LOGARITHMIC-TIME SCHEDULES FOR SCALING LANGUAGE MODELS WITH MOMENTUM

Anonymous authors
Paper under double-blind review

ABSTRACT

In practice, the hyperparameters (β_1, β_2) and weight-decay λ in ADAMW are typically kept at fixed values. Is there any reason to do otherwise? We show that for large-scale language model training, the answer is yes: by exploiting the power-law structure of language data, one can design time-varying schedules for $(\beta_1, \beta_2, \lambda)$ that deliver substantial performance gains. We study *logarithmic-time scheduling*, in which the optimizer's gradient memory horizon grows with training time. Although naïve variants of this are unstable, we show that suitable damping mechanisms restore stability while preserving the benefits of longer memory. Based on this, we present ADANA, an ADAMW-like optimizer that couples log-time schedules with explicit damping to balance stability and performance. We empirically evaluate ADANA across transformer scalings (45M to 4.4B parameters), comparing against ADAMW, MUON, and ADEMAMIX. When properly tuned, ADANA achieves up to 40% compute efficiency relative to a tuned ADAMW, with gains that persist—and even improve—as model scale increases. We further show that similar benefits arise when applying logarithmic-time scheduling to ADEMAMIX, and that logarithmic-time weight-decay alone can yield significant improvements. Finally, we present variants of ADANA that mitigate potential failure modes and improve robustness.

1 INTRODUCTION

A key empirical finding in language models is the emergence of scaling laws, which predict that the training loss decays as a power law in available compute C, data N, and model size D (Kaplan et al., 2020). In this work, we study optimization algorithms within the compute-optimal scaling regime for training language models (Hoffmann et al., 2022).

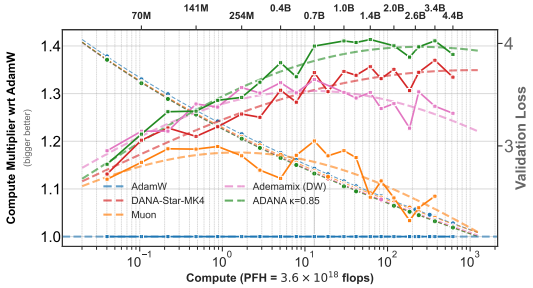


Figure 1: **Scaling laws and compute multiplier vs compute** C for ADANA variants with transformers (architecture in Tab. 1) on FineWeb. **Left axis:** compute-multiplier relative to ADAMW. ADANA and DANA-STAR-MK4 ($\kappa = 0.85$) benefits increase with scale; $\sim$40% compute savings. ADEMAMIX (DW)[1](configured at $\kappa = 0.85$; Sec. J), ADANA ($\kappa = 0.75$), and MUON outperform ADAMW but with diminishing gains at larger scales; increasing κ to 0.85 improves scaling. **Right axis:** validation loss fit to a broken power-law $L = a + bC^{-c} + eC^{-f}$ with shared saturation a across optimizers (Sec. E.1).

Scaling strategies differ fundamentally from standard hyperparameter tuning: hyperparameters cannot be re-swept at every scale without incurring prohibitive computational cost. Instead, it is essential to identify *functional forms* that prescribe how optimization hyperparameters should scale with model size D, enabling reliable extrapolation from small to large models. A desirable property of functional forms is that performance[2] at large scale matches or improves upon that observed at small scale. Fig. 1 illustrates this principle: algorithms with comparable small-scale performance can behave very differently under scaling—some degrade, while others maintain or improve performance. This divergence highlights the importance of scale-aware functional forms. While most prior work has focused on learning rate (LR) scaling (e.g., Yang et al. (2021)), we instead study the momentum parameters (β_1, β_2) and weight-decay λ in ADAMW, which are typically treated as fixed constants.

[1]We specify "(DW)" when an optimizer outside the ADANA class uses decaying weight-decay; See Sec. 2.3.

[2]Performance is measured by compute-multiplier and compute-efficiency (higher better; 1.4 is 40% savings in compute). For a target loss reached by algorithms A and B using respectively the compute C^{A}, and C^{B}, the ***compute-multiplier*** of B w.r.t A is $C^{\mathrm{A}}/C^{\mathrm{B}}$ and ***compute-efficiency*** is $(C^{\mathrm{A}} - C^{\mathrm{B}})/C^{\mathrm{B}}$; Sec C.2.1 for discussion and (Qiu et al., 2025a) for original usage.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Algorithm 1 ADANA
(Adaptive DAmped Nesterov Accel.)

Require: Parameters θ_0, $m_0 = 0$, $v_0 = 0$, peak LR $\gamma^* > 0$, LR schedule $\gamma(t)$, $\epsilon > 0$
Require: **(Specific to ADANA)** $\delta > 0$, $\kappa \in (0, 1)$, $\omega > 0$
Define schedules:
  $\beta_1(t) = \beta_2(t) = 1 - \delta/\delta+t$
  $\lambda(t) = \omega/t$
  $\alpha(t) = \tilde{\alpha}(1+t)^{1-\kappa}$
for $t = 0, \ldots, T-1$ **do**
  Sample minibatch
  Compute g_{t+1}
  $m_{t+1} = \beta_1(t) m_t + (1 - \beta_1(t)) g_{t+1}$
  $v_{t+1} = \beta_2(t) v_t + (1 - \beta_2(t)) g_{t+1}^2$
  $\theta_{t+1} = \theta_t - \gamma(t)\left(\gamma^* \frac{g_{t+1} + \alpha(t) m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} + \lambda(t)\theta_t\right)$

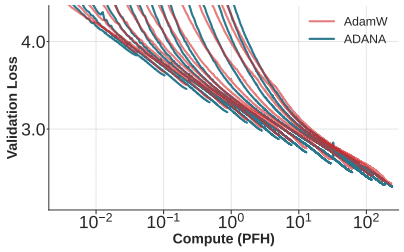


Figure 2: **Validation loss curves** across scales from 45.7M to 2.62B parameters for ADAMW and ADANA ($\kappa = 0.85$) on FineWeb Penedo et al. (2024); Train at compute-optimal scaling $D = 20N$ where N = # of params. and D = # tokens; Final validation loss follows scaling law Hoffmann et al. (2022). ADANA shows better loss than ADAMW.

A good scaling strategy should leverage the intrinsic structures in language. Beyond the well-known Zipfian distribution of token frequencies (Piantadosi, 2014), the next-token prediction problem itself exhibits power-law structure. Information-theoretic studies of natural language Shannon (1951); Hilberg (1990) show that the per-token entropy—the uncertainty in predicting the next token given a context of length T—decays as $T^{\beta-1}$, where $\beta \approx 0.88$ is the Hilberg exponent (Takahira et al., 2016) (see also Appendix M). In other words, the marginal benefit of each additional token of context diminishes as a power-law. This has a direct consequence for optimization: after T steps, one additional token of training data does not meaningfully improve next-token prediction—the informative signal is spread over a timescale of length $\Theta(T)$. Fixed momentum and weight-decay parameters impose a constant memory horizon with exponential forgetting, creating a fundamental mismatch with this growing timescale. *Logarithmic-time scheduling* resolves this by letting the optimizer's effective memory grow with time.

Log-time momentum schedules are known to accelerate gradient descent on (non-strongly) convex functions in *full-batch settings* (Nesterov, 2004), and have shown empirical benefits for SGD with momentum on small-scale deep auto-encoders (Sutskever et al., 2013). However, naively applying Nesterov's schedule to stochastic gradients can lead to unstable training, as proven in Thm. 7 Even et al. (2021) on smooth convex functions (see also Figs. 3, 38c). To address this instability, Ferbach et al. (2025) introduced DANA, an SGD method that combines log-time momentum with a damping schedule determined by the power-law exponent of the data. On a toy model with power-law data, DANA provably improves the scaling exponent (accelerates) over SGD without momentum (Fig. 38).

Motivated by these principles, **we study empirically the scalability of log-time schedules for momentum and weight decay in the practical setting of language model pretraining**, asking whether their improvements grow with increasing model size. Rather than SGD, we focus on ADAMW, a standard optimizer for modern LLM training, where momentum and weight decay default parameters are typically held constant throughout training rather than scheduled in log-time. Building on Ferbach et al. (2025) and the power-law structure observed in language, we introduce ADANA (Alg. 1), which equips ADAMW with scalable functional forms for momentum and weight decay (WD), together with momentum damping. ADANA uses:

1. *Logarithmic-time schedules* for the first and second moments and independent weight decay:
$$\beta_1(t) = \beta_2(t) = 1 - \frac{\delta}{\delta + t}, \qquad \lambda(t) = \frac{\omega}{t}, \qquad \text{where } \omega, \delta \text{ are constants.}$$
2. A *momentum damping schedule* $\alpha(t) = \tilde{\alpha}(1+t)^{1-\kappa}$, $\kappa \in (0, 1)$ theoretically motivated in Ferbach et al. (2025). [3] The exponent $\kappa > 0$ is straightforward to tune and empirically transfers across model scales (Fig. 5a); see Secs. 2 and L. In simplified settings, the optimal κ is tied to the power-law exponent of the data covariance (Ferbach et al., 2025); we speculate that, for language, it may similarly be related to the Hilberg exponent (see Sec. M). For batch size 32, empirically $\tilde{\alpha} \approx 1$.

[3] The $\alpha(t)$ is smaller ("damped") than standard Nesterov which is $\kappa = 0$ but still grows to accelerate.

Model (D)	Layers	Heads	Non-emb. (P)	Compute (C)
186M	10	14	96.3M	$8.8 \cdot 10^{-1}$
254M	12	16	151.0M	1.7
664M	18	24	509.6M	$1.3 \cdot 10^{1}$
1.41B	24	32	1.21B	$6.2 \cdot 10^{1}$
2.62B	30	40	2.36B	$2.2 \cdot 10^{2}$

Table 1: **Enoki scaling ladder** Charles et al. (2025). Head dim. = 64; Embed. dim. = Heads $\times$ Head dim.; MLP Hidden dim. = Heads $\times$ 256; Iteration = $\frac{\text{tokens}}{32\times2048}$; Non-embd params (P) = $12\times(\text{embd dim})^2\times$ transformer layers; Tot. params (D) = non-embd $+ 2\times$ embd dim $\times$ 50304.

Parameter	Value
Sequence length	2048
Batch size (sequences)	32
Tokens per batch	65,536
Vocabulary size	50,304
Warmup fraction	0.02
Final LR fraction γ^*	0.10
LR rule $\gamma(t)$	cosine decay
Precision/state	bfloat16/float32
Gradient clipping	0.5
Numerical stability ϵ	1e-8
Weight-decay†	independent

Table 2: **Training hyperparameters.** †Except embedding and normalization layers and biases.

Importantly, ADANA requires no major structural changes to architectures or training pipelines tuned for ADAMW and introduces only one additional hyperparameter, making it inexpensive to deploy.

We conducted a **comprehensive empirical evaluation** of ADANA (Alg. 1), ADAMW (Alg. 2), MUON (Alg. 5) and ADEMAMIX (Alg. 6) on scaling ladders of decoder-only transformers (with 45M-2.6B parameters) using FineWeb (Penedo et al. (2024)). Combined, ADANA's innovations yielded **compute-efficiency improvements of up to 40%**[1] **against ADAMW, with gains that persist as model size increases**. Moreover, applying log-time weight-decay schedules to some algorithms boosted their performance by about **10%** against their constant weight-decay counterparts.

Finally, we analyze potential failure modes of ADANA and introduce **robust variants of ADANA to sparse gradients and sensitivity to the κ parameter**. These variants retain comparable performance while we show they improve stability in theoretical and synthetic experimental settings.

Related Work (Extended in Sec. A). On the theoretical side, Varre and Flammarion (2022); Ferbach et al. (2025); Yarotsky (2025) show on simple, quadratic linear regression problems with power-law data that log-time momentum schedules with damping accelerates against SGD.

A related algorithm to ADANA, ADEMAMIX (Pagliardini et al. (2025), Alg. 6) uses multiple momentum buffers and more hyperparameters than ADAMW, which typically must be tuned at each scale and whose roles are not always transparent. In its original form, ADEMAMIX uses constant momentum parameters $\beta_i(t) \equiv \beta_i$ together with a constant damping parameter $\alpha(t) \equiv \tilde{\alpha}$; as a heuristic, the authors introduce a linear damping schedule $\alpha(t) = \tilde{\alpha} \cdot \frac{t}{T}$, where T denotes the training horizon. We show this schedule closely mirrors a damped schedule of the form $\alpha(t) = T^{-\kappa}(\delta + t)$ when scaling its hyperparameters in a particular way (comparison with ADANA; Sec. J). Note that in all our experiments, ADEMAMIX hyperparameters are not swept but set to mirror ADANA schedules. Empirically, ADEMAMIX improves upon ADAMW (Fig. 1), though it consistently under performs against ADANA at scale.

Semenov et al. (2025) benchmarks a wide range of optimizers for LLM training. They report strong performance from MARS (Yuan et al., 2025) and ADEMAMIX relative to ADAMW, particularly as the number of training iterations increases. Wen et al. (2025) find that matrix-preconditioned methods perform best at small scales but exhibit diminishing gains relative to ADAMW at larger scales.

1.1 EXPERIMENTAL SET-UP

We evaluate optimizers on decoder-only transformers trained on FineWeb (Penedo et al., 2024), following Semenov et al. (2025). Experiments are in the compute-optimal regime: for a fixed compute budget C, we choose the number of non-embedding parameters P and training tokens N to minimize loss. We adopt the framework of Hoffmann et al. (2022), modeling total training compute[4] as "$C = 6ND$" and predicting the optimal token count to scale as $N = 20D$, where D is the total parameter count including embeddings. Experiments in the main text are on the *Enoki* model, a GPT-3–like model with RoPE, QK-LayerNorm, pre-LN blocks, residual scaling, no weight tying, and no z-loss. We fix the head dimension at 64 while proportionally increasing width and depth (Table 1 & 2). We also include a set of experiments on the *Qwen3* model in Sec. G, which confirm those on Enoki, and experiments with both longer/shorter training horizons than Chinchilla (Figs. 5a,

[4]For C, we use D = non-embd + embd dim $\times$ 50304; see Sec. E for complete discussion about compute.

33). Details for the experimental setup are in Sec. C.3 & C; Algs. used in Sec. D; Baselining & hyperparameter procedures in Sec. F; Compute measurements/scaling law procedures in Sec. E.

Initialization. Embeddings and standard linear layers use fan-in scaling (std $= 1/\sqrt{\text{fan}_{\text{in}}}$); residual-branch projections (attention and MLP outputs) are further scaled by depth (std $= 1/\sqrt{2\,\text{fan}_{\text{in}}\,L}$, with L layers); all biases are zero.

LR search. To enable fair comparison across scales, we tune hyperparameters up to 700M and fit power-law scaling rules to the top-K ($K = 5$) peak learning rates γ^* for each optimizer, ranked by (weighted) final validation loss at each model size.[5] We fit a saturated power law $\gamma^*(P) = a \cdot (b + P)^{-d}$, where P is the number of non-embedding parameters, $a, b, d > 0$ are fitted parameters. See Sec. F.3 for fitting methodology, explicit formulas, and visualizations.

Batch. All experiments are performed with global batch $B = 32$ and $\tilde{\alpha} \equiv 1$ (Fig. 5a). We study and provide additional discussion on larger batches in Sec.2.1 & H. ADANA's compute efficiency gains extend to larger batch regimes, with appropriate batch-dependent scaling rules (especially on $\tilde{\alpha}$).

2 BUILDING ADANA

Consider the risk function $\mathcal{R} : \mathbb{R}^d \to \mathbb{R}$ and the problem: $\min_{\theta \in \mathbb{R}^d} \mathcal{R}(\theta) \stackrel{\text{def}}{=} \mathbb{E}_x[\mathcal{L}(\theta; x)]$. We denote (mini-batched) stochastic gradients as $g_{t+1} = \frac{1}{B} \sum_{i \in B} \nabla \mathcal{L}(\theta_t, x^i_{t+1})$ where $\{x^i_{t+1}\}$ is data generated at each iteration and not reused. For a detailed discussion of this section, see Sec. L.

We build ADANA (Alg. 1) by revisiting momentum and Nesterov-style acceleration (Nesterov, 1983; Sutskever et al., 2013) through the lens of logarithmic-time scheduling of ADAMW's adaptive 1st/2nd moments. The key modifications are: (1) time-dependent momentum coefficients $\beta_1(t) = \beta_2(t) = 1 - \delta/\delta+t$, (2) time-dependent *independent* weight-decay $\lambda(t) = \omega/t$ (see the definition in (Independent-WD)), and (3) a damped Nesterov-style update with scaling factor $\alpha(t) = (1 + t)^{1-\kappa}$ where $\kappa > 0$ is fixed for all scales. Specifically, ADANA modifies the NADAMW update rule as

$$\theta_{t+1} = \theta_t - \gamma(t) \left(\gamma^* \cdot \frac{\alpha(t) \cdot m_{t+1} + g_{t+1}}{\sqrt{v_{t+1}} + \epsilon} + \lambda(t) \theta_t \right). \tag{1}$$

The parameter update combines the current gradient g_{t+1} with the scaled momentum $\alpha(t) \cdot m_{t+1}$, both normalized by the adaptive second moment. Figure 2 shows the training dynamics of ADANA and ADAMW on different scales showcasing compute-optimal scaling laws.

2.1 SCHEDULING β_1: BENEFIT OF LOG-TIME MOMENTUM

We begin by considering the widely used ADAMW algorithm which maintains a first moment estimate m_t and second moment estimate v_t, updating as:

$$(1^{st}\ \textit{mom.})\ m_{t+1} = \beta_1 m_t + (1 - \beta_1) g_{t+1}, \quad (2^{nd}\ \textit{mom.})\ v_{t+1} = \beta_2 v_t + (1 - \beta_2) g^2_{t+1},$$
$$(\textit{param.})\ \theta_{t+1} = \theta_t - \gamma(t) \left(\frac{\gamma^* \cdot m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} + \lambda \theta_t \right), \tag{ADAMW}$$

where $\beta_1, \beta_2 \in (0, 1)$ are 1st/2nd moment hyperparameters, $\gamma(t) > 0$ is LR schedule, $\gamma^* > 0$ is peak LR, $\lambda > 0$ is the *independent* WD coefficient, and $\epsilon > 0$ numerical stability. A variant of ADAMW, NADAMW, (c.f. Dozat (2016)) replaces the update rule of ADAMW by adding an extra gradient g_{t+1} to the param. update. A common practical choice is to set β_1 and β_2 to fixed constants independent of model size and training time. However we recall a potential fundamental limitation in this setting: *constant β_1 may not improve scaling behavior beyond stochastic gradient descent (SGD).*

Constant β_1 may not add any benefit over SGD. Consider the standard momentum update,

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) g_{t+1}, \quad \theta_{t+1} = \theta_t - \gamma g_{t+1} - m_{t+1}.$$

Unrolling the recursion gives $m_{t+1} \asymp \sum_{j=0}^{t} \beta_1^j g_{t+1-j}$, revealing that constant momentum implements an exponentially weighted moving average of past gradients. In the small batch stochastic setting, gradients do not change appreciably over the resulting finite memory horizon, implying $m_{t+1} \approx$

[5] Search strategy: first explore log-scale LRs with factor-2 spacing, then refine to get top-5 (Sec. F.1) for heads 6–24; For heads > 24, used the fitted rule with confidence intervals; tuning at the largest sizes is impracticable.

g_{t+1}. As a consequence, the update reduces to SGD with a rescaled learning rate. This observation has been proven in specific settings (Paquette and Paquette, 2021; Ferbach et al., 2025; Zhang et al., 2019).

Log-Time Momentum Benefit. To overcome the limitations of fixed momentum, the effective memory of m_{t+1} must grow with training time. A natural and theoretically motivated approach is *logarithmic-time momentum*, achieved by scheduling $\beta_1(t) = 1 - \delta/\delta+t$. Under this schedule, gradients from a fixed *fraction* of the training history contribute to the momentum term. Notably, this is the same momentum schedule that enables acceleration in deterministic convex optimization (Nesterov, 1983). The term "logarithmic time" reflects that, under the change of variables $\tau = \log t$, this schedule corresponds to a *constant* effective momentum in τ (see Sec L.3).

$$\theta_{t+1} = \theta_t - \gamma(t)\left(\gamma^* \cdot \frac{\tilde{\gamma} \times g_{t+1} + \alpha(t) \cdot m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} + \lambda(t) \cdot \theta_t\right)$$

Optimizer	$\tilde{\gamma}$	$\alpha(t)$	β_1	β_2
AdamW (DW)	0.0	1.0	0.9	0.999
Log-NAdamW (DW)	1.0	$\delta + t$	$\delta/(\delta+t)$	$\delta/(\delta+t)$
Log-AdamW (DW)	0.0	1.0	$\delta/(\delta+t)$	$\delta/(\delta+t)$
ADANA No Gradient	0.0	$(1+t)^{1-\kappa}$	$\delta/(\delta+t)$	$\delta/(\delta+t)$

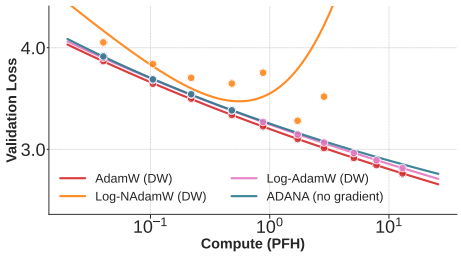


Figure 3: **Instabilities in Logarithmic-time Momentum** LOG-NADAMW (DW) is unstable due to the undamped momentum $\alpha(t) = \delta + t$. LOG-ADAMW (DW) and ADANA-no-gradient respectively use $\alpha(t) = 1$ and $\alpha(t) = (1+t)^{1-\kappa}$, $\kappa = 0.75$ without the stabilizing gradient term and show degraded performance against ADAMW (DW). Both stabilizing gradient and damped momentum $\alpha(t)$ are necessary to achieve good performances.

