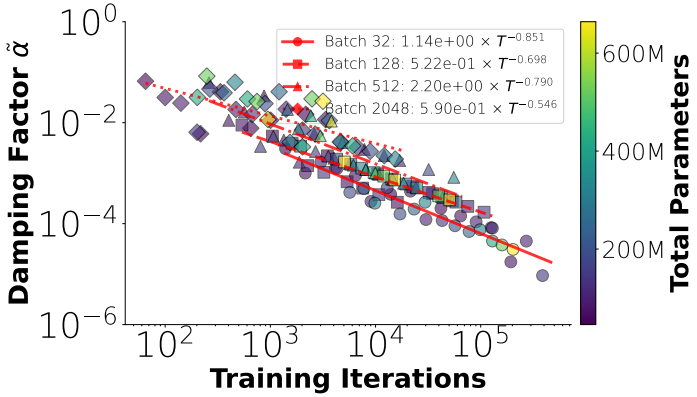
Main experimental result 1: Batch 32



Main experimental result 2: Batch 256

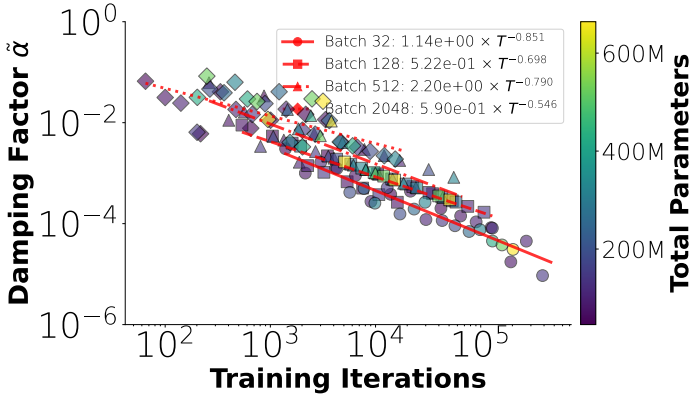


Batch size changes the fitted damping law



Both the prefactor and exponent move with batch. We do not yet know the right joint schedule $\widetilde{\alpha}^{\star}(T, B)$.

Batch size changes the fitted damping law



Both the prefactor and exponent move with batch. We do not yet know the right joint schedule $\widetilde{\alpha}^{\star}(T, B)$.

Lessons

Lesson 1: The break is parallel across optimizers

A two-power surrogate captures the bend:

$$L(C) = a + bC^{-c} + eC^{-f}.$$

Before the break

Advantages grow across part of the ladder: finite-range outscaling.

After the break

All four optimizers bend in roughly the same region and have similar large-compute exponents.

The shared break signals a new dominant loss component or regime; it does not erase the measured savings.

Related crossover model: Caballero et al. (2023).

Lesson 1: The break is parallel across optimizers

A two-power surrogate captures the bend:

$$L(C) = a + bC^{-c} + eC^{-f}.$$

Before the break

Advantages grow across part of the ladder: finite-range outscaling.

After the break

All four optimizers bend in roughly the same region and have similar large-compute exponents.

The shared break signals a new dominant loss component or regime; it does not erase the measured savings.

Related crossover model: Caballero et al. (2023).

Lesson 1: The break is parallel across optimizers

A two-power surrogate captures the bend:

$$L(C) = a + bC^{-c} + eC^{-f}.$$

Before the break

Advantages grow across part of the ladder: finite-range outscaling.

After the break

All four optimizers bend in roughly the same region and have similar large-compute exponents.

The shared break signals a new dominant loss component or regime; it does not erase the measured savings.

Related crossover model: Caballero et al. (2023).

Lesson 1: The break is parallel across optimizers

A two-power surrogate captures the bend:

$$L(C) = a + bC^{-c} + eC^{-f}.$$

Before the break

Advantages grow across part of the ladder: finite-range outscaling.

After the break

All four optimizers bend in roughly the same region and have similar large-compute exponents.

The shared break signals a new dominant loss component or regime; it does not erase the measured savings.

Related crossover model: Caballero et al. (2023).

Lesson 1: The break is parallel across optimizers

A two-power surrogate captures the bend:

$$L(C) = a + bC^{-c} + eC^{-f}.$$

Before the break

Advantages grow across part of the ladder: finite-range outscaling.