Performance Gains vs. Instability. While log-time momentum offers performance gains, it comes at the cost of stability. Consider the stochastic analogue of Nesterov's accelerated method, which uses logarithmic-time momentum, $\beta_1(t) = 1 - \delta/\delta+t$:

$$\textit{(mom.)}\ m_{t+1} = \beta_1(t)m_t + (1-\beta_1(t))g_{t+1},\quad \textit{(param.)}\ \theta_{t+1} = \theta_t - \gamma \cdot \left(g_{t+1} + \tfrac{t+\delta}{\delta} \cdot m_{t+1}\right).$$

The additional gradient term g_{t+1} in the update is essential. In Fig. 3, we compare ADAMW to a variant with log-momentum but without the additional gradient term (LOG-ADAMW, Alg. 3). Removing that additional gradient significantly degrades performance. Moreover, with stochastic gradients, instability arises from noise accumulation due to log-momentum. To give some intuition, writing $g_{t+1} = \nabla\mathcal{R}(\theta_t) + \varepsilon$ and unrolling the momentum recursion shows that the accumulating ε's grow in time, while only a single ε in the extra gradient term. Since there is only one LR to control both terms, the update becomes dominated by noise/accumulating ε term and "diverges". This behavior has been rigorously established on a toy scaling model (PLRF^3), where stochastic Nesterov diverges Even et al. (2021), and is corroborated by our transformer experiments in Fig. 3, where ADAMW with log-momentum and an additional gradient term (LOG-NADAMW) exhibits instability.

Damped Nesterov acceleration. The failure of stochastic Nesterov underscores a key design principle: log-momentum can be beneficial, but only when its contribution is sufficiently damped relative to the gradient to avoid instability. This motivates a *generalized Nesterov/*DANA *update rule* (Ferbach et al., 2025):

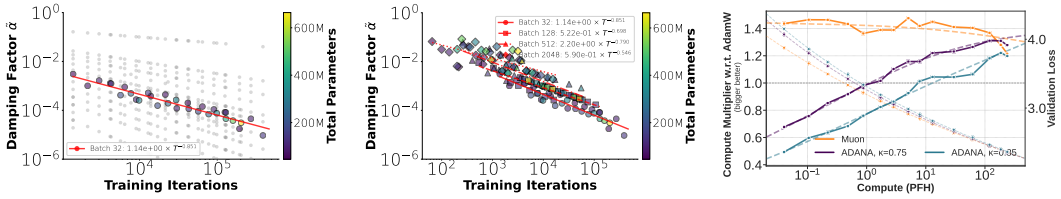
$$\textit{(mom.)}\ m_{t+1} = \left(1 - \tfrac{\delta}{\delta+t}\right)m_t + \tfrac{\delta}{\delta+t}g_{t+1},\quad \textit{(param.)}\ \theta_{t+1} = \theta_t - \gamma\left(g_{t+1} + \alpha(t)m_{t+1}\right),$$

where $\alpha(t)$ is an additional, time-dependent damping factor. Standard Nesterov corresponds to $\alpha(t) \approx t$ which is unstable for stochastic gradients. Larger $\alpha(t) \gtrsim t$ further amplifies the momentum contribution and worsens divergence, while $\alpha(t) = 0$ removes momentum entirely, reducing the method to SGD. The only viable regime is $\alpha(t) \lesssim t$, where the momentum contribution is deliberately damped compared with standard Nesterov. That way it remains comparable in magnitude to the stochastic gradient for stability but still grows during training to accelerate dynamics. This motivates the question: *How should one choose a damping schedule that suppresses noise accumulation while preserving the benefits of log-momentum?*

Damping Schedule with Performance Gains & Stability. The key design in ADANA is the damping schedule

$$\alpha(t) = \tilde{\alpha} \cdot (1+t)^{1-\kappa}, \quad \text{where } \kappa > 0. \tag{2}$$

This form is motivated by a theoretical analysis of DANA on the Power-Law Random Features model, a toy model which exhibits scaling laws similar to those observed empirically (Maloney et al., 2024; Bahri et al., 2024); the data x in PLRF has covariance whose j-th eigenvalue is $j^{-2\rho}$; see Sec. B. On PLRF, Ferbach et al. (2025) prove that a damping schedule of this form has the following properties, regardless of model size:



(a) $\alpha(t)$ **on time and scale.** (b) ↑ **batch** ⇒ ↑ **damping sched.** (c) **Large batch** $B = 256$

Figure 5: **(a) Dependence of $\alpha(t)$ on time and scale.** Sweeps of constant $\tilde{\alpha}$ in ADANA with $\alpha(t) = \tilde{\alpha} \cdot (1+t)$. Vary both the iterations, going beyond Chinchilla scale, and model sizes. Fitting optimal $\tilde{\alpha}$ recovers $\alpha(t) = (1+t)^{1-\kappa}$. Note here $\tilde{\alpha} \approx 1$ for $B = 32$ and $\kappa \approx 0.85$. **(b) Larger batch allows for larger damping sched.** Same setting as in Figure 5a: we use $\alpha(t) = \tilde{\alpha} \times (1+t)$ and sweep $\tilde{\alpha}$ across model size and iterations. Increasing batch size allows for larger constant $\tilde{\alpha}$. **(c) Large batch $B = 256$ MUON, ADAMW and ADANA with $\alpha(t) = (1+t)^{1-\kappa}$ and $\kappa \in \{0.75, 0.85\}$.** Under $\alpha(t) = (1+t)^{1-\kappa}$ with large batch $B = 256$, ADANA with $\kappa = 0.75$ outperforms $\kappa = 0.85$ hence allowing larger damping schedule. The performance increases with scale for ADANA with both $\kappa \in \{0.75, 0.85\}$ eventually outperforming MUON.

$\kappa \in [0, 1/2\rho)$: Diverges (include $\kappa = 0$, Nesterov).

$\kappa \in [1/2\rho, 1)$: Stable and **provably accelerates** over SGD (See Fig 38a). Optimal at $\kappa = 1/2\rho$, for which reason we call κ the *spectral dimension*.

$\kappa \in [1, \infty)$: Reduces to SGD performance.

The κ dependence picture generalizes to ADANA, small κ underperform ADAMW up to a critical point, after which they outperform ADAMW up to $\kappa = 1$, at which they have similar performance to ADAMW (DW). See Figure 8 for a κ-cross-section and Figure 4 for scale-dependence.

Spectral dimension, κ, on LLMs. In Fig. 4, across a transformer scaling ladder using Enoki scaling, all ADANA runs with $\kappa \in [0.75, 0.9]$ substantially outperform the ADAMW baseline. Notably, these gains do not diminish as model size increases, with optimal performance using $\kappa \approx 0.85$.

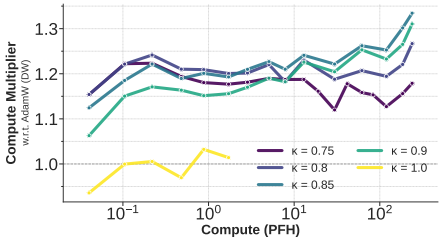


Figure 4: **Impact & sensitivity of κ on scaling performance of ADANA Alg. 1** on Enoki transformer scaling ladder with FineWeb; $\kappa = 0.85$ allows for best performance. At the largest scales compared with ADAMW-log-time WD, the optimal $\kappa = 0.85$ yields more than 30% (40% w.r.t ADAMW with constant WD) compute gain improvement against less than 20% for $\kappa = 0.75$; Performance improves across scales for more conservative $\kappa \geq 0.85$ while degrading for the more aggressive $\kappa \leq 0.8$; $\kappa = 1.0$ similar to the baseline as predicted by the PLRF toy model.[6]

In Sec. I we present measurements of the covariance spectra decay of the activations, in the spirit of comparison to the PLRF. We note that another interpretation of κ is related to fundamental information theoretic properties of the data, which we discuss in Sec. M; this point of view is not easily reconcilable with the PLRF interpretation, and a properly explaining κ-dependence of ADANA in the LLM setting is a future direction.

Alternative $\alpha(t)$ Schedules. Other damping schedules are possible. In particular, Ferbach et al. (2025) evidences on PLRF that a linear schedule $\alpha(t) = \tilde{\alpha} \times (\delta + t)$, with sufficiently small $\tilde{\alpha}$, can also achieve stability, albeit with slightly worse performance than $(1+t)^{1-\kappa}$. The constant that yields the best performance is $\tilde{\alpha} = T^{-\kappa}$, where T is the runtime horizon and κ is an explicit quantity dependent on the power-law exponent of the covariance spectrum. In Fig. 10, we compare ADANA with $(1+t)^{1-\kappa}$ to ADANA with $\alpha(t) = T^{-\kappa}(\delta + t)$ (see also ADEMAMIX Sec. J). While the latter consistently outperforms ADAMW with 15% compute gains, it performs worse than ADANA with $(1+t)^{1-\kappa}$.

Correct functional dependence on time and scale in $\alpha(t)$. We investigate whether alternative damping schedules can improve performance and, more importantly, whether our proposed functional form captures the correct dependence on both training iterations and model size (Fig. 5a). We run ADANA with a linear schedule $\alpha(t) = \tilde{\alpha} \cdot (1+t)$ and sweep the constant $\tilde{\alpha}$ under two settings: (i) *training far beyond the Chinchilla horizon* while fixing head size, to probe iteration T dependence, and (ii) *varying head size*, to probe scaling with model size (see Sec. J.3 & Sec. H, Fig. 23). Fitting the optimal $\tilde{\alpha}$ across both regimes reveals $\tilde{\alpha} = T^{-\kappa}$ with $\kappa \approx 0.83$ (Fig. 5a). Notably,

[6] Ferbach et al. (2025) shows a correspondence between $\kappa = 1$ and Schedule-free. Figure 38b shows it has the same scaling behavior than SGD as predicted by theory (Ferbach et al., 2025)

the optimal choice of κ appears largely independent of model size. This empirically recovers the schedule $\alpha(t) = (1+t)^{1-\kappa}$, predicted to be optimal in the PLRF model, and confirms the functional dependence of $(1+t)^{1-\kappa}$ on both iterations and scale.

Effect of Batch Size. Increasing the batch size $B > 0$ reduces the stochastic noise in the gradient estimates. Therefore, larger batch sizes should reduce the need for Nesterov damping $\alpha(t) = \tilde{\alpha}(1+t)^{1-\kappa}$. In the limit of very large batch $B \to \infty$ we expect no damping to be required for full-batch Nesterov $\tilde{\alpha} = 1,\ \kappa = 0$. Hence we expect that increasing B favors larger $\alpha(t)$ schedules in the form of either larger optimal $\tilde{\alpha}$, or smaller κ.

Sweeping through $\tilde{\alpha}$ in the setting $\alpha(t) = \tilde{\alpha} \times (1+t)$ in Figure 5b shows that increasing batch favors larger $\tilde{\alpha}$. We fitted $\tilde{\alpha} = \hat{\alpha} \times T^{-\kappa}$ where $\hat{\alpha}, \kappa$ are fitted variables across model size and iterations. While no clear behavior for $\hat{\alpha},\ \kappa$ seem to emerge from Figure 5b, we tested two strategies for scaling the damping factor $\alpha(t)$ at larger batch regimes. The first strategy consists in using smaller exponent κ while keeping $\tilde{\alpha} = 1$ and was used at batch 256 in Figure 5c. The second strategy consist in keeping $\kappa = 0.85$ but using a larger constant $\tilde{\alpha}$ and was used in Figure 26 with batch 512. In both cases, we observe that ADANA's performance consistently increases with scale with strong performance against ADAMW at the largest scales. Note that at small scale, ADANA's performance is strongly degraded which can be explained by an excessively large batch for the given scale and a small number of iterations ($T = 1743$ at the smallest scale, Head $= 6$, for batch size $B = 256$).

2.2 SCHEDULING β_2 WITH LOG-TIME

We now turn to how to schedule β_2. Using a long-momentum schedule $\beta_1(t) = 1 - \delta/(\delta+t)$ increases the contribution of past gradients in the update. If using a constant β_2, the contribution of these gradients in the second-moment update v_t is killed exponentially fast. In the case of time sparsity, where a single gradient coordinate is updated only rarely, this can lead to diverging updates due to division by exponentially small numbers. E.g., assume that $\epsilon = 0$ for simplicity and that the first gradient has non-zero first coordinate $(g_0)_1 \neq 0$ is and that for any $t \geq 0$, $(g_t)_0$, we see that the update ratio becomes

$$\left(\frac{m_t}{\sqrt{v_t}}\right)_1 = \frac{(1-\beta_1(0))\prod_{s=1}^{t-1}\beta_1(s)}{\sqrt{(1-\beta_2(0))\prod_{s=1}^{t-1}\beta_2(s)}}.$$

Optimizer	$\theta_{t+1} = \theta_t - \gamma(t)\left(\gamma^* \cdot \frac{g_{t+1}+\alpha(t)\cdot m_{t+1}}{\sqrt{v_{t+1}}+\epsilon} + \lambda(t)\cdot\theta_t\right)$ $\alpha(t)$	β_1	β_2
ADANA 1	$(1+t)^{1-\kappa}$	Log	Log
ADANA 2	$(1+T)^{1-\kappa}t/T$	Log	Log
ADANA 3	$(1+t)^{1-\kappa}$	Log	0.999
ADEMAMIX (DW)	$(1+T)^{1-\kappa}t/T$	Approx Log	0.999

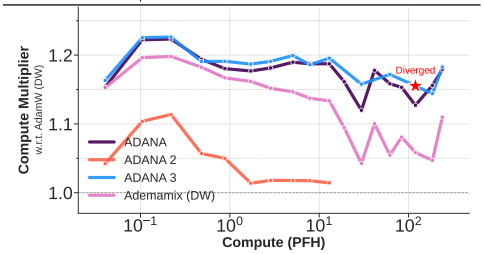


Figure 6: **Ablation Study on ADANA.** Comparison of compute gains relative to ADAMW for different variants of ADANA. ADEMAMIX and ADANA 1 significantly underperform ADANA, especially at larger scales due to the $\alpha(t)$ schedule. β_2 constant do not impact performance much but diverges at 36 heads. Details Sec. D.9.

This leads to the stability condition $\beta_1^2/\beta_2 < 1$ in order for the ratio to not explode (see Theorem L.1 for more details); In particular, any schedule $\beta_2 \geq \beta_1$ satisfies this stability condition. The diagonal choice $\beta_2 = \beta_1$ has additionally been observed to be practically optimal and avoids various potential pitfalls of ADAMW when $\beta_2 > \beta_1$ (Orvieto and Gower, 2025). In Figure 10, we compare the performance of ADANA with constant $\beta_2 = 0.999$ and logarithmic time $\beta_2 = 1 - \delta/(\delta+t)$. While both perform similarly at small scale, ADANA with constant $\beta_2 = 0.999$ diverges at 36 heads showing the instability arising from constant β_2 used with long momentum.

2.3 LOGARITHMIC-TIME WEIGHT-DECAY

We analyze weight-decay schedules and show that log-time scheduling $\lambda(t) = \omega/t$ which is naturally suited to problems in which information must accumulate on growing time scales, provides substantial improvements for free and across optimizers. We use decoupled weight-decay with the (Loshchilov and Hutter, 2019) parameterization, sometimes referred to as *independent weight-decay*

$$\theta_{t+1} = \theta_t - \gamma(t)(\gamma^* g_{t+1} - \lambda(t)\theta_t). \qquad \text{(Independent-WD)}$$

We compare two schedules: (i) constant weight-decay, most standard in prior work, under the scaling $\lambda = \omega/T$, and (ii) log-time approximation $\lambda(t) = \omega/(T/10+t)$, which is a version of $\lambda(t) = \omega/t$ with a warmup period of $T/10$ (denoted by *decaying weight-decay, DW*). The choice of 10 made ADAMW

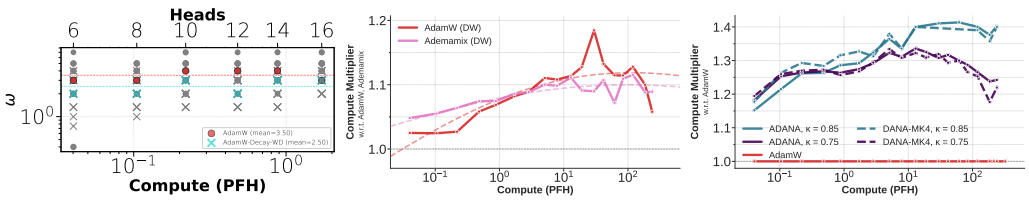


Figure 7: **Left: Sweeps of ω for constant weight-decay $\lambda = \omega/T$ and log-time weight-decay $\lambda(t) = \omega/(T/10+t)$.** The optimal weight decay seems in both cases constant across scales. **Center: Compute gains for ADAMW and ADEMAMIX with logaritmic-time versus constant weight-decay.** All experiments used $\omega = 4.0$. **Right: Comparison of ADANA and DANA-MK4 with $\kappa \in \{0.75, 0.85\}$.** Both algorithms are optimal with $\kappa = 0.85$ and share the same performance across scales.

(DW) perform well; see Figure 37 for other choices besides 10 and its effect on ADANA and ADAMW. Joint sweeps over (γ, ω) shown in (Fig. 7, left) evidence that in both cases the optimal ω remains constant across model scales, validating the proposed scaling (c.f. Sec K, Figs. 35 & 36). Empirically, log-time weight-decay consistently outperforms constant decay for both ADAMW and ADEMAMIX (Fig. 7, center). Importantly the compute efficiency gains are about 10% and appear to increase with scale.

Previous works on weight-decay scaling. Finally, several recent works have established scaling rules for weight-decay that align with our analysis, in particular the schedule $\lambda = \omega/T$ (see Sec. A). In our framework, Wang and Aitchison (2025); Bergsma et al. (2025) show that the ADAMW timescale $\tau := B/\lambda D$, where D is the dataset size and B the batch size, should remain constant across scales. Noting that $T := D/B$ is the number of iterations, this implies $\lambda \propto 1/T$. However, to our knowledge, no prior work apply weight-decay in log time as $\lambda(t) = \omega/t$. In Golatkar et al. (2019), the authors show that weight-decay in SGD improves generalization when applied early in training but can be neutral or harmful if applied later. This motivates our time-decaying weight-decay schedule, which enforces stronger regularization early and gradually relaxes it as training progresses.

Summary of Main Experimental Results. We present comprehensive experiments comparing ADANA and its variants against ADAMW, MUON, and ADEMAMIX across transformer scales from 45M to 4.4B parameters. Fig. 1 shows the main scaling comparison. We summarize:

1. **ADANA outperforms ADAMW** across all model sizes, with compute-efficiency gap increasing at larger scales; κ strongly influences performance with optimal at $\kappa = 0.85$. **DANA-STAR-MK4** significantly outperforms ADAMW but worse than **ADANA**, with a gap that decreases with scale.
2. Both **MUON** and **ADEMAMIX** with appropriate scheduling (see Sec. J) show competitive performance at smaller scales but the advantage diminishes at larger scales.
3. **ADAMW with weight-decay in log-time** (ADAMW (DW)) achieves substantial improvement compared to ADAMW, by itself around 10% compute gains.

3 HARDENING ADANA: FROM ADANA TO DANA-STAR-MK4

While ADANA performs well across a wide range of settings, we identify potential failure modes that motivate hardened variants. We summarize the key issues and solutions here; full details (including definitions and diagnostic experiments) are provided in Appendix N and Appendix L.2.

Inhomogeneous Spectral Dimensions. The spectral dimension κ depends on the effective data exponent on PLRF[3]. Within a single transformer, different layers may have different effective spectral dimensions. See, e.g., Fig. 28. If κ is too small for some layers, the log-time momentum term can dominate and cause instability. This motivates DANA-MK4. The key modification is clipping the damping schedule $\alpha(t)$ by a sign-to-noise ratio (SNR) estimator. Specifically, define normalized momentum factor $\texttt{mfac}$ and clipped scaling factor $\alpha_{\texttt{fac}}$

$$\texttt{norm} = \frac{1}{\sqrt{v_{t+1}+\epsilon}}, \quad \texttt{mfac} = |m_{t+1}| \cdot \texttt{norm},$$
$$\alpha_{\texttt{fac}} = \min\left\{t^{1-\kappa} \cdot \texttt{mfac}, \texttt{clipsnr}\right\}.$$

The quantity $\texttt{mfac}$ measures the magnitude of the momentum relative to the second moment, serving as a proxy for the SNR.