After the break

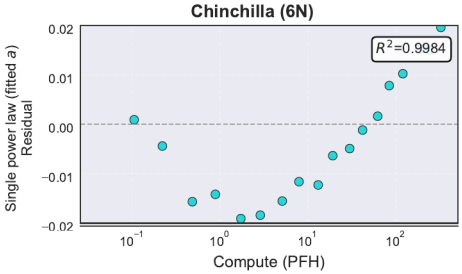
All four optimizers bend in roughly the same region and have similar large-compute exponents.

The shared break signals a new dominant loss component or regime; it does not erase the measured savings.

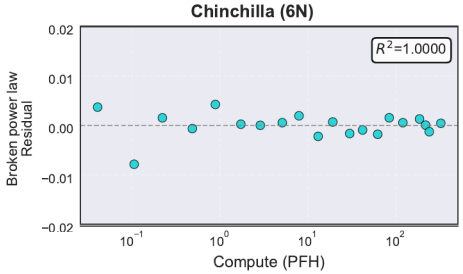
Related crossover model: Caballero et al. (2023).

Lesson 1: Residuals expose the shared break

Single power law



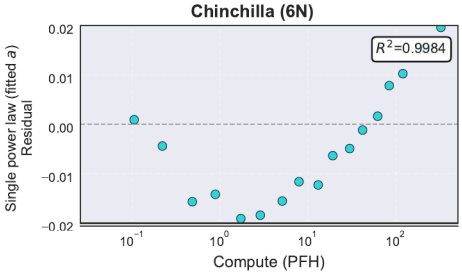
Broken power law



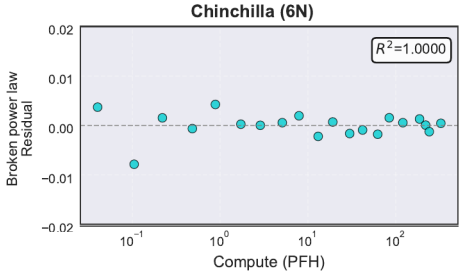
The single-power fit leaves a coherent arch despite high R^2; the broken law removes nearly all structured scale dependence.

Lesson 1: Residuals expose the shared break

Single power law

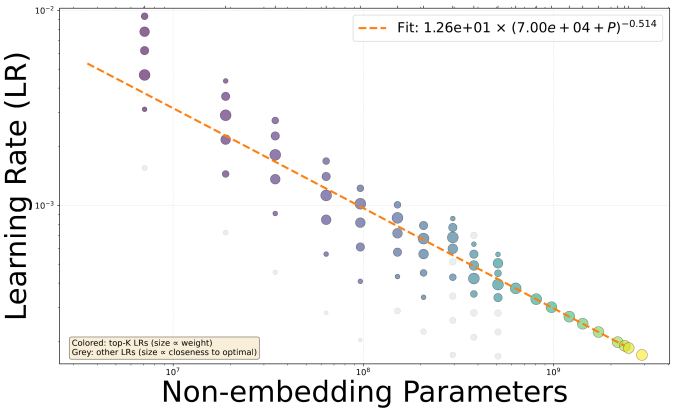


Broken power law



The single-power fit leaves a coherent arch despite high R^2; the broken law removes nearly all structured scale dependence.

Lesson 2: Baselining must scale with the model



At each calibrated model size:

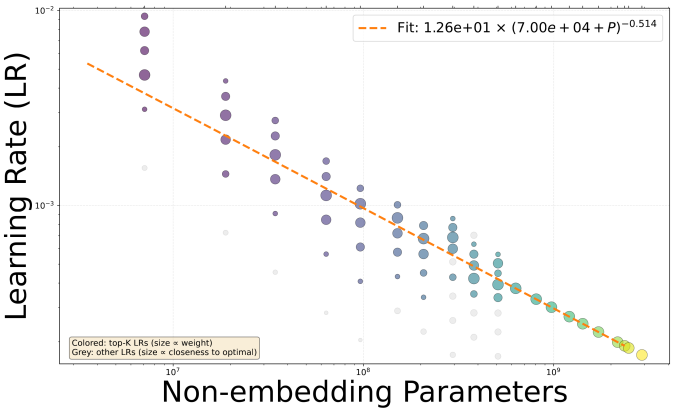
- sweep learning rates;
- retain the top $K = 5$;
- weight by validation loss.