Variant	Key Feature
ADANA	Base logarithmic-time scheduling
DANA-MK4	+ SNR clipping (κ robustness)
DANA-STAR	+ τ estimator (sparse gradients)
DANA-STAR-MK4	+ both

Table 3: **ADANA variant comparison.**

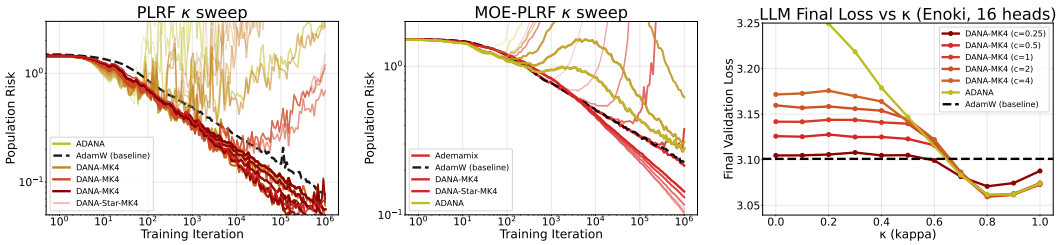


Figure 8: **Left: PLRF κ sweep ($m = 1$, batch size 1).** Population risk vs. training iteration for different κ and `clipsnr` values. ADANA (clipsnr$= 16$, olive) performs well at intermediate κ but diverges at low κ. DANA-MK4 variants with lower clipsnr (red shades) are more robust (darker $\implies$ lower `clipsnr`). **Center: MOE-PLRF κ sweep ($m = 1000$, batch size 10).** Sparse gradients due to the mixture-of-experts structure cause ADANA, ADEMAMIX, and DANA-MK4 to diverge while DANA-STAR-MK4 remains stable. **Right: LLM κ sensitivity.** Final validation loss vs. κ for varying `clipsnr`. ADANA shows strong dependence on κ. DANA-MK4 variants with lower clipsnr values are more robust, with clipsnr$= 0.25$ showing nearly flat performance for small κ. ADAMW baseline (dashed) is outperformed by all variants at $\kappa \approx 0.80$.

The `clipsnr` is a hyperparameter controlling the maximum allowed scaling (see Fig. 8 for sensitivity to `clipsnr`; if not specified, `clipsnr` $= 2$). This clipping prevents the momentum term from dominating, providing automatic adaptation to the local effective dimension. The term $\alpha_{\texttt{fac}} \cdot \text{sign}(m_{t+1})$ replaces the momentum term in Line 9 (Alg. 10). This makes ADANA robust to a range of effective spectral dimensions. See Fig. 7 (right) comparing ADANA and DANA-MK4.

Sparse Gradient Vulnerability. The long second moment buffer (necessary to balance the first moment average) is vulnerable to gradients that are sparsely updated. This occurs in embedding layers, where the Zipf-law distribution of token frequencies means rare tokens contribute gradients infrequently (see Figure 39). It potentially arises in Mixture of Experts models and transformer blocks, through mechanisms like attention sinks (Xiao et al., 2024). We prove in Sec. P that naive logarithmic-time momentum diverges *under sparse gradients*:

> **Thm. L.1.** ADAM with $\epsilon = 0$ and $\beta_1(t) = 1 - \delta/\delta+t$ will diverge for *all* choices of β_2 in the presence of sparse updates. The same holds for ADANA.

The solution is DANA-STAR, which introduces a probability estimator τ that tracks how frequently each parameter receives meaningful updates (See Alg. 9 & Sec. N for full details). The effective time $t_{\texttt{eff}} = \max\{t \cdot \tau, 1\}$ rescales the iteration count by the update probability, ensuring that rarely-updated parameters do not underflow the 2nd moment buffer.

Validation on Synthetic Experiments. We evaluate the proposed variants on PLRF[5] and its Mixture-of-Experts extension (MOE-PLRF; see Sec. B), which isolate the effects of κ misspecification and sparse gradients, respectively. On PLRF (Fig. 8, left), ADANA is highly sensitive to κ: too small values yield insufficient acceleration, while overly aggressive choices lead to divergence. In contrast, the stabilized DANA-MK4 variants remain robust across a wide range of κ, with SNR clipping automatically limiting the noise. On MOE-PLRF (Fig. 8, center), which introduces sparse gradients, these differences are amplified: ADANA exhibits a narrow optimal κ regime, whereas DANA-MK4 and DANA-STAR-MK4 maintain stable performance throughout.

Validation on LLMs. We observe the same trends in language model training (Fig. 8, right). While ADANA achieves strong performance when κ is well tuned, it is sensitive to misspecification. `clipsnr` in DANA-MK4 slightly reduces peak gains but yields markedly flatter performance across κ, improving robustness—a practical advantage when optimal hyperparameters vary across layers.

Conclusions. We introduced ADANA, which leverages log-time schedules and damping, motivated by the power-law structure of language data, to achieve improvements (40%) across a scaling ladder. Our functional forms for $\beta_1, \beta_2, \lambda$, and α are easily tunable at small scales, require no architectural modifications, and transfer effectively to larger models, with indications that they generalize across architectures (Qwen3, Sec. G). Log-time weight-decay schedules, alone, boosts performance for many optimizers, including ADAMW. Together, these results demonstrate the potential of scale-aware scheduling that leverages language's power-law structure to improve LLM optimization. Future work includes understanding log-time weight decay benefits, and investigating MUON with ADANA, as logarithmic-time scheduling may provide similar benefits to MUON.

AI USE STATEMENT

In this work, we used generative AI tools for coding and in particular implementing methods.

We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, assisting with translation, cleaning and reformatting dataset, supporting qualitative and thematic data analysis, interpreting results.

Generating synthetic data sets is not applicable to this work.

Additionally, we used generative AI tools for creating or modify scientific figures or images, identifying relevant literature, suggesting a structure for the research paper, proposing the title of the research paper. We have reviewed all AI-assisted work. For example, we restructured extensively the proposed structure of the initial draft of the paper as well as title. We verified references of the literature review proposed and made sure of their relevance.

We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We do not think this work raises questions regarding the Code of Ethics, apart from the general questions related to scaling which we detail in our Broader Impact Statement.

REPRODUCIBILITY STATEMENT

We made significant efforts in hyperparameter tuning for all algorithms to ensure that our claims are fair to each algorithm and are independent from hyperparameter tuning. We extensively detail our experimental methodology, baselining procedure and tuning for reproducibility in Appendix F. We additionally detail the experimental set-up in Appendices C and G and the algorithms used in Appendix D.9. We commit on releasing the code to reproduce our experiments upon acceptance. We chose to not make it available at submission time due to anonymity concerns. Finally, all our theoretical claims are justified. For-example Theorem L.1 and all related proofs are shown Appendix P.

REFERENCES

Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *The Journal of Machine Learning Research*, 18(1):8194–8244, 2017.

Mahmoud Assran and Michael Rabbat. On the Convergence of Nesterov's Accelerated Gradient Method in Stochastic Settings. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

Necdet Serhat Aybat, Alireza Fallah, Mert Gürbüzbalaban, and Asuman Ozdaglar. Robust Accelerated Gradient Methods for Smooth Strongly Convex Functions. *SIAM Journal on Optimization*, 30(1): 717–751, 2020. doi: 10.1137/19M1244925.

Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem. `http://github.com/google-deepmind`, 2020.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proc. Natl. Acad. Sci. USA*, 121(27):Paper No. e2311878121, 8, 2024.

Amir Beck and Marc Teboulle. Gradient-based algorithms with applications to signal recovery. *Convex Optimization in Signal Processing and Communications*, 2009.

Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power Lines: Scaling Laws for Weight Decay and Batch Size in LLM Pre-training. *arXiv preprint arXiv:2505.13738*, 2025.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 560–569. PMLR, 2018.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. DeepSeek LLM: Scaling Open-Source Language Models with Longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

Raghu Bollapragada, Tyler Chen, and Rachel Ward. On the fast convergence of minibatch heavy ball momentum. *IMA Journal of Numerical Analysis*, page drae033, 08 2024. ISSN 0272-4979. doi: 10.1093/imanum/drae033.

Blake Bordelon, Alexander Atanasov, and Cengiz Pehlevan. A Dynamical Model of Neural Scaling Laws. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 4345–4382. PMLR, 2024.

Blake Bordelon, Alexander Atanasov, and Cengiz Pehlevan. How Feature Learning Can Improve Neural Scaling Laws . *International Conference on Learning Representations (ICLR)*, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.

Ethan Caballero, Kshitij Gupta, Irina Rish, and David Krueger. Broken Neural Scaling Laws. In *International Conference on Learning Representations (ICLR)*, 2023.

Bugra Can, Mert Gurbuzbalaban, and Lingjiong Zhu. Accelerated Linear Convergence of Stochastic Momentum Methods in Wasserstein Distances. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97, pages 891–901. Proceedings of Machine Learning Research (PMLR), 2019.

Andrea Caponnetto and Ernesto De Vito. Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*, 7:331–368, 2007.

Luigi Carratino, Alessandro Rudi, and Lorenzo Rosasco. Learning with sgd and random features. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

Zachary Charles, Gabriel Teston, Lucio Dery, Keith Rush, Nova Fallen, Zachary Garrett, Arthur Szlam, and Arthur Douillard. Communication-Efficient Language Model Training Scales Reliably and Robustly: Scaling Laws for DiLoCo. *arXiv preprint arXiv:2503.09799*, 2025.

Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative Pretraining from Pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR, 2020.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:49205–49233, 2023.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984. doi: 10.1109/TCOM.1984.1096090.

James P. Crutchfield and David P. Feldman. Regularities unseen, randomness observed: The entropy convergence hierarchy. *Chaos*, 15(1):25–54, 2003. doi: 10.1063/1.1530990.

Hugo Cui, Bruno Loureiro, Florent Krzakala, and Lenka Zdeborová. Generalization Error Rates in Kernel Regression: The Crossover from the Noiseless to Noisy Regime. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

Anh Quang Dang, Reza Babanezhad Harikandeh, and Sharan Vaswani. (Accelerated) Noise-adaptive Stochastic Heavy-Ball Momentum. *Transactions on Machine Learning Research (TMLR)*, 2025. ISSN 2835-8856.

Łukasz Dębowski. The relaxed Hilberg conjecture: A review and new experimental support. *Journal of Quantitative Linguistics*, 22(4):311–337, 2015. doi: 10.1080/09296174.2015.1072578.

Aaron Defazio. Why Gradients Rapidly Increase Near the End of Training. *arXiv preprint arXiv:2506.02285*, 2025.

Aaron Defazio, Xingyu Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The Road Less Scheduled. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 9974–10007. Curran Associates, Inc., 2024.

Aaron Defazio, Konstantin Mishchenko, Parameswaran Raman, Hao-Jun Michael Shi, and Lin Xiao. Smoothing DiLoCo with Primal Averaging for Faster Training of LLMs. *arXiv preprint arXiv:2512.17131*, 2025.

Leonardo Defilippis, Bruno Loureiro, and Theodor Misiakiewicz. Dimension-free deterministic equivalents and scaling laws for random feature regression. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 104630–104693. Curran Associates, Inc., 2024.

Leonardo Defilippis, Yizhou Xu, Julius Girardin, Emanuele Troiani, Vittorio Erba, Lenka Zdeborová, Bruno Loureiro, and Florent Krzakala. Scaling laws and spectra of shallow neural networks in the feature learning regime. *arXiv preprint arXiv:2509.24882*, 2025.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling Vision Transformers to 22 Billion Parameters. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 7480–7512. PMLR, 2023.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Nolan Dey, Bin Claire Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: CompleteP enables compute-efficient deep transformers. *arXiv preprint arXiv:2505.01618*, 2025.

Aymeric Dieuleveut and Francis Bach. Nonparametric stochastic approximation with large step-sizes. *The Annals of Statistics*, 44(4):1363 – 1399, 2016. doi: 10.1214/15-AOS1391. URL https://doi.org/10.1214/15-AOS1391.

Timothy Dozat. Incorporating Nesterov Momentum into Adam. In *ICLR Workshop*, 2016.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

Werner Ebeling and Gregoire Nicolis. Entropy of symbolic sequences: The role of correlations. *Europhysics Letters*, 14(3):191–196, 1991. doi: 10.1209/0295-5075/14/3/001.

Mathieu Even, Raphaël Berthier, Francis Bach, Nicolas Flammarion, Hadrien Hendrikx, Pierre Gaillard, Laurent Massoulié, and Adrien Taylor. Continuized Accelerations of Deterministic and Stochastic Gradient Descents, and of Gossip Algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 28054–28066. Curran Associates, Inc., 2021.

Katie E Everett, Lechao Xiao, Mitchell Wortsman, Alexander A Alemi, Roman Novak, Peter J Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, et al. Scaling Exponents Across Parameterizations and Optimizers. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 12666–12700. PMLR, 2024.

Damien Ferbach, Katie Everett, Gauthier Gidel, Elliot Paquette, and Courtney Paquette. Dimension-adapted Momentum Outscales SGD. *arXiv preprint arXiv:2505.16098*, 2025.

Nicolas Flammarion and Francis Bach. From Averaging to Acceleration, There is Only a Step-size. In *Proceedings of The 28th Conference on Learning Theory (COLT)*, volume 40 of *Proceedings of Machine Learning Research*, pages 658–695. PMLR, 03–06 Jul 2015.

Sèbastien Gadat, Fabien Panloup, and Sofiane Saadane. Stochastic Heavy Ball. *arXiv preprints arXiv:1609.04228*, 2016.

Saeed Ghadimi and Guanghui Lan. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization I: A generic algorithmic framework. *SIAM J. Optim.*, 22(4): 1469–1492, 2012. ISSN 1052-6234. doi: 10.1137/110848864.

Saeed Ghadimi and Guanghui Lan. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization, II: Shrinking procedures and optimal algorithms. *SIAM J. Optim.*, 23(4):2061–2089, 2013. doi: 10.1137/110848876.

Aditya Sharad Golatkar, Alessandro Achille, and Stefano Soatto. Time Matters in Regularizing Deep Networks: Weight Decay and Data Augmentation Affect Early Learning Dynamics, Matter Little Near Convergence. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. In *arXiv preprint arXiv:1706.02677*, 2017.

Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, William Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi,

Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah Smith, and Hannaneh Hajishirzi. OLMo: Accelerating the Science of Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15789–15809. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.841.

Kanan Gupta, Jonathan W. Siegel, and Stephan Wojtowytsch. Nesterov acceleration despite very noisy gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 20694–20744, 2024.

D Hendrycks and K Gimpel. Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4246–4253, 2020.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Wolfgang Hilberg. Der bekannte Grenzwert der redundanzfreien Information in Texten—eine Fehlinterpretation der Shannonschen Experimente? *Frequenz*, 44(9–10):243–248, 1990. doi: 10.1515/FREQ.1990.44.9-10.243.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 30016–30030. Curran Associates, Inc., 2022.

Prateek Jain, Sham Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Accelerating Stochastic Gradient Descent for Least Squares Regression. In *Proceedings of the 31st Conference On Learning Theory (COLT)*, volume 75, pages 545–604, 2018.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL `https://kellerjordan.github.io/posts/muon/`.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alex Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham Kakade. On the Insufficiency of Existing Momentum Schemes for Stochastic Optimization. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9, 2018. doi: 10.1109/ITA.2018.8503173.

Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*, 2015.

Walid Krichene and Peter L Bartlett. Acceleration and Averaging in Stochastic Descent Dynamics. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30. Curran Associates, Inc., 2017.

Andrei Kulunchakov and Julien Mairal. A Generic Acceleration Framework for Stochastic Composite Optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Maxime Laborde and Adam Oberman. A Lyapunov analysis for accelerated gradient methods: from deterministic to stochastic case. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 108, pages 602–612. Proceedings of Machine Learning Research (PMLR), 2020.

Guanghui Lan. An optimal method for stochastic composite optimization. *Mathematical Programming*, 133(1):365–397, 2012.

Kiwon Lee, Andrew Cheng, Elliot Paquette, and Courtney Paquette. Trajectory of mini-batch momentum: batch size saturation and convergence in high dimensions. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:36944–36957, 2022.

Xilin Li. Black Box Lie Group Preconditioners for SGD. *arXiv preprint arXiv:2211.04422*, 2022.

Kaizhao Liang, Lizhang Chen, Bo Liu, and Qiang Liu. Cautious optimizers: Improving training with one line of code. *arXiv preprint arXiv:2411.16085*, 2024.

Licong Lin, Jingfeng Wu, Sham M. Kakade, Peter L. Bartlett, and Jason D. Lee. Scaling Laws in Linear Regression: Compute, Parameters, and Data. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 60556–60606. Curran Associates, Inc., 2024.

Chaoyue Liu and Mikhail Belkin. Accelerating SGD with momentum for over-parameterized learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A Scalable Stochastic Second-Order Optimizer for Language Model Pre-Training. *International Conference on Learning Representations (ICLR)*, 2024.

Jingyuan Liu, Jianlin Chen, Yao Huang, Xingyu Lu, Siming Xie, et al. Muon is Scalable for LLM Training. *arXiv preprint arXiv:2502.16982*, 2025. Moonlight Technical Report.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692*, 2019.

Nicolas Loizou and Peter Richtárik. Momentum and stochastic momentum for stochastic gradient, Newton, proximal point and subspace descent methods. *Comput. Optim. Appl.*, 77(3):653–710, 2020.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations (ICLR)*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *International Conference on Learning Representations (ICLR)*, 2019.

Alexander Maloney, Daniel A. Roberts, and James Sully. A Solvable Model of Neural Scaling Laws. *arXiv preprint arXiv:2210.16859*, 2024.

Martin Marek, Sanae Lotfi, Aditya Somasundaram, Andrew Gordon Wilson, and Micah Goldblum. Small Batch Size Training for Language Models: When Vanilla SGD Works, and Why Gradient Accumulation is Wasteful. *arXiv preprint arXiv:2507.07101*, 2025.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An Empirical Model of Large-Batch Training. *arXiv preprint arXiv:1812.06162*, 2018.

Sean McLeish, John Kirchenbauer, David Yu Miller, Siddharth Singh, Abhinav Bhatele, Micah Goldblum, Ashwinee Panda, and Tom Goldstein. Gemstones: A Model Suite for Multi-Faceted Scaling Laws. *arXiv preprint arXiv:2502.06857*, 2025.

Konstantin Mishchenko and Aaron Defazio. Prodigy: An expeditiously adaptive parameter-free learner. *arXiv preprint arXiv:2306.06101*, 2023.

Bruno Mlodozeniec, Pierre Ablin, Louis Béthune, Dan Busbridge, Michal Klein, Jason Ramapuram, and Marco Cuturi. Completed Hyperparameter Transfer across Modules, Width, Depth, Batch and Duration. *arXiv preprint arXiv:2512.22382*, 2025.

Depen Morwani, Nikhil Vyas, Hanlin Zhang, and Sham Kakade. Connections between Schedule-Free Optimizers, AdEMAMix, and Accelerated SGD Variants. *arXiv preprint arXiv:2502.02431*, 2025.

Yoonsoo Nam, Nayara Fonseca, Seok Hyeong Lee, Chris Mingard, and Ard A. Louis. An exactly solvable model for emergence and scaling laws in the multitask sparse parity problem. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 39632–39693, 2024.

Yu. Nesterov. On an approach to the construction of optimal methods of minimization of smooth convex functions. *Ekonom. i. Mat. Metody*, 24:509–517, 1988.

Yuri Nesterov. *Introductory lectures on convex optimization*. Springer, 2004.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Doklady Akademii Nauk SSSR*, 269(3):543–547, 1983.

OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

Antonio Orvieto and Robert M. Gower. In Search of Adam's Secret Sauce. *arXiv preprint arXiv:2505.21829*, 2025.

Antonio Orvieto, Jonas Kohler, and Aurelien Lucchi. The Role of Memory in Stochastic Optimization. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 356–366, 2020.

Matteo Pagliardini, Pierre Ablin, and David Grangier. The AdeMAMiX Optimizer: Better, Faster, Older. *International Conference on Learning Representations (ICLR)*, 2025.

Courtney Paquette and Elliot Paquette. Dynamics of stochastic momentum methods on large-scale, quadratic models. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:9229–9240, 2021.

Elliot Paquette, Courtney Paquette, Lechao Xiao, and Jeffrey Pennington. 4+3 Phases of Compute-Optimal Neural Scaling Laws. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 16459–16537. Curran Associates, Inc., 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32. Curran Associates, Inc., 2019.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training Deep Learning Models with Norm-Constrained LMOs. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 49069–49104. PMLR, 2025.

Steven T Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21(5):1112–1130, 2014.

Loucas Pillaud-Vivien, Alessandro Rudi, and Francis Bach. Statistical Optimality of Stochastic Gradient Descent on Hard Learning Problems through Multiple Passes. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving Discrepancies in Compute-Optimal Scaling of Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 100535–100570. Curran Associates, Inc., 2024.

Ofir Press and Lior Wolf. Using the Output Embedding to Improve Language Models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, 2017.

Shikai Qiu, Zixi Chen, Hoang Phan, Qi Lei, and Andrew Gordon Wilson. Hyperparameter Transfer Enables Consistent Gains of Matrix-Preconditioned Optimizers Across Scales. *arXiv preprint arXiv:2512.05620*, 2025a.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated Attention for Large Language Models: Non-linearity, Sparsity, and Attention-Sink-Free. *arXiv preprint arXiv:2505.06708*, 2025b.

Othmane Sebbouh, Robert M Gower, and Aaron Defazio. Almost sure convergence rates for Stochastic Gradient Descent and Stochastic Heavy Ball. In *Proceedings of Thirty Fourth Conference on Learning Theory (COLT)*, volume 134, pages 3935–3971. Proceedings of Machine Learning Research (PMLR), 2021.

Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. Benchmarking Optimizers for Large Language Model Pretraining. *arXiv preprint arXiv:2509.01440*, 2025.

Ishaan Shah, Anthony M Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J Shah, et al. Practical Efficiency of Muon for Pretraining. *arXiv preprint arXiv:2505.02222*, 2025.