Then fit and extrapolate

$$\gamma^*(P) = a(b + P)^d.$$

Fair optimizer comparisons require a learning-rate rule, not one learning rate.

Lesson 2: Baselining must scale with the model



At each calibrated model size:

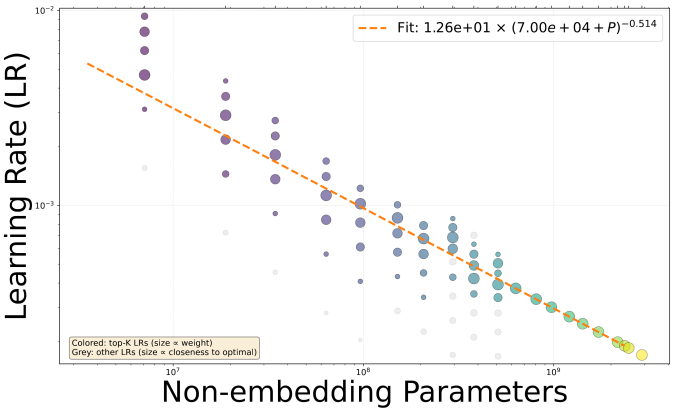
- sweep learning rates;
- retain the top $K = 5$;
- weight by validation loss.

Then fit and extrapolate

$$\gamma^*(P) = a(b + P)^d.$$

Fair optimizer comparisons require a learning-rate rule, not one learning rate.

Lesson 2: Baselining must scale with the model



At each calibrated model size:

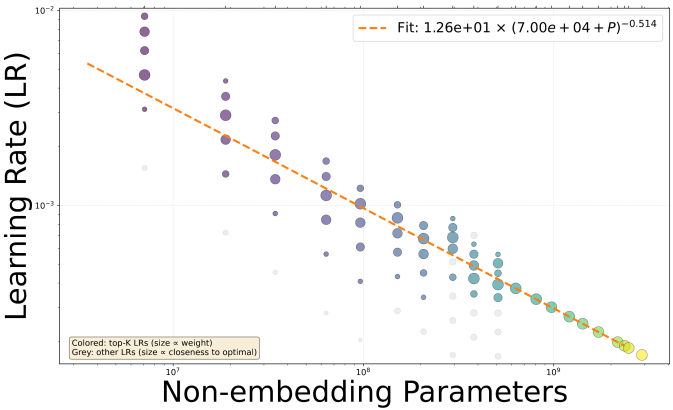
- sweep learning rates;
- retain the top $K = 5$;
- weight by validation loss.

Then fit and extrapolate

$$\gamma^*(P) = a(b + P)^d.$$

Fair optimizer comparisons require a learning-rate rule, not one learning rate.

Lesson 2: Baselining must scale with the model



At each calibrated model size:

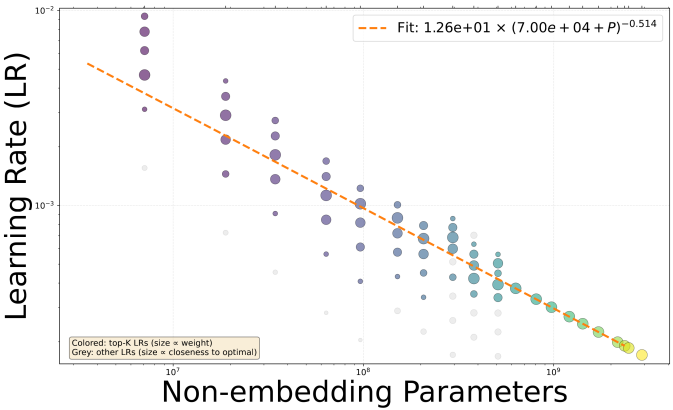
- sweep learning rates;
- retain the top $K = 5$;
- weight by validation loss.

Then fit and extrapolate

$$\gamma^*(P) = a(b + P)^d.$$

Fair optimizer comparisons require a learning-rate rule, not one learning rate.

Lesson 2: Baselining must scale with the model



At each calibrated model size:

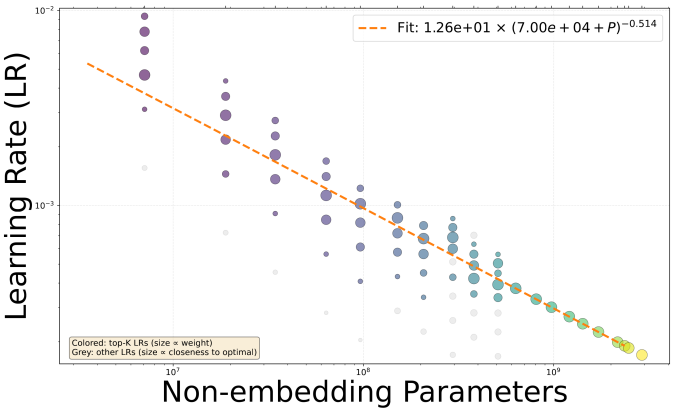
- sweep learning rates;
- retain the top $K = 5$;
- weight by validation loss.

Then fit and extrapolate

$$\gamma^*(P) = a(b + P)^d.$$

Fair optimizer comparisons require a learning-rate rule, not one learning rate.

Lesson 2: Baselining must scale with the model



At each calibrated model size:

- sweep learning rates;
- retain the top $K = 5$;
- weight by validation loss.

Then fit and extrapolate

$$\gamma^*(P) = a(b + P)^d.$$

Fair optimizer comparisons require a learning-rate rule, not one learning rate.

Lesson 2: A 50% learning-rate miss costs about 3% compute

$$\Delta L \approx \bar{\zeta}(\log q)^2, \qquad \bar{\zeta} = 1.93 \times 10^{-2}, \qquad \Delta \log C \approx \frac{\Delta L}{s(L-a)}.$$

At the 2.4B-parameter point, $s(L-a) \approx 0.099$ nats per unit log-compute.

learning-rate factor q	loss penalty	implied compute penalty
1.5	0.00317 nats	$\approx 3.2\%$
2	0.00927 nats	$\approx 9.8\%$

Baselining matters, but this local curvature cannot explain a reported 30%–40% compute gain.

Lesson 2: A 50% learning-rate miss costs about 3% compute

$$\Delta L \approx \bar{\zeta}(\log q)^2, \qquad \bar{\zeta} = 1.93 \times 10^{-2}, \qquad \Delta \log C \approx \frac{\Delta L}{s(L-a)}.$$

At the 2.4B-parameter point, $s(L-a) \approx 0.099$ nats per unit log-compute.

learning-rate factor q	loss penalty	implied compute penalty
1.5	0.00317 nats	$\approx 3.2\%$
2	0.00927 nats	$\approx 9.8\%$

Baselining matters, but this local curvature cannot explain a reported 30%–40% compute gain.

Lesson 2: A 50% learning-rate miss costs about 3% compute

$$\Delta L \approx \bar{\zeta}(\log q)^2, \qquad \bar{\zeta} = 1.93 \times 10^{-2}, \qquad \Delta \log C \approx \frac{\Delta L}{s(L-a)}.$$

At the 2.4B-parameter point, $s(L-a) \approx 0.099$ nats per unit log-compute.

learning-rate factor q	loss penalty	implied compute penalty
1.5	0.00317 nats	$\approx 3.2\%$
2	0.00927 nats	$\approx 9.8\%$

Baselining matters, but this local curvature cannot explain a reported 30%–40% compute gain.

Lesson 2: A 50% learning-rate miss costs about 3% compute

$$\Delta L \approx \bar{\zeta}(\log q)^2, \qquad \bar{\zeta} = 1.93 \times 10^{-2}, \qquad \Delta \log C \approx \frac{\Delta L}{s(L-a)}.$$

At the 2.4B-parameter point, $s(L-a) \approx 0.099$ nats per unit log-compute.

learning-rate factor q	loss penalty	implied compute penalty
1.5	0.00317 nats	$\approx 3.2\%$
2	0.00927 nats	$\approx 9.8\%$

Baselining matters, but this local curvature cannot explain a reported 30%–40% compute gain.