C.J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G.E. Dahl. Measuring the Effects of Data Parallelism on Neural Network Training. *Journal of Machine Learning Research*, 20:1–49, 2019.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3): 379–423, 623–656, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Claude E. Shannon. Prediction and entropy of printed English. *Bell System Technical Journal*, 30(1): 50–64, 1951. doi: 10.1002/j.1538-7305.1951.tb01366.x.

Noam Shazeer. GLU Variants Improve Transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4596–4604. PMLR, 2018.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28, pages 1139–1147, 2013.

Ryosuke Takahira, Kumiko Tanaka-Ishii, and Łukasz Dębowski. Entropy rate estimates for natural language—a new extrapolation of compressed large-scale corpora. *Entropy*, 18(10):364, 2016. doi: 10.3390/e18100364.

Shohei Taniguchi, Keno Harada, Gouki Minegishi, Yuta Oshima, Seong Cheol Jeong, Go Nagahara, Tomoshi Iiyama, Masahiro Suzuki, Yusuke Iwasawa, and Yutaka Matsuo. ADOPT: Modified Adam Can Converge with Any beta2 with the Optimal Rate. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:72438–72474, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288*, 2023b.

Aditya Varre and Nicolas Flammarion. Accelerated SGD for Non-Strongly-Convex Least Squares. In *Proceedings of Thirty Fifth Conference on Learning Theory (COLT)*, volume 135 of *Proceedings of Machine Learning Research*, pages 2062–2126, 2022.

Sharan Vaswani, Francis Bach, and Mark Schmidt. Fast and Faster Convergence of SGD for Over-Parameterized Models and an Accelerated Perceptron. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics (ICML)*, volume 89 of *Proceedings of Machine Learning Research*, pages 1195–1204. PMLR, 2019.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Mujin Kwun, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and Stabilizing Shampoo using ADAM for Language Modeling. *International Conference on Learning Representations (ICLR)*, 2025.

Xi Wang and Laurence Aitchison. How to set AdamW's weight decay as you scale model and dataset size. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 62222–62250. PMLR, 2025.

Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic Pretraining Optimizers and Where to Find Them. *arXiv preprint arXiv:2509.02046*, 2025.

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie E Everett, Alexander A Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, et al. Small-scale proxies for large-scale Transformer training instabilities. In *International Conference on Learning Representations (ICLR)*, 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. In *International Conference on Learning Representations (ICLR)*, 2024.

Lechao Xiao. Rethinking conventional wisdom in machine learning: From generalization to scaling. *arXiv preprint arXiv:2409.15156*, 2024.

Zeke Xie, Issei Sato, and Masashi Sugiyama. Understanding and scheduling weight decay. *OpenReview*, 2020.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 2020.

Yan Yan, Tianbao Yang, Zhe Li Li, Qihang Lin, and Yi Yang. A Unified Analysis of Stochastic Momentum Methods for Deep Learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2955–2961. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/410.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025.

Greg Yang and Edward J Hu. Tensor Programs IV: Feature Learning in Infinite-Width Neural Networks. In *International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 11727–11737. PMLR, 2021.

Greg Yang and Etai Littwin. Tensor programs ivb: Adaptive optimization in the infinite-width limit. *arXiv preprint arXiv:2308.01814*, 2023.

Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 17084–17097. Curran Associates, Inc., 2021.

Dmitry Yarotsky. Corner Gradient Descent. *arXiv preprint arXiv:2504.12519*, 2025.

Dmitry Yarotsky and Maksim Velikanov. SGD with Memory: Fundamental Properties and Stochastic Acceleration. *International Conference on Learning Representations (ICLR)*, 2025.

Huizhuo Yuan, Yifeng Liu, Shuang Wu, Xun Zhou, and Quanquan Gu. MARS: Unleashing the Power of Variance Reduction for Training Large Models. *International Conference on Learning Representations (ICLR)*, 2025.

Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sachdeva Sachdeva, George Dahl, Christopher Shallue, and Roger Grosse. Which Algorithmic Choices Matter at Which Batch Sizes? Insights From a Noisy Quadratic Model. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32. Curran Associates, Inc., 2019.

Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean Foster, and Sham Kakade. How Does Critical Batch Size Scale in Pre-training? *International Conference on Learning Representations (ICLR)*, 2025a.

Yushun Zhang, Congliang Chen, Naichen Shi, Ruoyu Sun, and Zhi-Quan Luo. Adam Can Converge Without Any Modification On Update Rules. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:28386–28399, 2022.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Diederik P Kingma, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use Fewer Learning Rates To Gain More. *International Conference on Learning Representations (ICLR)*, 2025b.

Rosie Zhao, Depen Morwani, David Brandfonbrener, Nikhil Vyas, and Sham Kakade. Deconstructing What Makes a Good Optimizer for Autoregressive Language Models. *International Conference on Learning Representations (ICLR)*, 2025.

APPENDIX OVERVIEW

Broader Impact Statement. The work presented in this paper is an empirical study on how to scale hyperparameters for different optimisers in the context of LLM pretraining. While a common positive foreseeable impact of algorithms and hyperparameter schedules with better scaling properties is that they would allow more efficient model training and hence lower energy consumption of AI, it is important to acknowledge that, 1) this research itself had had a negative environmental impact using around 22 years of compute with a H-100 GPU (details in the appendix) 2) many times across history, cost-lowering technological improvements have nevertheless led to an increase in consumption due to the commonly named Jevons paradox. Regarding our experiments, we include results on language models trained on a public language dataset and a synthetic dataset. We do not release any pretrained models, but because the dataset we used is based on Common Crawl, which has potential issues of fairness, harm, accountability, and transparency, we may be contributing to reinforcing these issues by supporting the creation of FineWeb as a standard in NLP, thereby obfuscating these concerns. Finally, we anticipate that one potential negative impact of this line of research is that fundamental advances in scaling will benefit more entities with greater computational resources, thereby enhancing the centralisation of technological power.

This appendix provides supplementary material organized into the following sections.

Appendix A: Related Work. We survey related work on adaptive optimization methods, scaling laws for language models, and momentum-based acceleration techniques. This includes discussion of ADAM variants, second-order methods, and prior work on hyperparameter scheduling.

Appendix B: Synthetic Experiments. We describe the Power-Law Random Features (PLRF) model and its Mixture-of-Experts extension (MOE-PLRF), which serve as theoretical testbeds for studying optimizer behavior. These models capture the essential structure of language data while permitting rigorous analysis.

Appendix C: Experimental Setup. We detail our scaling setup, training configuration, hyperparameter scaling methodology, and compute budget calculations. This section also includes detailed specifications of the Enoki model architecture (attention mechanisms, feed-forward networks, normalization, initialization strategy, and parameter count formulas). This section provides the foundation for fair comparison across model scales.

Appendix D: Algorithms. We present complete algorithmic descriptions for all optimizers studied: ADAMW, LOG-ADAMW, MUON, ADEMAMIX, DANA variants, and the hardened ADANA family (DANA-MK4, DANA-STAR, DANA-STAR-MK4). Each algorithm includes its specific hyperparameters and update rules.

Appendix E: Scaling Laws. We analyze single versus broken power-law fits, compare compute formulas (Kaplan $6P$, Chinchilla $6D$, DeepSeek M), examine optimizer outscaling behavior, and survey measured exponents from the literature. This section also includes sensitivity analysis for the spectral dimension parameter κ.

Appendix F: Baselining Procedure. We describe our systematic hyperparameter search procedure, including the two-stage learning rate search strategy, weight-decay grid, and other training details. Learning rate scaling laws for the Enoki architecture are derived and analyzed.

Appendix G: Qwen3 Experiments. We present comprehensive results on the Qwen3 architecture, including model implementation details, scaling law fits, learning rate scaling analysis, and a comparison of scaling rules between Enoki and Qwen3. This validates that our findings generalize beyond the Enoki architecture.

Appendix H: Batch Sizes Experiments We extend our analysis of ADANA to larger batch and show similarly that ADANA attains strong performance which increases with scale in a larger batch setting.

Appendix I: Data Exponent Analysis. We measure the activation covariance spectra in transformers to estimate the data exponent ρ, connecting empirical observations to PLRF theory and explaining the optimal range of κ values.

Appendix J: AdEMAMix Correspondence. We analyze the relationship between ADEMAMIX's warmup schedules and DANA's logarithmic-time scheduling, showing how proper scheduling enables similar performance gains.

Appendix K: Additional Weight Decay Details. We give additional details on the sweeps performed for our weight-decay study and additional intuition on why log-time weight-decay might improve performance.

Appendix L: Building ADANA Theory. We develop the theoretical foundations of ADANA, including the role of log-momentum in Nesterov acceleration and the analysis of sparse gradient challenges.

Appendix M: Connection With Hilberg Exponent. We discuss a connection between the Hilberg exponent —a concept from information theory—and the logarithmic time schedules in ADANA.

Appendix N: Failure Modes & Hardened Variants. We detail the failure modes of naive logarithmic-time scheduling under sparse gradients and describe the design of hardened variants (DANA-MK4, DANA-STAR) that address these issues.

Appendix O: Nesterov Formulations. We derive the equivalence between different formulations of Nesterov's accelerated method and show how they relate to the DANA family of algorithms.

Appendix P: Proofs. We provide complete proofs of theoretical results stated in the main text, including convergence guarantees and stability analysis.

Appendix R: Compute Usage. We report the total compute used for all experiments, broken down by model size and optimizer.

A ADDITIONAL RELATED WORK

Momentum and Accelerated SGD Methods. Prior work has established convergence guarantees for stochastic gradient descent with classical momentum (SGD-M) (i.e., momentum parameters held fixed) in both strongly convex and non-strongly convex settings Flammarion and Bach (2015); Gadat et al. (2016); Orvieto et al. (2020); Yan et al. (2018); Sebbouh et al. (2021)[7]. For quadratic objectives, SGD-M exhibits linear convergence of the iterates (though not in L^2) under additional assumptions Loizou and Richtárik (2020). Under additional conditions on the stochastic gradient noise, linear convergence to a neighborhood of the optimum has been shown in Kidambi et al. (2018); Can et al. (2019). The batch size plays a critical role in the performance of both SGD and SGD-M. For small batch sizes, SGD-M does not necessarily outperform plain SGD Kidambi et al. (2018); Paquette and Paquette (2021); Zhang et al. (2019), whereas acceleration can emerge in the large-batch regime Lee et al. (2022); Bollapragada et al. (2024); Dang et al. (2025).

Convergence results for stochastic Nesterov's accelerated method Nesterov (2004) (S-NAG) have been established for both strongly convex and non-strongly convex settings Kulunchakov and Mairal (2019); Assran and Rabbat (2020); Aybat et al. (2020). Under stronger assumptions—such as the strong growth condition Vaswani et al. (2019) or additive noise on stochastic gradients Laborde and Oberman (2020)—convergence to the optimum at an accelerated rate can be guaranteed. As noted in Thm. 7 Even et al. (2021) (see also references therein), a naive implementation of stochastic Nesterov acceleration fails to converge in the non-strongly convex setting.

ADANA, along with ADEMAMIX Pagliardini et al. (2025), can be viewed as special instances of "accelerated SGD" methods (see nonexhaustive list Jain et al. (2018); Allen-Zhu (2017); Ghadimi and

[7]This is a non-exhaustive list of work on stochastic momentum methods.

Lan (2012; 2013); Liu and Belkin (2020); Gupta et al. (2024); Defazio et al. (2024); Krichene and Bartlett (2017)). Recent work by Morwani et al. (2025) establishes connections between ADEMAMIX, SCHEDULE-FREE ADAMW Defazio et al. (2024); Defazio (2025), and accelerated SGD variants. In particular, the authors propose removing the EMA on the short-term average in ADEMAMIX —an approach similar in spirit to ADANA —and show that this simplified variant remains competitive in practice. The Lion algorithm (Chen et al., 2023) is also related to ADANA as it combines a gradient term and momentum term in the update. However, it doesn't scale the damping schedule (taken constant) neither uses logarithmic time for the momentum (also taken constant). It additionally uses a sign update instead of a division by the second moments.

On the theoretical side, Varre and Flammarion (2022) study accelerated SGD for least-squares regression, demonstrating that acceleration can be achieved using schedules similar to the DANA-CONSTANT schedule introduced in Ferbach et al. (2025). Relatedly, Yarotsky and Velikanov (2025); Yarotsky (2025) analyze the acceleration of stochastic momentum methods on quadratic problems under power-law spectral assumptions.

Other Momentum Based Algorithms in Practice Sutskever et al. (2013) first proposed to use Nesterov momentum (Nesterov, 1983) to train neural networks in practice. In the same line of work NADAMW (Dozat, 2016) combines Nesterov updates with the ADAM algorithm and shows improved performance against the baseline ADAM. ADAN is similar to NADAMW in how it combines Nesterov momentum with ADAM but averages gradient differences rather than raw gradients in NADAMW. Yuan et al. (2025) proposes MARS, a variance reduction momentum algorithm. MARS originally uses two gradient computations at each step, hence the authors propose MARS-APPROX, an approximated version closer to previously discussed momentum-based frameworks. Overall, while these works incorporate Nesterov momentum in various ways, we highlight a crucial point of distinction between our work and these: all these works focus on the strongly convex formulation of Nesterov where the momentum β is constant, while we focus on the convex form of Nesterov where the momentum term $\beta \approx 1 - 1/t$ goes to 1 in a log-time manner. This creates a significant difference and, to the best of our knowledge, using the convex form of Nesterov with log-time schedule in a practical algorithm has not been explored before. As we explain, any fixed momentum schedule creates a mismatch with respect to the power-law nature of real data and is bound to behave like no momentum at scale. On the other hand, we argue that this power-law structure brings the problem closer to a convex problem. Additionally, we highlight that logarithmic-time momentum requires two crucial ingredients for stability in practice: an additional gradient term in the update and a damping schedule on the momentum. We extensively discuss and ablate these in Figure 3.

Scaling laws and PLRF. Scaling laws and compute-optimality for the PLRF model under one-pass SGD have been analyzed by Paquette et al. (2024); Bordelon et al. (2024); Lin et al. (2024) with an extension to feature learning in Bordelon et al. (2025); Defilippis et al. (2025). Particularly, Paquette et al. (2024); Ferbach et al. (2025) developed scaling laws for SGD and DANA under a deterministic equivalent and showed that there exists 4 distinct phases. They used their scaling law to find compute-optimality. Related work on ridge-regression gradient descent includes Cui et al. (2021) and Defilippis et al. (2024). Other theoretical guarantees for scaling laws beyond PLRF, typically for gradient flow, have been established in works such as Nam et al. (2024). Other works have studied SGD under power-law covariance including Carratino et al. (2018); Dieuleveut and Bach (2016); Pillaud-Vivien et al. (2018) on the least-squares problem.

Architecture Scaling. In McLeish et al. (2025), the authors study the impact of architecture choices such as depth vs width scaling or optimizers choices for training large language models. They release a model suite of checkpoints and highlight the sensitivity of scaling laws fits to training choices, significantly impacting the final results.

Hyperparameters Transfer at Scale. There is extensive literature on the optimal scaling of optimizer hyperparameters for large neural networks, with particular emphasis on initialization and learning rate Yang and Hu (2021); Everett et al. (2024); Wortsman et al. (2023), batch size McCandlish et al. (2018); Shallue et al. (2019); Zhang et al. (2025a), and the epsilon parameter in adaptive optimizers Yang and Littwin (2023); Wortsman et al. (2023); Everett et al. (2024). In contrast, momentum hyperparameters in large-scale models are typically treated as fixed constants rather than

scaled quantities. Notably, the GPT-3 model Brown et al. (2020) used $\beta_2 = 0.95$ instead of the more common values $\beta_2 = 0.99$ or 0.999, a choice that may improve stability at large batch sizes.

A recent line of work focuses on scaling optimizer hyperparameters—such as learning rate and weight-decay—to ensure stable and non-degenerate training dynamics as model size increases. A seminal contribution is Yang et al. (2021), which demonstrates how to optimally transfer hyperparameters from small-scale settings, where tuning is feasible, to large-scale regimes, where tuning is computationally prohibitive. Building on this framework, Dey et al. (2025) introduce the CompleteP parametrization, extending prior approaches to enable *depth transfer*. Mlodozeniec et al. (2025) further refine this framework by enabling transfer across batch size and number of training iterations, and additionally demonstrate the feasibility of per-module hyperparameter transfer, yielding substantial performance gains.

Batch Scaling. McCandlish et al. (2018) showed that training parallelism scales nearly linearly with batch size up to a critical batch size determined by the noise scale of the problem, beyond which performance saturates. Zhang et al. (2025a) study how this critical batch size scales during pretraining and show that it correlates more strongly with dataset size than with model size. Marek et al. (2025) investigate the impact of small batch sizes in training, showing that smaller batches often lead to better final losses and that vanilla SGD remains stable when training LLMs with very small batch sizes. They also propose a scaling rule for ADAMW hyperparameters in the small-batch regime. In particular, they suggest scaling ADAMW 's β_2 with batch size B according to $\log(\beta_2) \propto B$, so that the second-moment EMA averages over a comparable number of tokens across batch sizes. Since their experiments fix the total number of training tokens D, we have $B = D/T$, and thus their result is equivalent to $\log(\beta_2) \propto 1/T$, recovering our proposed logarithmic-time scaling $\beta_2 = 1 - \frac{\delta}{T}$.

Optimizers Benchmarking and Performance Comparison. In Semenov et al. (2025), the authors compare the performance of SOAP (Vyas et al., 2025), MARS (Yuan et al., 2025), D-MUON (Liu et al., 2025), SCHEDULE-FREE ADAMW (Defazio et al., 2024), LION (Chen et al., 2023), ADOPT (Taniguchi et al., 2024), PRODIGY (Mishchenko and Defazio, 2023), SOPHIA (Liu et al., 2024), and SIGNUM (Bernstein et al., 2018). They report strong and consistent performance from MARS and ADEMAMIX relative to ADAMW, particularly as the number of training iterations increases, while other optimizers exhibit similar or worse performance. The authors also study the influence of batch size, warm-up, and weight-decay, and find that the use of z-loss has little impact and can even degrade performance.

Similarly, Wen et al. (2025) compare the performance of MUON, NADAMW (Dozat, 2016), CAUTIOUS (Liang et al., 2024), ADAM-MINI (Zhang et al., 2025b), SCION (Pethick et al., 2025), KRON (Li, 2022), SOAP, and SOPHIA. They find that matrix-preconditioned optimizers perform best at small scale, but that their advantages diminish relative to ADAMW as scale increases. They attribute this discrepancy with Semenov et al. (2025) to the use of larger batch sizes, which tend to favor matrix-based methods.

While both Semenov et al. (2025); Wen et al. (2025) show that MUON achieves over $\sim 10\%$ compute gains relative to ADAMW at scale, recent work in Qiu et al. (2025a) demonstrates that, under appropriate weight-decay scaling, MUON and SHAMPOO can yield improvements of $\sim 30\%$-40%. Specifically, Qiu et al. (2025a) advocate combining μP scaling (Yang et al., 2021) with weight-decay scaling of the form $\lambda \sim \frac{\omega}{\text{lr} \times \text{width}}$. Shah et al. (2025) similarly report substantial gains for MUON over ADAMW, showing in particular that MUON remains highly effective at large batch sizes, thereby improving the Pareto frontier of compute–time trade-offs under a fixed validation-loss constraint. Finally, Zhao et al. (2025) show that, aside from SGD, many optimizers—including LION, SIGNUM, and ADAFACTOR Shazeer and Stern (2018)—exhibit performance comparable to ADAMW.

Choosing ADAMW Hyper-parameters Other works focused on designing efficient or simplified choices of hyper-parameters for ADAMW. Zhang et al. (2022) identified a condition on β_2 to be $\frac{\beta_1(s)}{\sqrt{\beta_2(s)}} < 1$. More recently Orvieto and Gower (2025) highlighted that picking $\beta_1 = \beta_2$ has empirical and theoretical benefits, hence aligning with our schedules choices for ADANA. Additionally, Zhao et al. (2025) show that ADAMW with tied $\beta_1 = \beta_2$ behaves similarly to SIGNUM.

Weight-decay Scheduling and Scaling.

- **Constant weight-decay.** In Mlodozeniec et al. (2025), the Complete(d) parametrization recommends for hidden and unembedding weights a peak learning rate scaled as $\gamma^* \asymp m_N^{-1} m_L^{\alpha-1} \sqrt{\frac{m_B}{m_D}}$ where m_N, m_B, m_D scale respectively as width, batch size and token count and $\alpha \in [1/2, 1]$ controls the depth scaling. They predict similarly the decoupled weight-decay λ to scale as $\lambda \asymp m_N \sqrt{\frac{m_B}{m_D}}$ on hidden weights. Note that they are not in the independent weight-decay formulation, hence λ is hit by γ^* in the update. With $\alpha = 1$ and rewriting the number of iterations $T = \sqrt{\frac{m_D}{m_B}}$, we see that $\lambda \asymp \frac{1}{\gamma^* \times T}$ which matches our scaling rule $\lambda \asymp \frac{1}{T}$ in the independent weight-decay formulation. Xiao (2024) study alternative scaling rules for learning rate and weight decay. They compare μP scaling with width-independent weight decay to a regime in which both the peak learning rate and the independent weight decay scale inversely with model width, $\gamma^*, \lambda \propto m_N^{-1}$. They show that the latter scaling yields better performance at large widths, highlighting the importance of identifying appropriate hyperparameter scaling laws to optimize performance at scale.
- **Weight-decay schedules.** Defazio (2025) shows that constant weight-decay scheduling causes gradient norms to increase near the end of training and proposes to decay the weight-decay schedule on normalized layers as $\lambda(t) \propto \gamma(t)\lambda$ to counter this effect, resulting in lower loss along training. While this schedule decays when using decaying schedules on $\gamma(t)$, it generally does not match the schedule $1/t$. In Xie et al. (2020), the authors propose an adaptive method for weight-decay of the form $\lambda(t) \propto \frac{1}{v_t^\rho}$ where v_t is the second moment of the gradients and ρ typically set to $1/2$.

B Synthetic Experiments

In this section, we discuss two important toy setups studied extensively in the theoretical scaling law literature because of their rich behavior and because they phenomenologically capture aspects of scaling law setups that occur in practice.

B.1 Power-law Random Features (PLRF)

The four-parameter model called *power-law random features* (PLRF) (Maloney et al., 2024; Bahri et al., 2024; Paquette et al., 2024) is studied extensively in the theoretical scaling law literature because of its rich behavior and because it phenomenologically captures many aspects of scaling law setups that occur in practice (Paquette et al., 2024; Bordelon et al., 2024; Lin et al., 2024).

This model assumes the following. For a data vector $x \in \mathbb{R}^v$ we embed this vector in $\mathbb{R}^d$ using a matrix $W \in \mathbb{R}^{v \times d}$ and construct noiseless targets by dotting a fixed $b \in \mathbb{R}^v$ with the sample x. This leads to the formal problem statement:

$$\min_{\theta \in \mathbb{R}^d} \left\{ \tfrac{1}{2} \mathcal{P}(\theta) \overset{\text{def}}{=} \tfrac{1}{2} \mathbb{E}_x \left[(\langle W^T x, \theta \rangle - \langle x, b \rangle)^2 \right] \right\}. \tag{3}$$

The matrix W allows the model to have variable capacity (d) independent of the data dimension, and we choose the matrix W to have entries distributed as $N(0, 1/d)$. The key structural assumptions are as follows.

Assumption B.1 (Power-law data and targets, ρ and η)**.** *The samples $x \in \mathbb{R}^v$ are distributed according to $(x_j) \sim j^{-\rho} z_j$ for all $1 \leq j \leq v$ and $\{z_j\}_{j=1}^v \sim N(0, 1)$. The targets are scalars constructed by dotting the sample x with a signal $b \in \mathbb{R}^v$ whose entries $(b_j) = j^{-\eta}$.*

Power-law type data distributions are ubiquitous in language, vision, and many other tasks, and these distributions are largely responsible for making this model phenomenologically similar to scaling law setups (Maloney et al., 2024, Fig. 2,3). Moreover, this setup allows for a theoretical analysis of compute-optimal neural scalings – finding optimal allocation of d given a fixed compute budget (Paquette et al., 2024; Ferbach et al., 2025). It was shown in (Paquette et al., 2024) for a large portion of the (ρ, η) this model captures the Chinchilla scaling rule. Without the random matrix W, (ρ, η) are related to what is known in the literature as source and capacity conditions (Caponnetto and De Vito, 2007; Carratino et al., 2018; Dieuleveut and Bach, 2016; Pillaud-Vivien et al., 2018).

The hidden dimensionality v is assumed to be large and proportionate to d, so that $v/d \to r \in (1, \infty)$. In the case that $2\rho > 1$, this assumption can be relaxed, in that one can take v larger as a function of

d or even $v = \infty$. It should be noted that in many scaling law setups, such as (Hoffmann et al., 2022), the task scales with the parameter count, so that it is natural to assume v grows as d grows.

The PLRF model exhibits scaling laws: the population risk decays as a power of training time, with the exponent depending on ρ and β. DANA-DECAYING and more generally the DANA algorithms were studied under the PLRF model and were shown to exhibit significant exponent speed-up when $\kappa = 1/(2\rho)$ with ρ the data exponent. In Appendix I, we measure the activation covariance spectra in transformers and find power-law exponents consistent with the optimal κ values observed in practice.

B.2 MIXTURE-OF-EXPERTS PLRF (MOE-PLRF)

We introduce the *Mixture-of-Experts (MOE) PLRF* to add sparse gradients coming from a mixture-of-experts. In this setup, we have m experts and each expert has their own PLRF model with the data being generated with the same power-law exponents (ρ, η). Each sample gets routed to one expert's parameters and only that expert's parameters will be updated. We denote the ith expert's parameter by $\theta^{(i)} \in \mathbb{R}^d$. We define the probability of routing to expert i as power-law distributed with exponent $\zeta > 0$, that is,

$$p(i) = \frac{i^{-\zeta}}{\sum_{j=1}^{m} j^{-\zeta}}. \tag{4}$$

The experts each share the same random features matrix $W \in \mathbb{R}^{v \times d}$ where $W_{ij} \sim N(0, 1/d)$. The optimization problem we are interested in solving is

$$\begin{gathered} \min_{\Theta \in \mathbb{R}^{d \times m}} \mathbb{E}_{i \sim p, x} \left[\tfrac{1}{2} \left(\langle W^T x, \theta^{(i)} \rangle - \langle x, b \rangle \right)^2 \right], \\ \text{where } \Theta = [\theta^{(1)} | \theta^{(2)} | \dots | \theta^{(m)}]. \end{gathered} \tag{5}$$

For a batch of B samples $\{x_t^k, b_t^k\}_{k=1}^B$ such that $(x_t^k)_j \sim N(0, j^{-2\alpha})$ and $(b_t^k)_j = j^{-\beta}$ for every k, we assign each (x_t^k, b_t^k) to an expert $i_t^k \sim p$. This can be represented by a routing matrix $R \in \mathbb{R}^{m \times B}$,

$$R_{ik} = \begin{cases} 1, & \text{if expert } i \text{ is assigned to sample } k \\ 0, & \text{otherwise.} \end{cases}$$

Let $X_t \in \mathbb{R}^{v \times B}$ be the matrix of data inputs where column k contains x_t^k and let $Y_t \in \mathbb{R}^B$ be the vector of targets where the kth entry contains $\langle x_t^k, b_t^k \rangle$.

Frequently-selected experts (small i) receive updates with high probability, while rarely-selected experts (large i) receive updates with probability proportional to $i^{-\zeta}$. The Zipfian selection distribution mimics the frequency distribution of tokens in natural language: common words appear frequently while rare words appear infrequently.

This setup creates exactly the sparse gradient structure used to analyze ADAM and its variants with log-time β_1 and β_2 (Appendix L.2.1 and Appendix N). Particularly, we show that divergent behavior for ADAM-like algorithms when $\beta_1(t) = 1 - \delta/(t + \delta)$ and β_2 held fixed independent of d (see Theorem L.1 in Section L.2.1).

B.3 EXPERIMENTAL VALIDATION

Figure 9 shows the population risk versus training time for different values of κ. When κ is too small, acceleration is insufficient and convergence mirrors ADAM. When κ is too large, instability dominates and the loss grows. Between these regimes lies a stable band where acceleration occurs. The center of this band coincides with the theoretical prediction $\kappa = 1/(2\rho)$ derived from the PLRF scaling law analysis.

We run MOE-PLRF experiments with $m = 100$ experts, data exponent $\rho = 1.5$, Zipf exponent $\zeta = 1.0$ for expert selection, and $v = 1000$ features per expert.

Connection to Transformer Training. The MOE-PLRF model connects to transformer training in several important ways. First, token embeddings can be viewed through this lens: each vocabulary token is an "expert" that receives gradients only when that token appears in the training batch. The

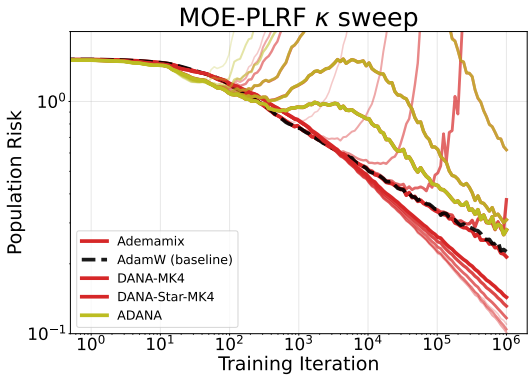


Figure 9: **MOE-PLRF Population risk vs. log-time for different κ values.** The optimal κ matches the theoretical prediction $\kappa = 1/(2\rho)$. ADAM baseline shown as a black dashed line.

Zipfian distribution of natural language means rare tokens behave like tail experts in MOE-PLRF. Second, attention patterns create similar sparsity: positions that rarely attend to certain keys have sparse gradient patterns for those key projections. Third, modern architectures increasingly use explicit mixture-of-experts layers, where the connection is direct.

The success of DANA-STAR on MOE-PLRF predicts its benefits for transformer training where sparsity is present and the gradient structure becomes more pronounced.

C EXPERIMENTAL SETUP: SUMMARY

We evaluate our optimizers on decoder-only transformer architectures trained on the FineWeb dataset (Penedo et al., 2024) with cross-entropy loss. We adapt the implementation used by Semenov et al. (2025).

C.1 SCALING SET-UP

Model architecture We used two different architectures for our experiments: Enoki and Qwen3 Yang et al. (2025). We describe below the Enoki model while details on Qwen3 can be found in Appendix G. The Enoki model is a GPT-3-like decoder-only transformer with the following modern improvements: Rotary Positional Embedding (RoPE) (Su et al., 2024), QK-LayerNorm applied before RoPE (Henry et al., 2020), residual initialization scaling $1/\sqrt{\text{depth}}$, pre-layer-norm on attention and MLP (Xiong et al., 2020), and no weight tying (Press and Wolf, 2017). We do not use z-loss (Chowdhery et al., 2023).

To study the performance of optimizers across scales, we design a scaling rule that specifies how to increase each component of the architecture as the number of parameters grows. Following Charles et al. (2025), the Enoki model fixes the head dimension at 64 and scales the number of transformer layers n_{layer} and attention heads (# of heads $=$ heads) proportionally:

$$n_{\text{layer}} = \lfloor 3/4 \times \text{heads} \rfloor. \tag{6}$$

In this way, the model's width $n_{\text{embd}} = \text{heads} \times 64$ grows proportionally with the depth n_{layer}.

Initialization Strategy We use the ScaledGPT initialization scheme. Weights from embedding matrices and linear layers are initialized with mean zero and variance $1/\text{fan}_{\text{in}}$. Weights from projections on the residual branch (attention output and MLP output) have an additional depth-dependent scaling factor, yielding variance $1/(2 \cdot \text{fan}_{\text{in}} \cdot n_{\text{layer}})$. All bias terms are initialized to zero.

Compute Budget and Chinchilla Scaling Denote D the total number of parameters (including embeddings), P the number of non-embedding parameters, N the number of training tokens and C the compute budget in floating-point operations. The compute budget is usually approximated as $C = 6 \cdot N \cdot P$ Kaplan et al. (2020) ignoring optimizer FLOPs, which are typically a small fraction

of total FLOPs). For the Enoki model, $P = 12 \cdot n_{\text{embd}}^2 \cdot n_{\text{layer}}$, and the total parameter count is $D = P + 2 \cdot n_{\text{embd}} \cdot 50304$.

Importantly, we follow Chinchilla-optimal scaling (Hoffmann et al., 2022) where training tokens and parameters are scaled proportionally: $N = 20 \cdot D$.

Training Configuration All experiments use batch size 32 sequences with sequence length 2048, yielding 65,536 tokens per step. The vocabulary size is 50,304 (GPT-2 tokenizer). All runs use the same random seed for initialization and data ordering. We additionally use warmup and cosine decay on the learning rate as well as decoupled weight-decay (Loshchilov and Hutter, 2019). More details can be found in Appendix F.2.

C.2 Hyperparameter Scaling

To enable fair comparison across model scales, we fit power-law scaling rules to the optimal learning rates found at each model size. For each optimizer, we collect the top-K ($K = 5$) learning rates (ranked by final validation loss) at each scale and fit a saturated power-law using weight $(K - k + 1)^2 \times P$ for the k-th best loss at non-embedding parameters P:

$$\gamma(P) = a \cdot (b + P)^d, \tag{7}$$

where P is the number of non-embedding parameters and a, b, d are fitted coefficients with $a, b > 0$. The saturation term b prevents divergence at small scales, while the exponent d (typically between -0.4 and -0.9) controls how aggressively learning rate decreases with model size.

We observe that different optimizers exhibit distinct scaling behavior. Notably, for the Enoki architecture, AdamW exibits a learning rate scaling as square root of non-embedding parameters P with $d = -0.52$, aligned with recent studies (Yang et al., 2021; Dey et al., 2025; Mlodozeniec et al., 2025). Muon maintains relatively high learning rates at scale ($d \approx -0.42$), while Adana ($\kappa = 0.85$) requires more aggressive reduction ($d \approx -0.62$). For Qwen3, the smallest model sizes (below 100M non-embedding parameters) deviate from power-law scaling and are excluded from the fit. The complete fitting methodology, explicit formulas for all optimizers, and visualization of the scaling laws are provided in Appendix F.3.

C.2.1 Compute Saved Analysis

To quantify the practical benefit of each optimizer, we measure "compute saved" through the compute multiplier metric. Let a scaling ladder of compute budgets $C_1, C_2, \cdots C_k$ where C_i corresponds to training at the compute-optimal horizon (Hoffmann et al., 2022) model size i. For each optimizer B under study (e.g. Adana, Muon, Ademamix...) and each model size i, we have access to the final loss $\mathcal{L}_i^B$ of optimizer B under budget C_i. We can do the same for any baseline algorithm A (often AdamW or AdamDW) to get L_i^B. This naturally gives rise to an affine by part function $\mathcal{L}^B : \mathbb{R} \to \mathbb{R}$ such that $\forall i \in \{1, \cdots, k\}$, $\mathcal{L}^B(C_i) = \mathcal{L}_i^B$ with linear interpolation between adjacent points. Outside the observed range, the function is extended to $+\infty$ and $-\infty$ using the slope of the nearest segment, i.e. with slope respectively $\frac{\mathcal{L}_k^B - \mathcal{L}_{k-1}^B}{C_k^B - C_{k-1}^B}$ and $\frac{\mathcal{L}_2^B - \mathcal{L}_1^B}{C_2^B - C_1^B}$.

Fix a model scale i and let $C^B = C_i$ denote the compute budget used by optimizer B, yielding final loss $\mathcal{L}^B(C^B)$. We define C^A as the compute required by a baseline optimizer A to reach the same loss,

$$\mathcal{L}^A(C^A) = \mathcal{L}^B(C^B),$$

where C^A is obtained by inverting the piecewise-affine extrapolation of $\mathcal{L}^A$.

The compute multiplier and compute efficiency (or relative compute saved) are then respectively

$$\frac{C^A}{C^B} \quad \text{and} \quad \frac{C^A - C^B}{C^B},$$

which quantifies the additional compute optimizer A would need to match optimizer B at compute C^B. Positive values indicate improved compute efficiency. (see also Qiu et al. (2025a); Liu et al. (2025)).

C.3 Architecture Details

C.3.1 Enoki Model Implementation

Table 4: **ScaledGPT Initialization Strategy.** The ScaledGPT initialization scheme uses fan-in variance scaling for general linear layers and embeddings, with additional depth-dependent scaling for residual projection layers. Here L denotes n_layer, d_{embd} the embedding dimension, d_{qkv} the total QKV dimension, and d_{hidden} the MLP hidden dimension.

Layer Type	Init Std Dev
Token Embedding	$\sigma = 1/\sqrt{d_{\text{embd}}}$
LM Head	$\sigma = 1/\sqrt{d_{\text{embd}}}$
Attention QKV (c_attn)	$\sigma = 1/\sqrt{d_{\text{embd}}}$
Attention Output (c_proj)	$\sigma = 1/\sqrt{2 \times d_{\text{qkv}} \times L}$
MLP Input (c_fc)	$\sigma = 1/\sqrt{d_{\text{embd}}}$
MLP Output (c_proj)	$\sigma = 1/\sqrt{2 \times d_{\text{hidden}} \times L}$
All Bias Terms	$b = 0$

The Enoki model takes its name from the enoki mushroom, reflecting its architectural design with a relatively small head dimension of 64 and hence many heads to keep the width comparable to the depth.

The transformer architecture underlying Enoki follows the GPT-3 paradigm (Brown et al., 2020) with several modern stabilization improvements inspired by recent scaling studies (Wortsman et al., 2023). We retain the GPT-3 tokenizer and vocabulary size of $50,304$ tokens.

Attention mechanism. The attention implementation employs multi-head self-attention with causal masking. Queries, keys, and values are computed through separate linear projections from the input hidden state. A critical stabilization technique is the application of QK-LayerNorm (Dehghani et al., 2023) to the query and key vectors *before* applying Rotary Position Embeddings (RoPE) (Su et al., 2024).

Feed-forward network. The MLP component uses a standard two-layer architecture with GELU activation (Hendrycks and Gimpel, 2016). The hidden dimension is set to $4 \times d_{\text{model}}$, following the GPT convention. All linear layers omit bias terms, following Chowdhery et al. (2023).

Normalization and residual connections. Enoki employs pre-LayerNorm architecture, applying LayerNorm before each attention and MLP sublayer. A final LayerNorm precedes the language modeling head. No bias terms are used in the LayerNorm layers.

Scaling rule. The Enoki scaling rule fixes the head dimension and scales width and depth according to the number of heads h:

$$d_h = 64 \text{ (fixed)} \tag{8}$$
$$n_{\text{head}} = h \tag{9}$$
$$n_{\text{layer}} = \lfloor 3h/4 \rfloor \tag{10}$$
$$n_{\text{embd}} = 64h \tag{11}$$
$$d_{\text{mlp}} = 4 \times n_{\text{embd}} \tag{12}$$

This yields a relatively shallow but wide architecture compared to standard scaling rules, which we found beneficial for optimizers that benefit from larger gradient signals per layer.

Parameter count formulas. For Enoki:

$$\text{per_layer} = 12 \times n_{\text{embd}}^2 \tag{13}$$
$$\text{non_emb} = n_{\text{layer}} \times \text{per_layer} \tag{14}$$
$$\text{total} = \text{non_emb} + 2 \times n_{\text{embd}} \times V \tag{15}$$

Table 5: **Architectural Details of the Enoki Model.** The Enoki model fixes head dimension to 64 and scales width and depth linearly with the number of attention heads. MLP hidden dimension is $4 \times d_{\text{embd}}$. Head count is $\frac{4}{3} n_{\text{layer}}$.

D (Total)	Layers	Embd Dim	MLP Hidden
70.39M	6	512	2,048
140.97M	9	768	3,072
254.02M	12	1,024	4,096
423.69M	15	1,280	5,120
664.14M	18	1,536	6,144
1.41B	24	2,048	8,192
2.62B	30	2,560	10,240

Table 6: **Scaling of Parameters and Compute in the Enoki Model.** Training tokens follow Chinchilla scaling ($N = 20D$). Compute budget $C = 6NP$ follows Kaplan et al. (2020). PFdays = petaflop-days.

D (Total)	P (Non-embd)	N (Tokens)	C (PFdays)
45.71M	7.08M	0.914B	4.5×10^{-4}
70.39M	18.87M	1.41B	1.8×10^{-3}
140.97M	63.70M	2.82B	1.3×10^{-2}
186.48M	96.34M	3.73B	2.5×10^{-2}
254.02M	150.99M	5.08B	5.3×10^{-2}
423.69M	294.91M	8.47B	1.7×10^{-1}
664.14M	509.61M	13.28B	4.7×10^{-1}
1.41B	1.21B	28.28B	2.4
2.62B	2.36B	52.34B	8.6

where $V = 50304$ is the vocabulary size. The factor of 12 arises from the attention projections ($4 \times n_{\text{embd}}^2$ for Q, K, V, and output) plus the MLP projections ($8 \times n_{\text{embd}}^2$ for the up and down projections with $4\times$ expansion).

C.3.2 Initialization Details

Proper initialization is crucial for stable training of deep transformers, and the choice of initialization scheme interacts non-trivially with the learning rate schedule and optimizer hyperparameters.

The connection between initialization and learning rates. There is a symmetry between initialization, learning rate, and parameter multipliers (Yang and Hu, 2021) where the same training dynamics can be achieved with different but equivalent (initialization, learning rate) pairs.

Our initialization scheme. We use fan-in scaling for all linear layers, with residual projections (attention output and MLP down projection) incorporating an additional depth-dependent factor:

$$W \sim \mathcal{N}\left(0, \frac{1}{\text{fan_in}}\right) \tag{16}$$

$$W_{\text{proj}} \sim \mathcal{N}\left(0, \frac{1}{2 \times \text{fan_in} \times n_{\text{layer}}}\right) \tag{17}$$

This scheme ensures that the forward pass variance is approximately preserved regardless of layer width. The depth scaling on residual projections prevents the residual stream variance from growing unboundedly with depth, which is essential for stable training of deep transformers. We do not use parameter multipliers or per-layer learning rates.

Embedding initialization. Token embeddings are initialized separately from the transformer blocks, typically with standard deviation $1/\sqrt{n_{\text{embd}}}$:

$$E \sim \mathcal{N}\left(0, \frac{1}{n_{\text{embd}}}\right) \tag{18}$$

This scaling ensures that the norm of individual embedding vectors is approximately 1 in expectation, preventing the embedding layer from dominating or being dominated by subsequent layers.

Initialization in related work. The OLMo model family (Groeneveld et al., 2024) uses a similar fan-in initialization with depth scaling on residual projections. Qwen3 (Yang et al., 2025) follows similar principles while incorporating adjustments for the SwiGLU activation. Compared to the parameterization strategies in Everett et al. (2024), our parameterization is most similar to their standard parameterization with global learning rate. Their standard parameterization implementation uses a small constant of 0.01 for the embedding initialization standard deviation, which is similar to our choice of $1/\sqrt{n_{\text{embd}}} = 1/\sqrt{50304} \approx 0.0045$.