Lesson 3: Weight decay belongs on the logarithmic clock

$$\lambda_{\text{const}}(t) = \frac{\omega}{T} \qquad \lambda_{\Omega}(t) = \frac{\omega}{T/\Omega + t}$$

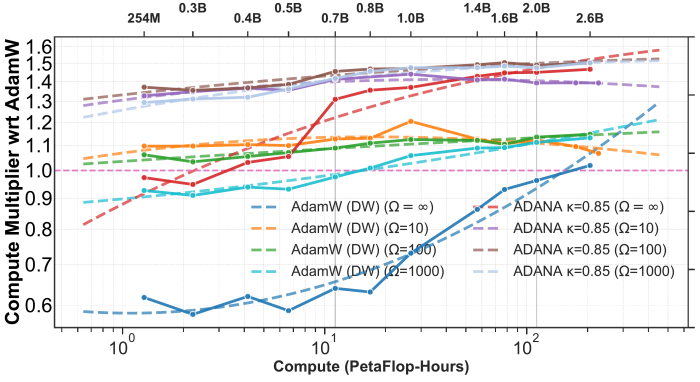
If momentum is organized in logarithmic time, constant step-time weight decay cannot scale uniformly with it.

Lesson 3: Weight decay belongs on the logarithmic clock

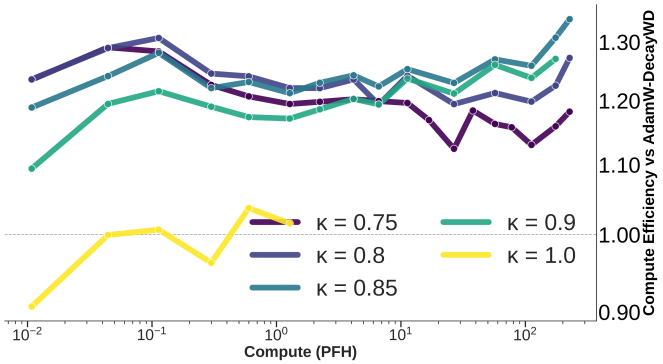
$$\lambda_{\text{const}}(t) = \frac{\omega}{T} \qquad \lambda_{\Omega}(t) = \frac{\omega}{T/\Omega + t}$$

If momentum is organized in logarithmic time, constant step-time weight decay cannot scale uniformly with it.

$T/10$ ablation

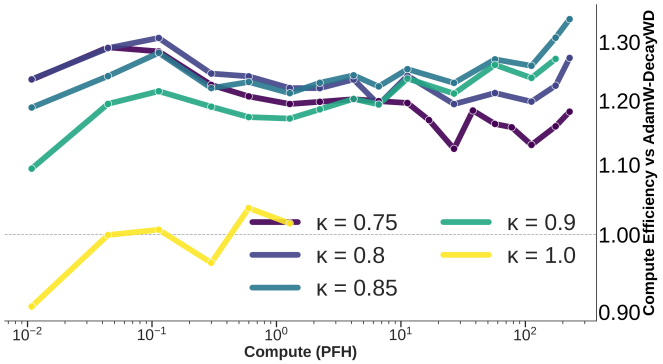


Lesson 4: Damping is approximately stable across this ladder



An intermediate κ near 0.85 transfers across the measured scales; this is empirical transfer, not yet a universal batch rule.

Lesson 4: Damping is approximately stable across this ladder



An intermediate κ near 0.85 transfers across the measured scales; this is empirical transfer, not yet a universal batch rule.

Post-mortem: outscaling becomes a multiplicative gap

PLRF at fixed batch

- DANA changes the compute-optimal exponent relative to SGD.
- Outscaling is provable.

Transformer experiment

- Before $\sim$ 1B parameters, the gap grows with scale.
- Near $\sim$ 1B, both curves share an exponent change.
- After the break, the slopes are nearly parallel: the gain is multiplicative.

The measured ladder does not support one global transformer outscaling law. It sharpens the need for a theory of $\widetilde{\alpha}^\star(T, B)$, not only $\widetilde{\alpha}^\star(T)$.

Post-mortem: outscaling becomes a multiplicative gap

PLRF at fixed batch

- DANA changes the compute-optimal exponent relative to SGD.
- Outscaling is provable.

Transformer experiment

- Before $\sim$ 1B parameters, the gap grows with scale.
- Near $\sim$ 1B, both curves share an exponent change.
- After the break, the slopes are nearly parallel: the gain is multiplicative.