D Algorithms Used in Baselining & Referenced

To be explicit about the algorithms used in our baselining experiments and referenced throughout the text, we include descriptions below. We specify for each algorithm hyperparamters which are specific to that algorithm (e.g., for AdamW, β_1, β_2) and hyperparameters that are common across all algorithms (e.g., learning rate γ).

D.1 AdamW

The main algorithm we used for comparison in our benchmarking is AdamW Loshchilov and Hutter (2019) with fixed weight-decay and fixed 1st-/2nd-moment parameters β_1 and β_2 respectively.

Algorithm 2 AdamW (Adam Kingma and Ba (2015) with independent weight-decay)

Require: Initial parameters θ_0, 1st/2nd moments $m_0 = 0$, $v_0 = 0$, resp., peak LR $\gamma^* > 0$, LR schedule $\gamma(t)$, stability constant $\epsilon > 0$, $\omega > 0$ (weight-decay), number of iterations T
Require: **(Specific hyperparameters to AdamW)** 1st/2nd moment $\beta_1, \beta_2 \in (0, 1)$
Define schedules:
  $\beta_1(t) \equiv \beta_1$, $\beta_2(t) \equiv \beta_2$ {Constant, typically $\beta_1 = 0.9$ and $\beta_2 = 0.99$}
  $\lambda(t) \equiv \omega$ {Independent weight decay constant}
for $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample minibatch, compute stochastic gradient g_{t+1}
  $m_{t+1} = \beta_1 \cdot m_t + (1 - \beta_1) \cdot g_{t+1}$ {1st moment}
  $v_{t+1} = \beta_2 \cdot v_t + (1 - \beta_2) \cdot g_{t+1}^2$ {2nd moment}
  $\theta_{t+1} = \theta_t - \gamma(t) \left(\gamma^* \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} + \lambda \cdot \theta_t \right)$

The Adam algorithm Kingma and Ba (2015) without weight-decay, sets $\lambda = 0$ in AdamW. We also note that we are using **independent weight-decay**, denote for clarity in this section as λ_{ind}. The standard implementations of AdamW in `PyTorch` Paszke et al. (2019) and `Optax` Babuschkin et al. (2020) use *coupled weight-decay*, denote by λ_{cwd}. Coupled weight-decay contains the learning rate, that is,

$$\text{(independent weight-decay)} = \text{(learning rate} \times \text{coupled weight-decay)}, \quad \lambda_{\text{ind}} = \gamma^* \times \lambda_{\text{cwd}}.$$

It was shown in Wortsman et al. (2023) that independent weight-decay reduces learning rate sensitivity when transferring small models to large models.

D.2 Log-AdamW

Log-AdamW Algorithm 3 applies the full logarithmic time change to AdamW—scheduling both the momentum parameters β_1, β_2 and the weight-decay λ—but *without* the Nesterov-type update that

adds an extra gradient term. This serves as an ablation to determine whether the benefits of ADANA come primarily from the log-time scheduling itself, or whether the Nesterov acceleration (which includes the additional g_{t+1} term in the update) is essential.

Algorithm 3 LOG-ADAMW (ADAMW with log-time scheduling, no Nesterov term)

Require: Initial parameters θ_0, 1st/2nd moments $m_0 = 0$, $v_0 = 0$, resp., peak LR $\gamma^* > 0$, LR schedule $\gamma(t)$, stability constant $\epsilon > 0$, $\omega > 0$ (weight-decay), number of iterations T
Require: **(Specific hyperparameters to LOG-ADAMW)** $\delta > 0$ (schedule parameter, typically $\delta = 8$)
Define schedules:
  $\beta_1(t) = 1 - \frac{\delta}{\delta+t}$, $\beta_2(t) = 1 - \frac{\delta}{\delta+t}$ {Log-time momentum}
  $\lambda(t) = \frac{\omega}{T/10+t}$ {Log-time decaying weight-decay}
for $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample minibatch, compute stochastic gradient g_{t+1}
  $m_{t+1} = \beta_1(t) \cdot m_t + (1 - \beta_1(t)) \cdot g_{t+1}$ {1st moment}
  $v_{t+1} = \beta_2(t) \cdot v_t + (1 - \beta_2(t)) \cdot g_{t+1}^2$ {2nd moment}
  $\theta_{t+1} = \theta_t - \gamma(t) \left(\gamma^* \frac{m_{t+1}}{\sqrt{v_{t+1}}+\epsilon} + \lambda(t) \cdot \theta_t \right)$ {No extra g_{t+1} term}

Compared to ADANA and DANA, the key omission is the Nesterov-type update structure. In ADANA (with $\lambda = \epsilon = 0$), the parameter update includes an additional gradient term: $\theta_{t+1} = \theta_t - \gamma(g_{t+1} + \alpha(t)m_{t+1})/\sqrt{v_{t+1}}$, where the g_{t+1} provides immediate responsiveness while the momentum m_{t+1} captures long-horizon information. LOG-ADAMW uses only the momentum m_{t+1}, testing whether log-time scheduling alone—without this acceleration structure—is sufficient for improved optimization on language modeling tasks.

D.3 LOG-NADAMW

We define LOG-NADAMW using LOG-ADAMW on which we add a Nesterov-type update that adds an extra gradient term. We additionally add a momentum schedule $\alpha(t) = \frac{\delta+t}{\delta}$ on the momentum update. LOG-NADAMW serves as an ablation to show that a damping schedule o $\alpha(t)$ is necessary for stability. Indeed, LOG-NADAMW is unstable in the stochastic setting as shown on Figure 3.

Algorithm 4 LOG-NADAMW (ADAMW with log-time scheduling, additional gradient term and momentum scheduling)

Require: Initial parameters θ_0, 1st/2nd moments $m_0 = 0$, $v_0 = 0$, resp., peak LR $\gamma^* > 0$, LR schedule $\gamma(t)$, stability constant $\epsilon > 0$, $\omega > 0$ (weight-decay), number of iterations T
Require: **(Specific hyperparameters to LOG-NADAMW)** $\delta > 0$ (schedule parameter, typically $\delta = 8$)
Define schedules:
  $\beta_1(t) = 1 - \frac{\delta}{\delta+t}$, $\beta_2(t) = 1 - \frac{\delta}{\delta+t}$ {Log-time momentum}
  $\lambda(t) = \frac{\omega}{T/10+t}$ {Log-time decaying weight-decay}
  $\alpha(t) = \frac{\delta+t}{\delta}$ {Momentum learning rate}
for $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample minibatch, compute stochastic gradient g_{t+1}
  $m_{t+1} = \beta_1(t) \cdot m_t + (1 - \beta_1(t)) \cdot g_{t+1}$ {1st moment}
  $v_{t+1} = \beta_2(t) \cdot v_t + (1 - \beta_2(t)) \cdot g_{t+1}^2$ {2nd moment}
  $\theta_{t+1} = \theta_t - \gamma(t) \left(\gamma^* \frac{g_{t+1}+\alpha(t)m_{t+1}}{\sqrt{v_{t+1}}+\epsilon} + \lambda(t) \cdot \theta_t \right)$ {Combination of g_{t+1} and scheduled m_{t+1} term}

D.4 MUON

Background. MUON (MomentUm Orthogonalized by Newton-schulz) was originally proposed by Jordan et al. (2024) and demonstrated strong results training small-scale language models. The key insight is to replace standard gradient updates with orthogonalized updates: rather than moving in

the direction of the gradient, MUON approximately projects the momentum buffer onto the nearest orthogonal matrix. This constrains updates isometrically across all directions, fundamentally differing from ADAMW's element-wise adaptive approach.

For 2D weight matrices $W \in \mathbb{R}^{A\times B}$, MUON internally maintains SGD-style momentum (not ADAM's exponential moving average), then applies Newton-Schulz iterations to approximate the polar decomposition $M = U\Sigma V^\top \mapsto UV^\top$. This orthogonalization step can be efficiently computed in bfloat16 on GPU using a quintic iteration with carefully chosen coefficients.

Learning rate convention. A critical practical consideration is the learning rate scaling convention. The theoretical MUON update has RMS equal to $1/\sqrt{\max(A, B)}$ for full-rank matrices, where A and B are the matrix dimensions. To match the typical RMS of ADAMW updates (approximately 0.2–0.4), the Moonlight technical report (Liu et al., 2025) introduced a rescaling factor:

$$\gamma_{\text{adjusted}} = \gamma \cdot \sqrt{\max(A, B)} \cdot \texttt{matched_adamw_rms}. \tag{19}$$

This "matched AdamW RMS" convention ensures that MUON updates have comparable magnitude to ADAMW regardless of parameter shape, enabling transfer of learning rate schedules between the two optimizers.

Hybrid approach. For parameters that are not 2D matrices—including embeddings, layer norms, and biases—MUON falls back to ADAMW.

Algorithm 5 MUON (Jordan et al., 2024; Liu et al., 2025)

Require: Initial 2D parameters $W_0 \in \mathbb{R}^{A\times B}$, momentum buffer $M_0 = 0$
Require: Peal LR $\gamma^* > 0$, LR schedule $\gamma(t)$, weight-decay $\lambda > 0$, momentum $\mu \in (0, 1)$, number of iterations T
Require: **(Specific to MUON)** NS iterations K (typically 5), `matched_adamw_rms` ρ (typically 0.2–0.3), Nesterov flag
Newton-Schulz coefficients: $a = 3.4445$, $b = -4.7750$, $c = 2.0315$
for $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample minibatch, compute stochastic gradient $G_{t+1} \in \mathbb{R}^{A\times B}$
  $M_{t+1} = \mu \cdot M_t + G_{t+1}$ {SGD-style momentum}
  if Nesterov **then**
    $\tilde{M} = G_{t+1} + \mu \cdot M_{t+1}$
  else
    $\tilde{M} = M_{t+1}$
  // Newton-Schulz orthogonalization
  $X_0 = \tilde{M}/(\|\tilde{M}\|_F + \epsilon)$ {Normalize to spectral norm ≤ 1}
  if $A > B$ **then**
    $X_0 \leftarrow X_0^\top$ {Work with wider matrix}
  for $k = 0, 1, \ldots, K-1$ **do**
    $X_{k+1} = aX_k + (bX_kX_k^\top + c(X_kX_k^\top)^2)X_k$ {Quintic NS iteration}
  if $A > B$ **then**
    $O_{t+1} \leftarrow X_K^\top$
  else
    $O_{t+1} \leftarrow X_K$
  // Apply scaled update with weight-decay
  $\gamma_{\text{adj}} = \gamma(t)\gamma^* \cdot \rho \cdot \sqrt{\max(A, B)}$ {Matched AdamW RMS scaling}
  $W_{t+1} = (1 - \gamma(t)\lambda) \cdot W_t - \gamma_{\text{adj}} \cdot O_{t+1}$

The Newton-Schulz iteration approximates the matrix sign function, effectively computing $(MM^\top)^{-1/2}M \approx UV^\top$ where $M = U\Sigma V^\top$ is the SVD. The quintic coefficients $(a, b, c) = (3.4445, -4.7750, 2.0315)$ are optimized to maximize convergence rate near zero while maintaining stability.

D.5 ADEMAMIX

Background. ADEMAMIX (Pagliardini et al., 2025) extends ADAM by adding a second exponential moving average (EMA) of the gradients, with two very different timescales. The design inspiration is that "a single EMA cannot both give significant weight to recent gradients, and give non-negligible weight to older gradients."

Notation convention. In the original ADEMAMIX paper, $\beta_1 \approx 0.9$ denotes the fast (short-range) momentum and $\beta_3 \approx 0.9999$ (but usually swept) denotes the slow (long-range) momentum. **For consistency with our ADANA notation, we reverse these roles**: in our presentation, β_1 denotes the *long-range* momentum (large, close to 1) and β_3 denotes the *short-range* momentum (smaller, typically 0.9). This aligns with the DANA framework where $\beta_1(t) = 1 - \delta/t$ represents the logarithmic-time schedule for long-horizon memory.

Warmup schedules. Training with constant large momentum is unstable early in training. ADEMAMIX addresses this with warmup schedules: the long momentum $\beta_1(t)$ gradually increases from β_3 to its target value, and the mixing coefficient $\alpha(t)$ linearly warms up from 0 to its final value over T_α iterations. See Appendix J for detailed analysis of how these schedules relate to DANA's logarithmic-time schedule.

Algorithm 6 ADEMAMIX (Pagliardini et al., 2025) (notation: β_1 = long momentum, β_3 = short momentum)

Require: Initial parameters θ_0, fast/slow EMAs $m_1^{(0)} = m_3^{(0)} = 0$, second moment $v_0 = 0$
Require: Learning rate $\gamma > 0$, weight-decay $\lambda > 0$, stability constant $\epsilon > 0$, number of iterations T
Require: **(Specific to ADEMAMIX)** Short momentum $\beta_3 \approx 0.9$, long momentum $\beta_1 \approx 0.9999$, second moment $\beta_2 \approx 0.999$, mixing coefficient $\alpha > 0$ (typically 5–10), warmup horizon T_α

for $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample minibatch, compute stochastic gradient g_{t+1}
  // Warmup schedules (optional)
  $\hat{\beta}_1(t) = \texttt{warmup_scheduler}(\beta_3, \beta_1, t, T)$ {Gradually increase to β_1}
  $\hat{\alpha}(t) = \min(t/T_\alpha, 1) \cdot \alpha$ {Linear warmup of mixing coefficient}
  // Momentum updates
  $m_{t+1}^{(3)} = \beta_3 \cdot m_t^{(3)} + (1 - \beta_3) \cdot g_{t+1}$ {Fast/short-range EMA}
  $m_{t+1}^{(1)} = \hat{\beta}_1(t) \cdot m_t^{(1)} + (1 - \hat{\beta}_1(t)) \cdot g_{t+1}$ {Slow/long-range EMA}
  $v_{t+1} = \beta_2 \cdot v_t + (1 - \beta_2) \cdot g_{t+1}^2$ {Second moment}
  // Bias correction for fast EMA and second moment
  $\hat{m}_{t+1}^{(3)} = m_{t+1}^{(3)} / (1 - \beta_3^{t+1})$
  $\hat{v} = v_{t+1} / (1 - \beta_2^{t+1})$
  // Parameter update (no bias correction on slow EMA)
  $\theta_{t+1} = \theta_t - \gamma(t) \left(\gamma^* \frac{\hat{m}_{t+1}^{(3)} + \hat{\alpha}(t) \cdot m_{t+1}^{(1)}}{\sqrt{\hat{v}} + \epsilon} - \lambda \cdot \theta_t \right)$

Note that in its original formulation, Pagliardini et al. (2025) use a different weight-decay formulation

$$\theta_{t+1} = \theta_t - \gamma(t)\gamma^* \left(\frac{\hat{m}_{t+1}^{(3)} + \hat{\alpha}(t) \cdot m_{t+1}^{(1)}}{\sqrt{\hat{v}} + \epsilon} - \lambda \cdot \theta_t \right)$$

but for consistency with our framework we wrote Algorithm 6 with independent weight-decay (Loshchilov and Hutter, 2019; Wortsman et al., 2023).

The slow EMA $m^{(1)}$ does not use bias correction, as the warmup schedule handles the initial transient. The mixing coefficient α (typically 5–10) controls the relative contribution of old versus recent gradients. When $\alpha = 0$, ADEMAMIX reduces to standard ADAM.

D.6 Naive Implementation of Stochastic Nesterov's Accelerated Method (non-strongly convex setting)

Nesterov's accelerated method Nesterov (1983) is known to achieve optimal rates for minimizing non–strongly convex objective functions. A stochastic variant of Nesterov's method Nesterov (1988) is defined by the updates

$$\theta_{t+1} = y_t - \gamma g_t, \quad y_{t+1} = \theta_{t+1} + \mu_t(\theta_{t+1} - \theta_t), \quad \text{where } \mu_t = 1 - \frac{3}{t+3}, \; g_{t+1} = \nabla \mathcal{L}(y_t). \tag{20}$$

Here $\gamma > 0$ denotes the learning rate. For simplicity, we write $\mathcal{L}(y_t) \overset{\text{def}}{=} \mathcal{L}(y_t, x_{t+1})$ to emphasize that the gradient is stochastic. While Nesterov's accelerated method is originally formulated using full-batch (deterministic) gradients, here we replace the deterministic gradient with a stochastic one, yielding the most naive stochastic implementation of the method.

We note that Nesterov (1988); Beck and Teboulle (2009) use the parameter choice $\mu_t = 1 - \frac{3}{t+3}$, while later works Lan (2012); Flammarion and Bach (2015) observe that one may alternatively use $\mu_t = 1 - \frac{2}{t+1}$ with similar acceleration guarantees with full-batch gradients. However, Flammarion and Bach (2015) shows that this naive stochastic implementation diverges on power-law random features models (see Section B). This highlights the need to modify the algorithm to ensure stability in the presence of gradient noise (see Section L for further discussion).

In Section O.1, we show that the above update can be written approximately as an exponential moving average (EMA) of the momentum, yielding the equivalent representation

$$\begin{aligned} &\textit{(mom.)} \quad m_{t+1} = \left(1 - \frac{\delta}{\delta + t}\right) m_t + \frac{\delta}{\delta + t} \cdot g_{t+1}, \\ &\textit{(param.)} \quad \theta_{t+1} = \theta_t - \gamma \left(g_{t+1} + \frac{t+\delta}{\delta} m_{t+1}\right). \end{aligned} \tag{21}$$

We leave δ as a free hyperparameter, as it was shown in Ferbach et al. (2025) (and previously noted in Lan (2012); Flammarion and Bach (2015)) that multiple choices of δ yield comparable accelerated performance. We write the full version of stochastic Nesterov's accelerated method in Algorithm 7.

Algorithm 7 Stochastic Nesterov's Accelerated Method

Require: Initial parameters θ_0, 1st moment $m_0 = 0$, learning rate $\gamma > 0$, , number of iterations T
Require: **(Specific to Stochastic Nesterov)** δ a large constant.
Define schedules:
  $\beta_1(t) \equiv 1 - \frac{\delta}{\delta+t}$, {Log-time schedule}
  $\alpha(t) \equiv \frac{\delta+t}{\delta}$, {Momentum schedule}
for $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample minibatch, compute stochastic gradient g_{t+1}
  $m_{t+1} = \beta_1(t) \cdot m_t + (1 - \beta_1(t)) \cdot g_{t+1}$ {1st moment}
  $\theta_{t+1} = \theta_t - \gamma \left(g_{t+1} + \alpha(t) \cdot m_{t+1}\right)$

D.7 Dana Algorithm/Generalized Nesterov's accelerated method

To fix the divergence of the naive implementation of stochastic Nesterov's accelerated method, Ferbach et al. (2025) proposed a ***Generalized Nesterov's accelerated method*** (see also Defazio et al. (2024); Pagliardini et al. (2025); Defazio et al. (2025); Varre and Flammarion (2022); Flammarion and Bach (2015)) which updates

$$m_{t+1} = \beta_1(t) \cdot m_t + (1 - \beta_1(t)) \cdot g_{t+1} \quad \text{and} \quad \theta_{t+1} = \theta_t - \gamma(g_{t+1} + \boldsymbol{\alpha(t)} \cdot m_{t+1}) \tag{22}$$

$$\text{where } g_{t+1} = \nabla \mathcal{L}(\theta_t) \quad \text{and} \quad \beta_1(t) = 1 - \frac{\delta}{t+\delta}. \tag{23}$$

Here, $\boldsymbol{\alpha(t)}$ is a damping coefficient that must be tuned. When $\alpha(t) \asymp t$, we recover Stochastic Nesterov's accelerated method (Alg. 7). Choosing $\alpha(t) \gtrsim t$, amplifies the momentum term; further increasing the divergence problems from the noise of the stochastic gradients. When $\alpha(t) \lesssim t$, the

momentum term is dampened which helps with stability. At the same time, excessive damping causes the method to behave like standard SGD, forfeiting acceleration and scaling benefits. This raises the central question:

*Is there a choice of $\alpha(t)$ that achieves both **stability** and **acceleration** at scale?*

The DANA[8] algorithm in Ferbach et al. (2025) studied specific $\alpha(t)$ schedules and concluded that such damping schedules exist for the power-law random features (PLRF) model. In particular, the DANA algorithm studied damping schedules of the form:

$$\alpha(t) = \bar{\gamma}(d) \cdot (1+t)^{1-\kappa}, \tag{24}$$

where $\bar{\gamma}$ may depend on the parameter dimension d and $\kappa > 0$. The authors in Ferbach et al. (2025) noted two particular schedules

- DANA-DECAYING, $\alpha(t) \asymp (1+t)^{1-\kappa}$: For a specific choice of κ related to the spectral properties of the data, this schedule yielded the most acceleration over the class of schedules in (24) and its performance did not dimension as the model sized increased. The authors denoted generalized stochastic Nesterov accelerated method with this schedule as DANA-DECAYING (see also Yarotsky and Velikanov (2025) for related algorithm). When $\kappa \geq 1$, DANA-DECAYING did not accelerate over SGD on the PLRF.
- DANA-CONSTANT, $\alpha(t) \asymp \frac{t}{d}$, where d is the model size. Since model size d is ambiguous outside the PLRF on multi-layer transformers, it is possible to use $\alpha(t) \asymp \frac{t}{T^\kappa}$ where T is the training horizon. While this schedule achieved some amount of acceleration, it did not accelerate as favorably as DANA-DECAYING and can be unstable beyond the training horizon when using $\alpha(t) \asymp \frac{t}{T^\kappa}$. The authors denoted this algorithm as DANA-CONSTANT (see also Varre and Flammarion (2022) for related algorithm).