The measured ladder does not support one global transformer outscaling law. It sharpens the need for a theory of $\widetilde{\alpha}^\star(T, B)$, not only $\widetilde{\alpha}^\star(T)$.

Post-mortem: outscaling becomes a multiplicative gap

PLRF at fixed batch

- DANA changes the compute-optimal exponent relative to SGD.
- Outscaling is provable.

Transformer experiment

- Before $\sim$ 1B parameters, the gap grows with scale.
- Near $\sim$ 1B, both curves share an exponent change.
- After the break, the slopes are nearly parallel: the gain is multiplicative.

The measured ladder does not support one global transformer outscaling law. It sharpens the need for a theory of $\widetilde{\alpha}^\star(T, B)$, not only $\widetilde{\alpha}^\star(T)$.

Post-mortem: outscaling becomes a multiplicative gap

PLRF at fixed batch

- DANA changes the compute-optimal exponent relative to SGD.
- Outscaling is provable.

Transformer experiment

- Before $\sim$ 1B parameters, the gap grows with scale.
- Near $\sim$ 1B, both curves share an exponent change.
- After the break, the slopes are nearly parallel: the gain is multiplicative.

The measured ladder does not support one global transformer outscaling law. It sharpens the need for a theory of $\widetilde{\alpha}^{\star}(T, B)$, not only $\widetilde{\alpha}^{\star}(T)$.

Post-mortem: outscaling becomes a multiplicative gap

PLRF at fixed batch

- DANA changes the compute-optimal exponent relative to SGD.
- Outscaling is provable.

Transformer experiment

- Before $\sim$ 1B parameters, the gap grows with scale.
- Near $\sim$ 1B, both curves share an exponent change.
- After the break, the slopes are nearly parallel: the gain is multiplicative.

The measured ladder does not support one global transformer outscaling law. It sharpens the need for a theory of $\widetilde{\alpha}^{\star}(T, B)$, not only $\widetilde{\alpha}^{\star}(T)$.

Post-mortem: outscaling becomes a multiplicative gap

PLRF at fixed batch

- DANA changes the compute-optimal exponent relative to SGD.
- Outscaling is provable.

Transformer experiment

- Before $\sim$ 1B parameters, the gap grows with scale.
- Near $\sim$ 1B, both curves share an exponent change.
- After the break, the slopes are nearly parallel: the gain is multiplicative.

The measured ladder does not support one global transformer outscaling law. It sharpens the need for a theory of $\widetilde{\alpha}^{\star}(T, B)$, not only $\widetilde{\alpha}^{\star}(T)$.

Post-mortem: outscaling becomes a multiplicative gap

PLRF at fixed batch

- DANA changes the compute-optimal exponent relative to SGD.
- Outscaling is provable.

Transformer experiment

- Before $\sim$ 1B parameters, the gap grows with scale.
- Near $\sim$ 1B, both curves share an exponent change.
- After the break, the slopes are nearly parallel: the gain is multiplicative.

The measured ladder does not support one global transformer outscaling law. It sharpens the need for a theory of $\widetilde{\alpha}^{\star}(T, B)$, not only $\widetilde{\alpha}^{\star}(T)$.

Post-mortem: outscaling becomes a multiplicative gap

PLRF at fixed batch

- DANA changes the compute-optimal exponent relative to SGD.
- Outscaling is provable.

Transformer experiment

- Before $\sim$ 1B parameters, the gap grows with scale.
- Near $\sim$ 1B, both curves share an exponent change.
- After the break, the slopes are nearly parallel: the gain is multiplicative.

The measured ladder does not support one global transformer outscaling law. It sharpens the need for a theory of $\widetilde{\alpha}^{\star}(T, B)$, not only $\widetilde{\alpha}^{\star}(T)$.

Further reading

- Weijie Su, Stephen Boyd, and Emmanuel Candès. "A Differential Equation for Modeling Nesterov's Accelerated Gradient Method." JMLR 2016. JMLR 17(153)
- Nolan Dey et al. "Don't Be Lazy: CompleteP Enables Compute-Efficient Deep Transformers." NeurIPS 2025. arXiv:2505.01618
- Mitchell Wortsman et al. "Small-Scale Proxies for Large-Scale Transformer Training Instabilities." ICLR 2024. arXiv:2309.14322
- Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. "Benchmarking Optimizers for Large Language Model Pretraining." 2025. arXiv:2509.01440

Further reading

- Weijie Su, Stephen Boyd, and Emmanuel Candès. "A Differential Equation for Modeling Nesterov's Accelerated Gradient Method." JMLR 2016. JMLR 17(153)
- Nolan Dey et al. "Don't Be Lazy: CompleteP Enables Compute-Efficient Deep Transformers." NeurIPS 2025. arXiv:2505.01618
- Mitchell Wortsman et al. "Small-Scale Proxies for Large-Scale Transformer Training Instabilities." ICLR 2024. arXiv:2309.14322
- Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. "Benchmarking Optimizers for Large Language Model Pretraining." 2025. arXiv:2509.01440

Further reading

- Weijie Su, Stephen Boyd, and Emmanuel Candès. "A Differential Equation for Modeling Nesterov's Accelerated Gradient Method." JMLR 2016. JMLR 17(153)
- Nolan Dey et al. "Don't Be Lazy: CompleteP Enables Compute-Efficient Deep Transformers." NeurIPS 2025. arXiv:2505.01618
- Mitchell Wortsman et al. "Small-Scale Proxies for Large-Scale Transformer Training Instabilities." ICLR 2024. arXiv:2309.14322
- Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. "Benchmarking Optimizers for Large Language Model Pretraining." 2025. arXiv:2509.01440

Further reading

- Weijie Su, Stephen Boyd, and Emmanuel Candès. "A Differential Equation for Modeling Nesterov's Accelerated Gradient Method." JMLR 2016. JMLR 17(153)
- Nolan Dey et al. "Don't Be Lazy: CompleteP Enables Compute-Efficient Deep Transformers." NeurIPS 2025. arXiv:2505.01618
- Mitchell Wortsman et al. "Small-Scale Proxies for Large-Scale Transformer Training Instabilities." ICLR 2024. arXiv:2309.14322
- Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. "Benchmarking Optimizers for Large Language Model Pretraining." 2025. arXiv:2509.01440

Open questions

- **The shared break.** What mechanism causes it? Can a controlled change move or remove it? Detailed open scaling ladders would make this testable.
- **Batch interpolation.** What schedule connects full-batch Nesterov to stochastic DANA as batch size and the number of steps vary, and does dimension also enter?
- **Below $\alpha = 1$.** Why do the known stochastic schedules fail to accelerate there when deterministic Nesterov has no analogous barrier?
- **The exponent $\kappa \approx 0.85$.** Which language power law explains it: temporal decay, the Hilberg exponent, or something else?

Open questions

- **The shared break.** What mechanism causes it? Can a controlled change move or remove it? Detailed open scaling ladders would make this testable.
- **Batch interpolation.** What schedule connects full-batch Nesterov to stochastic DANA as batch size and the number of steps vary, and does dimension also enter?
- **Below $\alpha = 1$.** Why do the known stochastic schedules fail to accelerate there when deterministic Nesterov has no analogous barrier?
- **The exponent $\kappa \approx 0.85$.** Which language power law explains it: temporal decay, the Hilberg exponent, or something else?

Open questions

- **The shared break.** What mechanism causes it? Can a controlled change move or remove it? Detailed open scaling ladders would make this testable.
- **Batch interpolation.** What schedule connects full-batch Nesterov to stochastic DANA as batch size and the number of steps vary, and does dimension also enter?
- **Below $\alpha = 1$.** Why do the known stochastic schedules fail to accelerate there when deterministic Nesterov has no analogous barrier?
- **The exponent $\kappa \approx 0.85$.** Which language power law explains it: temporal decay, the Hilberg exponent, or something else?

Open questions

- **The shared break.** What mechanism causes it? Can a controlled change move or remove it? Detailed open scaling ladders would make this testable.
- **Batch interpolation.** What schedule connects full-batch Nesterov to stochastic DANA as batch size and the number of steps vary, and does dimension also enter?
- **Below $\alpha = 1$.** Why do the known stochastic schedules fail to accelerate there when deterministic Nesterov has no analogous barrier?
- **The exponent $\kappa \approx 0.85$.** Which language power law explains it: temporal decay, the Hilberg exponent, or something else?