For more details, see Section L.

Choosing κ in the damping schedule: Practical guidance. On the PLRF model, the optimal value of κ depends on spectral properties of the data. Consequently, theory predicts that κ should remain unchanged as the problem dimension grows, making it a transferable hyperparameter across scales. In practice, the specific spectral properties of data are unknown and κ must be tuned. Empirically (see Figure 4 & Figure 5a), we find that κ is indeed transferable across model sizes and relatively easy to tune. In our experiments on transformer architectures trained on FineWeb data Penedo et al. (2024), effective values of κ lie in the range **0.75-0.9**, with **0.85** performing best across all scales considered in this work and the performance across this range of κ was quite similar. Moreover, κ appears to transfer well across architectures, performing well with minimal tuning when moving from Enoki models to Qwen.

Algorithm 8 DANA algorithm (Variants DANA-CONSTANT, DANA-DECAYING)

Require: Initial parameters θ_0, 1st moments $m_0 = 0$, learning rate $\gamma > 0$, number of iterations T
Require: **(Specific to DANA)** $\delta = 8$ a large constant, $\kappa > 0$ (spectral dimension), $\hat{\gamma} > 0$ damping constant factor.
Define schedules:
  $\beta_1(t) \equiv 1 - \frac{\delta}{\delta+t}$ {Equal log-time schedule}
  $\alpha(t) = \hat{\gamma} \times (1+t)^{1-\kappa}$ {DANA-DECAYING damping schedule}
  $\alpha(t) = \hat{\gamma} T^{-\kappa} \times (1+t)$ {DANA-CONSTANT damping schedule}
for $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample minibatch, compute stochastic gradient g_{t+1}
  $m_{t+1} = \beta_1(t) \cdot m_t + (1 - \beta_1(t)) \cdot g_{t+1}$ {1st moment}
  $\theta_{t+1} = \theta_t - \gamma \left(g_{t+1} + \alpha(t) \cdot m_{t+1} \right)$

[8]The DANA algorithm presented in Ferbach et al. (2025) is of slightly modified form than the one presented here. We show a near equivalence to the generalized Nesterov's accelerated method presented in this section in Section O.3.

Table 7: **Hyperparameters for ADANA variants.** Hyperpameters used for Qwen3 and Enoki scaling experiments (See Sec. C.3.1 for architectural details and Sec. F learning rate scaling schedules)

Parameter	ADANA	DANA-STAR	DANA-MK4	DANA-STAR-MK4
δ	8	8	8	8
κ	0.85	0.85	0.85	0.85
ω	4	4	4	4
clipSNR	-	-	2.0	2.0
τ_0	-	0	-	0

Algorithm 9 DANA-STAR

Require: Initial parameters θ_0, 1st/2nd moments $m_0 = 0$, $v_0 = 0$, probability initialization $\tau = 0$, peak LR $\gamma^* > 0$, LR schedule $\gamma(t)$, stability constant $\epsilon > 0$, number of iterations T

Require: **(Specific to DANA-STAR)** $\delta > 0$ (typically $\delta = 8$), $\kappa \in (0, 1)$ (spectral dimension), $\omega > 0$ (weight-decay), $\tau_0 = 0$ (probability estimator)

Define schedules:

$\beta_1(t) = \beta_2(t) = 1 - \frac{\delta}{\delta+t}$ {Logarithmic time}

$\lambda(t) = \frac{\omega}{\tau_{/10}+t}$ {Decaying weight-decay}

$\alpha(t) = (1+t)^{1-\kappa}$ {Damping schedule}

for $t = 0, 1, 2, \ldots, T-1$ **do**

Sample minibatch, compute stochastic gradient g_{t+1}

$m_{t+1} = \beta_1(t) \cdot m_t + (1 - \beta_1(t)) \cdot g_{t+1}$ {1st moment}

$v_{t+1} = \beta_2(t) \cdot v_t + (1 - \beta_2(t)) \cdot g_{t+1}^2$ {2nd moment}

$\tau_{\texttt{update}} = \frac{|g_{t+1}|}{|g_{t+1}|+\sqrt{v_{t+1}}+\epsilon}$

$\tau_{t+1} = (1 - \frac{\delta}{t+\delta})\tau_t + \frac{\delta}{t+\delta}\tau_{\texttt{update}}$ {Update class probabilities}

$t_{\texttt{eff}} = \max\{t \cdot \tau_{t+1}, 1\}$, $\tau_{\texttt{clip}} = \min\{\tau_{t+1}, 0.5\}$

$\tilde{\tau}_{t+1} = \max\{\tau_{\texttt{clip}}/(1 - \tau_{\texttt{clip}}), \frac{1}{1+t}\}$

$\theta_{t+1} = \theta_t - \gamma(t)\left(\gamma^* \frac{\sqrt{\tilde{\tau}_{t+1}} \odot g_{t+1}}{\sqrt{v_{t+1}}+\epsilon} + \gamma^* \alpha(t_{\texttt{eff}}) \cdot \frac{\sqrt{\tilde{\tau}_{t+1}} \odot m_{t+1}}{\sqrt{v_{t+1}}+\epsilon} + \lambda(t) \cdot \theta_t\right)$

D.8 ADANA VARIANTS

In this section, we described the exact algorithms for the ADANA variant, DANA-MK4, DANA-STAR, and DANA-STAR-MK4, described in Sec. 3 with full details in Sec. L and Sec. N. In Table 7, we provide the hyperparameters we typically used in the LLM experiments.

DANA-STAR. In Section 3 (with additional details in Section N), we build a variant of ADANA to handle sparse gradients. We denote this variant as DANA-STAR. We highlight in (red) the differences between DANA-STAR and ADANA in Algorithm 9. The main difference between ADANA and DANA-STAR is the addition of τ which attempts to estimate the probability of an update. While this does add an additional memory cost, it is not substantial.

DANA-MK4. In Section 3 (with additional details in Section N), we build a variant of ADANA to handle in homogeneous spectral dimensions, particularly making less sensitive to the choice of κ in the damping scheduling $\alpha(t) = (1+t)^{1-\kappa}$. We denote this variant as DANA-MK4. We highlight in (red) the differences between DANA-MK4 and ADANA in Algorithm 10. We note that $\text{sign}(\cdot)$ is applied entry-wise in Algorithm 10.

DANA-STAR-MK4. We combine both DANA-MK4 and DANA-STAR to create DANA-STAR-MK4 which is both less sensitive to choices of κ and less sensitive to sparse gradients. We give a complete description in Algorithm 11 and highlight any differences between ADANA in (red). The specific hyperparameters are the same as DANA-MK4 and DANA-STAR.

Algorithm 10 DANA-MK4 Algorithm

Require: Initial parameters θ_0, 1st/2nd moments $m_0 = 0$, $v_0 = 0$, peak LR $\gamma^* > 0$, LR schedule $\gamma(t)$, stability constant $\epsilon > 0$, number of iterations T

Require: **(Specific to DANA-MK4)** $\delta > 0$ (typically $\delta = 8$), $\kappa \in (0, 1)$ (spectral dimension), $\omega > 0$ (weight-decay), $\texttt{clipsnr} > 0$ (SNR clipping threshold)

Define schedules:

$\beta_1(t) = \beta_2(t) = 1 - \frac{\delta}{\delta + t}$ {Logarithmic time}

$\lambda(t) = \frac{\omega}{T/10 + t}$ {Decaying weight-decay}

for $t = 0, 1, 2, \ldots, T-1$ **do**

  Sample minibatch, compute stochastic gradient g_{t+1}

  $m_{t+1} = \beta_1(t) \cdot m_t + (1 - \beta_1(t)) \cdot g_{t+1}$ {1st moment update}

  $v_{t+1} = \beta_2(t) \cdot v_t + (1 - \beta_2(t)) \cdot g_{t+1}^2$ {2nd moment update}

  $\texttt{norm} = \frac{1}{\sqrt{v_{t+1}} + \epsilon}$, $\texttt{mfac} = |m_{t+1}| \cdot \texttt{norm}$

  $\alpha_{\texttt{fac}} = \min\{t^{1-\kappa} \cdot \texttt{mfac}, \texttt{clipsnr}\}$ {Clipped scaling}

  $\theta_{t+1} = \theta_t - \gamma(t) \left(\gamma^* g_{t+1} \odot \texttt{norm} + \gamma^* \text{sign}(m_{t+1}) \odot (\alpha_{\texttt{fac}} + |m_{t+1}| \odot \texttt{norm}) + \lambda(t) \cdot \theta_t\right)$

Algorithm 11 DANA-STAR-MK4 Algorithm

Require: Initial parameters θ_0, 1st/2nd moments $m_0 = 0$, $v_0 = 0$, probability initialization $\tau = 0$, peak LR $\gamma^* > 0$, LR schedule $\gamma(t)$, stability constant $\epsilon > 0$;

Require: **(Specific to DANA-STAR-MK4)** $\delta > 0$ (typically $\delta = 8$), $\kappa \in (0, 1)$ (spectral dimension), $\omega > 0$ (weight-decay), $\tau_0 = 0$ (probability estimator), $\texttt{clipsnr} > 0$ (SNR clipping threshold), number of iterations T

Define schedules:

$\beta_1(t) = \beta_2(t) = 1 - \frac{\delta}{\delta + t}$ {Logarithmic time}

$\lambda(t) = \frac{\omega}{T/10 + t}$ {Decaying weight-decay}

for $t = 0, 1, 2, \ldots, T-1$ **do**

  Sample minibatch, compute stochastic gradient g_{t+1}

  $m_{t+1} = \beta_1(t) \cdot m_t + (1 - \beta_1(t)) \cdot g_{t+1}$ {1st moment update}

  $v_{t+1} = \beta_2(t) \cdot v_t + (1 - \beta_2(t)) \cdot g_{t+1}^2$ {2nd moment update}

  $\tau_{\texttt{update}} = \frac{|g_{t+1}|}{|g_{t+1}| + \sqrt{v_{t+1}} + \epsilon}$

  $\tau_{t+1} = (1 - \frac{\delta}{t+\delta})\tau_t + \frac{\delta}{t+\delta}\tau_{\texttt{update}}$ {Update class probabilities}

  $t_{\texttt{eff}} = \max\{t \cdot \tau_{t+1}, 1\}$, $\tau_{\texttt{clip}} = \min\{\tau_{t+1}, 0.5\}$

  $\tilde{\tau}_{t+1} = \max\{\tau_{\texttt{clip}}/(1 - \tau_{\texttt{clip}}), \frac{1}{1+t}\}$

  $\texttt{norm} = \frac{\sqrt{\tilde{\tau}_{t+1}}}{\sqrt{v_{t+1}} + \epsilon}$, $\texttt{mfac} = \frac{|m_{t+1}| \cdot \texttt{norm}}{\tilde{\tau}_{t+1}}$

  $\alpha_{\texttt{fac}} = \min\{t_{\texttt{eff}}^{1-\kappa} \cdot \texttt{mfac}, \texttt{clipsnr}\}$ {Clipped scaling}

  $\theta_{t+1} = \theta_t - \gamma(t) \left(\gamma^* g_{t+1} \odot \texttt{norm} + \gamma^* \text{sign}(m_{t+1}) \odot (\tilde{\tau}_{t+1} \cdot \alpha_{\texttt{fac}} + |m_{t+1}| \odot \texttt{norm}) + \lambda(t) \cdot \theta_t\right)$

D.9 ABLATION ON ADANA AND COMPARISON WITH ADEMAMIX

We now describe in more details the different algorithms used in the ablation study in Figure 6 and extend this ablation in Figure 10. The general algorithm is shown in Algorithm 12 and uses an additional Exponential Moving Average (EMA) $\beta_3 \in (0, 1)$ on the gradient term $m_t^{(3)}$ compared to Algorithm 1). This EMA is introduced to study its impact on performance and compare ADANA with ADEMAMIX (DW) Algorithm 6 where such an EMA is used. In Table 8 we describe the different schedules and hyper-parameters used for each algorithm in Figure 10. All algorithms in this section use decaying weight-decay.

Algorithms Details. We summarize the specificity of each algorithm used in Figure 10 in Table 8. All algorithms use $\kappa = 0.75$ and log-time weight-decay. Note that algorithms using constant EMAs such as β_2 or β_3 constant have an additional bias correction update, as done in ADAMW, although we only explicitly wrote this bias correction for β_3 in Algorithm 12 for clarity. For example for constant

Algorithm 12 ADANA with short momentum EMA

Require: Initial parameters θ_0, 1st/2nd moments $m_0 = 0$, $v_0 = 0$, peak LR $\gamma^* > 0$, LR schedule $\gamma(t)$, stability constant $\epsilon > 0$, number of iterations T

Require: **(Specific to ADANA with short momentum EMA)** $\delta > 0$ (typically $\delta = 8$), $\kappa \in (0, 1)$ (spectral dimension), $\omega > 0$ (weight-decay), $\beta_3 \in (0, 1)$ (short momentum EMA)

Define schedules:

$\beta_1(t) = \beta_2(t) = 1 - \frac{\delta}{\delta+t}$ {Logarithmic time}

$\lambda(t) = \frac{\omega}{T/10+t}$ {Decaying weight-decay}

$\alpha(t) = (1+t)^{1-\kappa}$ {Damped Nesterov scaling}

for $t = 0, 1, 2, \ldots, T-1$ **do**

Sample minibatch, compute stochastic gradient g_{t+1}

$m^{(3)}_{t+1} = \beta_3 \cdot m^{(3)}_t + (1-\beta_3) \cdot g_{t+1}$ {Short 1st moment}

$m_{t+1} = \beta_1(t) \cdot m_t + (1-\beta_1(t)) \cdot g_{t+1}$ {1st moment}

$v_{t+1} = \beta_2(t) \cdot v_t + (1-\beta_2(t)) \cdot g^2_{t+1}$ {2nd moment}

$\hat{m}^{(3)}_{t+1} = \frac{m^{(3)}_{t+1}}{1-\beta_3^{t+1}}$ {Bias Correction}

$\theta_{t+1} = \theta_t - \gamma(t) \left(\gamma^* \frac{\hat{m}^{(3)}_{t+1} + \alpha(t) \cdot m_{t+1}}{\sqrt{v_{t+1}}+\epsilon} + \lambda(t) \cdot \theta_t \right)$

β_2, the update uses $\theta_{t+1} = \theta_t - \gamma(t) \left(\gamma^* \frac{\hat{m}^{3)}_{t+1} + \alpha(t) \cdot m_{t+1}}{\sqrt{\hat{v}_{t+1}}+\epsilon} + \lambda(t) \cdot \theta_t \right)$ where $\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}$. Note that for ADEMAMIX (DW) we set $\beta_1 = 1 - \frac{\delta}{T}$.

Table 8: **Optimizers used in the Ablation Study of ADANA.** ADANA variants use Algorithm 12 and ADEMAMIX uses Algorithm 6.

		Main Changes			Additional Modifications	
Variant	**Optimizer**	β_3	β_2	$\alpha(t)$	β_3 Bias Correction	β_2 Bias Correction
ADANA	Original ADANA	0.0	$\frac{\delta}{\delta+t}$	$(1+t)^{1-\kappa}$	\	\
ADANA 2	Dana-constant $\alpha(t)$	0.0	$\frac{\delta}{\delta+t}$	$T^{-\kappa} \times t$	\	\
ADANA 3	Constant β_2	0.0	0.999	$(1+t)^{1-\kappa}$	\	$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}$
ADANA 4	Dana-constant $\alpha(t)$ and Constant β_2	0.0	0.999	$T^{-\kappa} \times t$	\	$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}$
ADANA 5	Dana-constant $\alpha(t)$, Short EMA	0.9	$\frac{\delta}{\delta+t}$	$T^{-\kappa} \times t$	$\hat{m}^{(3)}_{t+1} = \frac{m^{(3)}_{t+1}}{1-\beta_3^{t+1}}$	\
ADANA 6	Dana-constant $\alpha(t)$, Short EMA and Constant β_2	0.9	0.999	$T^{-\kappa} \times t$	$\hat{m}^{(3)}_{t+1} = \frac{m^{(3)}_{t+1}}{1-\beta_3^{t+1}}$	$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}$
ADANA 7	Short EMA	0.9	$\frac{\delta}{\delta+t}$	$(1+t)^{1-\kappa}$	$\hat{m}^{(3)}_{t+1} = \frac{m^{(3)}_{t+1}}{1-\beta_3^{t+1}}$	\
ADEMAMIX (DW)	ADEMAMIX (DW)	0.9	0.999	$T^{-\kappa} \times t$	$\hat{m}^{(3)}_{t+1} = \frac{m^{(3)}_{t+1}}{1-\beta_3^{t+1}}$	$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}$

Ablation Results. A first observation is that ADEMAMIX (DW) significantly under-performs ADANA, especially at larger scales. To understand the cause of this performance gap, we ran ADANA with $\alpha(t) = T^{-\kappa} \times t$ schedule (ADANA 2), showing a strong decrease in performance. However, combining with a short-momentum EMA $\beta_3 = 0.9$ (ADANA 5) almost recovers ADEMAMIX (DW) performance. This proves that the DANA-CONSTANT-type schedule $\alpha(t) = T^{-\kappa} \times t$ appears to be less stable than our ADANA schedule $\alpha(t) = (1+t)^{1-\kappa}$. On the other hand a short momentum average (as is used in ADEMAMIX) can recover competitive performance and partly close the gap (while still showing decreased performance compared to ADANA). Similarly, adding a short-momentum EMA $\beta_3 = 0.9$ to original ADANA shows increased performance, especially at small scale, but remains negligible at larger scale. Note that it additionally increases memory by storing an additional momentum copy. Finally, constant β_2 has little effect at small scale but shows some instabilities at larger scales and especially diverges for one run at about 10^2 PFH of compute.

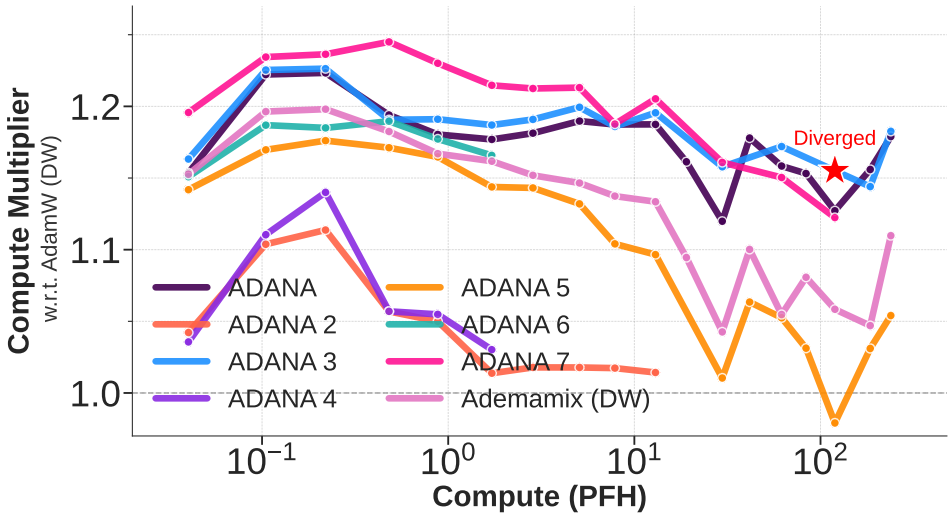
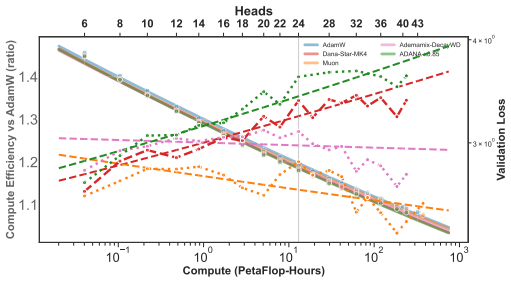
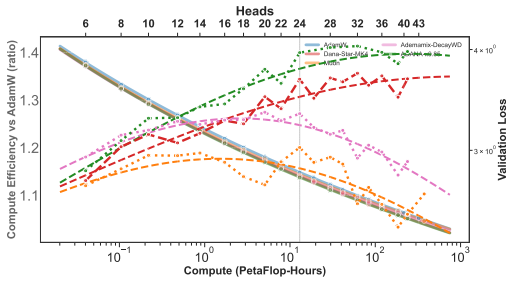


Figure 10: **Ablation Study.** Compute multipliers of variants of ADANA and ADEMAMIX (DW) relative to ADAMW (DW). All variants and the baseline ADAMW use log-time weight-decay and are summarized in Table 6 and Algorithm 12.



(a) Single power-law: $L = a + eC^{-f}$. Fitted $a = 1.045$; for AdamW: $e = 2.16$, $f = 0.074$ ($R^2 = 0.997$).



(b) Broken power-law: $L = a + bC^{-c} + eC^{-f}$. Fitted $a = 0.106$; for AdamW: $b = 0.506$, $c = 0.191$, $e = 2.58$, $f = 0.027$ ($R^2 = 0.9999$).

Figure 11: **Single vs broken power-law fits** using Chinchilla compute ($C = 6ND$). (a) The single power law forces a compromise between fitting small and large compute scales, yielding a saturation $a \approx 1.0$ much higher than the broken power law's $a \approx 0.11$. (b) The broken power law captures two distinct scaling regimes with different exponents ($c \approx 0.19$ vs $f \approx 0.03$), enabling accurate extrapolation.

E Scaling Laws

E.1 Single vs Broken Power-Law Fits

As shown in Figures 1 and 11, we fit loss scaling curves to the functional form

$$L(C) = a + b \cdot C^{-c} + e \cdot C^{-f}, \tag{25}$$

where C denotes compute in PetaFlop-Hours (PFH)[9], a is a shared saturation level across all optimizers, and (b, c, e, f) are optimizer-specific parameters. This "broken power-law" form extends the standard single power-law model $L(C) = a + b \cdot C^{-c}$ used in Chinchilla-style scaling analyses (Hoffmann et al., 2022).

The need for broken power laws has been identified in prior work. Caballero et al. (2023) demonstrate that neural network performance often exhibits multiple distinct power-law regimes with smooth transitions between them, rather than following a single power law. Similarly, Paquette et al. (2024) show that even simplified models of compute-optimal scaling (the PLRF) necessitates functional forms like (25). These findings motivate the use of a richer functional form that can capture transitions between scaling regimes.

E.2 Why Single Power Laws Are Insufficient

A standard single power law fit of the form $L(C) = a + b \cdot C^{-c}$ fails to capture the curvature present in our data at both small and large compute scales. Figure 11 compares single and broken power law fits for the same data. The single power law systematically underestimates loss at small scales and overestimates at large scales, leading to poor extrapolation properties.

A deeper issue with the single power law is that the exponent f is not uniquely determined by the data. Because a represents intrinsic model capacity—an unknown quantity that we assume all optimizers can achieve—different assumptions about a lead to different fitted exponents. Table 9 demonstrates this sensitivity: constraining a to small values (as implied by the broken power law) yields $f \approx 0.048$, while allowing a to grow large yields $f \approx 0.074$.

This sensitivity arises because the single power law conflates two distinct scaling regimes into a single exponent. When a is constrained to be small, the fit captures the gradual long-term regime ($f \approx 0.05$); when a is allowed to be large, the fit shifts toward the steep initial regime ($f \approx 0.08$).

[9] One PetaFlop-Hour equals 3.6×10^{18} floating-point operations ($10^{15} \times 3600$). Our experiments span approximately 10^{-2} to 2×10^2 PetaFlop-Hours. This unit is a convenient choice because a single H100 GPU delivers roughly one PetaFLOP of effective throughput for mixed-precision training, so PetaFlop-Hours correspond in order of magnitude to GPU-hours.

Table 9: **Single power-law exponent depends on assumed saturation (Chinchilla $6N$ compute).** Fitting $L = a + eC^{-f}$ to AdamW data with different upper bounds on the saturation parameter a. The exponent f varies by 54% (from 0.048 to 0.074) depending on the bounds for the assumed saturation level.

Upper bound on a	**Fitted a**	e	f
0.11	0.013	3.20	0.048
0.15	0.019	3.20	0.048
0.50	0.118	3.10	0.050
1.00	0.372	2.84	0.055
2.00	0.885	2.32	0.068
$0.95 \times L_{\min}$ (default)[10]	1.046	2.16	0.074

Table 10: **Fitted scaling law parameters (Chinchilla $6N$ compute).** All fits share a common saturation level a across optimizers. The single power law uses form $L = a + eC^{-f}$; the broken power law uses $L = a + bC^{-c} + eC^{-f}$ where the first term (b, c) captures the steep initial regime and the second term (e, f) captures the gradual long-term regime. For Kaplan $6P$ compute, see Table 11.

Optimizer	**Single Power Law** ($a = 1.042$)			**Broken Power Law** ($a = 0.106$)				
	e	f	R^2	b	c	e	f	R^2
AdamW	2.164	0.074	0.9971	0.405	0.211	2.68	0.030	0.99996
Dana-Star-MK4 (κ=0.85)	2.132	0.077	0.9970	0.404	0.217	2.64	0.030	0.99991
ADana (κ=0.85)	2.124	0.075	0.9963	0.404	0.217	2.64	0.030	0.99995
Muon	2.139	0.073	0.9960	0.404	0.217	2.65	0.029	0.99990
Ademamix (dec. WD)	2.129	0.074	0.9962	0.404	0.218	2.64	0.029	0.99995

The broken power law resolves this ambiguity by fitting both regimes explicitly, yielding $c \approx 0.21$ for the steep regime and $f \approx 0.03$ for the gradual regime, with $a \approx 0.106$ determined by the data rather than assumed. Table 10 summarizes the fitted parameters for all optimizers under both functional forms using the Chinchilla compute formula.

Several observations emerge from these fits. The steep-regime exponent c in the broken power law shows meaningful variation across optimizers (0.110–0.123), with Dana-Star-MK4 ($\kappa = 0.85$) having the smallest value and Muon the largest. In contrast, the gradual-regime parameters (e, f) are nearly identical across optimizers, indicating the efficiency gains do not appear to continue scaling.

E.3 Comparison of compute formulas

The relationship between compute budget C and model/data scale depends on how we measure the model's computational cost. In prior work, several conventions have been proposed (Kaplan et al., 2020; Hoffmann et al., 2022; Bi et al., 2024). We compare three formulations of FLOPs per token:[11]

$$6P = 72\, n_{\text{layer}}\, n_{\text{embd}}^2 \qquad \text{(Kaplan)} \tag{26}$$

$$6N = 72\, n_{\text{layer}}\, n_{\text{embd}}^2 + 6\, n_{\text{vocab}}\, n_{\text{embd}} \qquad \text{(Chinchilla)} \tag{27}$$

$$M = 72\, n_{\text{layer}}\, n_{\text{embd}}^2 + 12\, n_{\text{layer}}\, n_{\text{embd}}\, l_{\text{seq}} \qquad \text{(DeepSeek)} \tag{28}$$

where n_{layer} is the number of transformer layers, n_{embd} is the model width, n_{vocab} is the vocabulary size, and l_{seq} is the sequence length. Here P denotes the non-embedding parameter count and N the total parameter count (including embeddings).

The formula $6P$ counts only non-embedding matrix multiplications and corresponds to the approximation $C \approx 6PD$ used by Kaplan et al. (2020), which we use throughout this paper. The formula $6N$

[11]The notation $6N_1$, $6N_2$, and M follows the DeepSeek scaling analysis (Bi et al., 2024). We use $6P$ and $6N$ to emphasize the connection to model parameters.

Table 11: **Fitted scaling law parameters (Kaplan $6P$ compute).** Same format as Table 10 but using the Kaplan compute formula $6P$ which excludes embedding parameters. The single power-law exponents are smaller ($f \approx 0.064$–0.066) than with Chinchilla compute, reflecting the different compute scale.

Optimizer	Single Power Law ($a = 0.987$)			Broken Power Law ($a = 0.105$)				
	e	f	R^2	b	c	e	f	R^2
AdamW	2.146	0.065	0.9998	0.402	0.125	2.63	0.034	0.99977
Dana-Star-MK4 (κ=0.85)	2.114	0.066	0.9998	0.402	0.130	2.59	0.033	0.99975
ADana (κ=0.85)	2.107	0.066	0.9998	0.401	0.129	2.59	0.033	0.99976
Muon	2.124	0.064	0.9997	0.402	0.133	2.60	0.032	0.99970
Ademamix (dec. WD)	2.113	0.064	0.9998	0.402	0.132	2.59	0.032	0.99973

additionally includes the embedding and unembedding operations, corresponding to the Chinchilla (Hoffmann et al., 2022) approach. The DeepSeek formula M includes the attention computation overhead (the $12\, n_{\text{layer}}\, n_{\text{embd}}\, l_{\text{seq}}$ term accounts for the QK and attention-value products). Bi et al. (2024) report that the M formula produced better scaling law extrapolations for their models.

Table 12 compares single power law fits $L = a + eC^{-f}$ using each compute formula on AdamW training data. The choice of compute formula affects both the fitted saturation level a and the scaling exponent f: formulas that account for additional compute (embedding in $6N$, attention in M) yield larger effective compute values and hence different exponents.

Table 12: **Sensitivity of scaling exponent to compute formula.** Single power law fits $L = a + eC^{-f}$ for AdamW on Enoki architecture using different compute formulas. The $6P$ formula (non-embedding only) is our default. Including embeddings ($6N$) or attention overhead (M) yields different exponents. The right columns show fits with $a = 0$ (no saturation), which yields smaller exponents and worse fits. Fit residuals are shown in Figure 12.

Compute Formula	Fitted a				Forced $a = 0$		
	a	e	f	R^2	e	f	R^2
$6P$ (Kaplan, default)	0.977	2.16	0.065	0.9998	3.15	0.043	0.9988
$6N$ (Chinchilla)	1.046	2.16	0.074	0.9971	3.22	0.048	0.9928
M (DeepSeek)	1.010	2.17	0.069	0.9997	3.19	0.045	0.9979

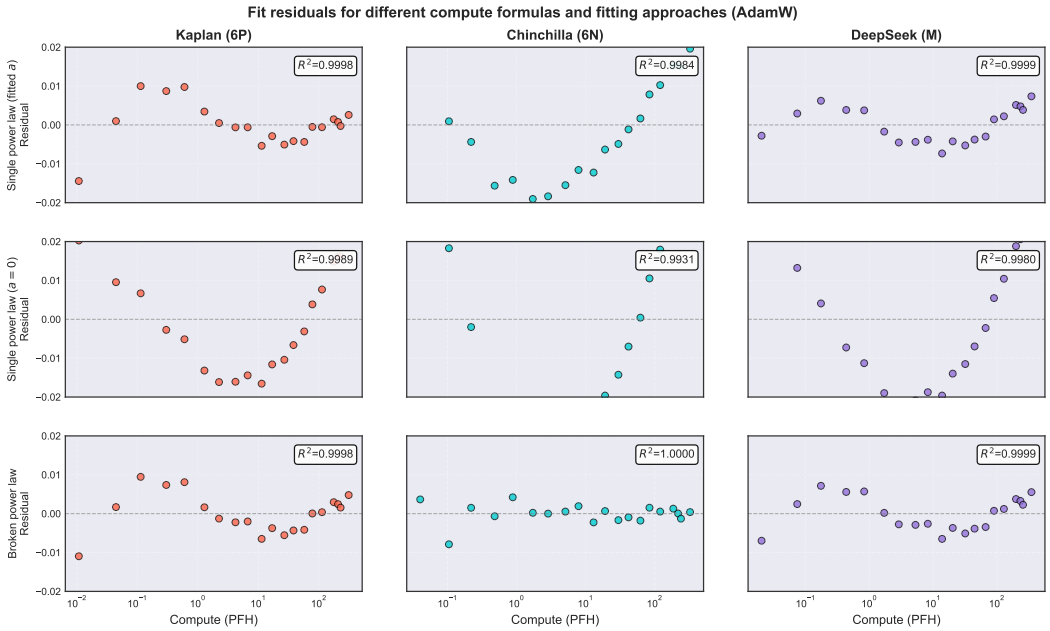


Figure 12: **Fit residuals for different compute formulas and fitting approaches (AdamW).** Columns: Kaplan ($6P$), Chinchilla ($6N$), and DeepSeek (M) compute formulas. Rows: single power law $L = a + eC^{-f}$ with fitted saturation (top), single power law with forced $a = 0$ (middle), broken power law $L = a + bC^{-c} + eC^{-f}$ (bottom). The forced $a = 0$ fits show systematic positive residuals at small compute scales. The Chinchilla formula with broken power law achieves the tightest fit ($R^2 = 0.9999$) by capturing both large-scale and small-scale behavior.

The variation in exponent f (from 0.065 to 0.074) across compute formulas illustrates a methodological concern: reported scaling exponents depend on conventions for measuring compute. The $6N$ formula yields the largest exponent because embedding operations contribute a larger fraction of compute at small scales than at large scales, creating an apparent steeper scaling. The DeepSeek M formula partially compensates by including attention overhead; as we do not scale the sequence length, this disappears with model scale.

The right columns of Table 12 show fits with $a = 0$ forced, corresponding to pure power laws $L = eC^{-f}$ without saturation. Forcing $a = 0$ systematically reduces the scaling exponent (from $f \approx 0.065$–0.074 to $f \approx 0.043$–0.048) and degrades fit quality. The residual plots in Figure 12 show that the forced $a = 0$ fits have systematic positive residuals at small compute scales, while the free a fits have more uniformly distributed residuals.

E.4 Does ADana Outscale AdamW?

A natural question is whether ADana (or other optimizers) can "outscale" AdamW in the sense of Ferbach et al. (2025): an algorithm *outscales* another if it achieves a larger loss exponent. On the power-law random features (PLRF) model, Dana provably outscales SGD when the spectral dimension satisfies $2\rho > 1$.

For the Chinchilla problem—compute-optimal training of transformers on natural language—this question is more subtle. The broken power law

$$L = a + bC^{-c} + eC^{-f} \tag{29}$$

captures two distinct scaling regimes: a small-scale regime dominated by bC^{-c} (steep decay) and a large-scale regime dominated by eC^{-f} (gradual decay). If the exponents c and f differ across optimizers, this would suggest different asymptotic scaling behavior.

Table 13 and Figure 13 show the fitted exponents for each optimizer using the $6N$ Chinchilla compute formula. Two observations emerge:

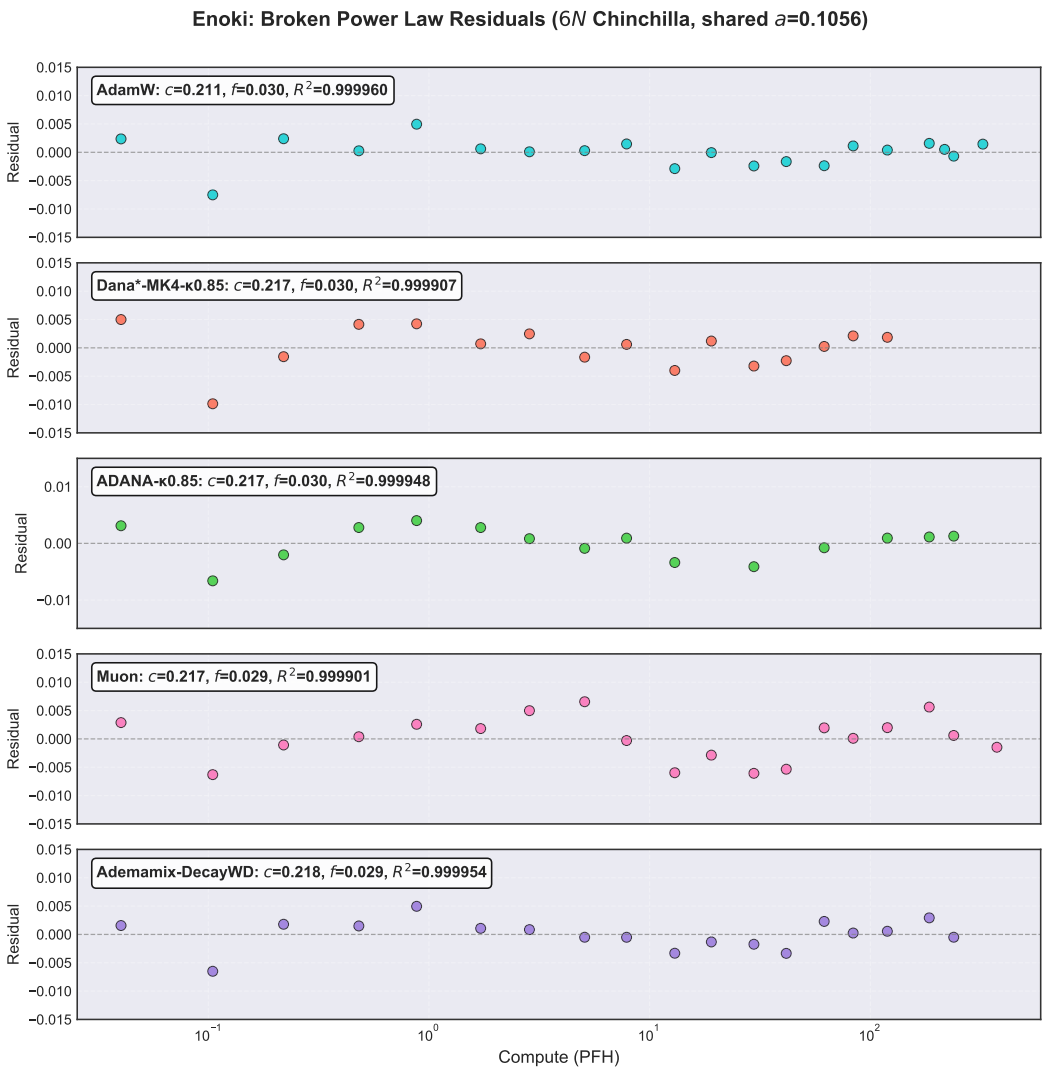


Figure 13: **Broken power law residuals across optimizers (6N Chinchilla, shared $a = 0.106$).** All optimizers achieve $R^2 \geq 0.9999$, indicating that the broken power-law functional form captures the scaling behavior consistently across optimization algorithms. The uniformly tight fits suggest that scaling law predictions can be reliably extrapolated from any optimizer.

Small-scale behavior. The exponent c is nearly identical across all optimizers ($c \approx 0.211$–0.218), with only a 3% variation between AdamW ($c = 0.211$) and the Dana variants ($c \approx 0.217$).

Large-scale behavior. The exponent f governing large-scale behavior is also nearly identical across all optimizers ($f \approx 0.029$–0.030). This indicates that the compute savings achieved by Dana variants persist at scale: the performance gap between optimizers appears as a consistent vertical shift in log-log space rather than a change in scaling exponent.

We emphasize that these changes are modest, and that further scaling experiments are needed to validate the scaling gap between the optimizers.

E.5 Measured exponents from the literature

To contextualize our measured scaling exponents, we survey the compute scaling law literature. Two distinct exponents appear in these discussions: the *loss scaling exponent* f describing how loss decreases with compute ($L \propto C^{-f}$), and the *allocation exponent* a describing how optimal model

Table 13: **Broken power-law exponents by optimizer.** Fits of (29) with shared saturation $a = 0.106$. The small-scale exponent c (steep decay) shows modest variation across optimizers, while the large-scale exponent f (gradual decay) is nearly identical ($f \approx 0.029$–0.030).

Optimizer	b	c	e	f	R^2
AdamW	0.405	0.211	2.68	0.030	0.99996
Dana-Star-MK4 (κ=0.85)	0.404	0.217	2.64	0.030	0.99991
ADana (κ=0.85)	0.404	0.217	2.64	0.030	0.99995
Muon	0.404	0.217	2.65	0.029	0.99990
Ademamix (dec. WD)	0.404	0.218	2.64	0.029	0.99995

Table 14: **Compute scaling law exponents from the literature.** Loss exponent f from fits of the form $L = eC^{-f}$ (pure power law) or $L = a + eC^{-f}$ (with irreducible loss). The "Formula" column indicates the compute calculation: $6P$ = Kaplan (non-embedding FLOPs), $6N$ = Chinchilla (includes embeddings), M = DeepSeek (includes attention overhead). Dagger (†) indicates exponent was extracted from figures rather than reported directly; question mark indicates formula unclear from paper. All optimizers are AdamW, except for Qiu et al. (2025a) which uses Muon and OpenAI which are unknown.

Paper	**Formula**	**Loss Exp. f**	**Irred.?**	**Max Scale (PfH)**	**Dataset**
Kaplan et al. (2020)	$6P$	0.050	No	∼550	WebText2
Brown et al. (2020) [GPT-3]	$6P$	0.048	No	∼87,000	Common Crawl
OpenAI (2023) [GPT-4]	?	0.067†	No	$> 10^6$	Internal
		0.120†	Yes		
Hoffmann et al. (2022)	$6N$	0.05	No	∼833	MassiveText
		0.154	Yes		
DeepSeek-AI (2024)	M	0.048†	No	∼125,000	In-house
Charles et al. (2025) [DiLoCo]	$6P$	0.048	No	∼33,000	C4, Dolma
Dey et al. (2025) [CompleteP]	?	0.051–0.056	No	∼278	SlimPajama
Gadre et al. (2025) [Gemstones]	$6N$	0.052†	Yes	∼5,600	Dolma 1.7
Porian et al. (2024)	$6P$	0.050†	No	∼28	OpenWebText2
Qiu et al. (2025) [Muon μP]	$6N$	0.049–0.051†	No	∼47	FineWeb
This work	$6P$	**0.043**	No	∼370	FineWeb
		0.065	Yes		
	$6N$	**0.048**	No		
		0.074	Yes		
	M	**0.045**	No		
		0.069	Yes		

size scales with compute ($N^* \propto C^a$). We focus primarily on the loss scaling exponent, which is directly comparable to our measurements.

Table 14 summarizes scaling law exponents across the literature. A striking pattern emerges: nearly all studies report $f \approx 0.05$ when fitting pure power laws without an irreducible loss term. The apparent discrepancy with Chinchilla's $f \approx 0.15$ arises because their exponent applies to the reducible portion $L - E$, not the total loss.

Kaplan et al. (2020) (Kaplan et al., 2020). The foundational scaling laws paper fit $L(C) = (C_c/C)^{f_C}$ with $f_C = 0.050$ and $C_c = 3.1 \times 10^8$ PF-days. They advocated training larger models on less data than subsequent work would recommend, finding optimal allocation $N^* \propto C^{0.73}$.

Brown et al. (2020) [GPT-3] (Brown et al., 2020). Validated Kaplan scaling at unprecedented scale, fitting $L(C) = 2.57 \cdot C^{-0.048}$ with C in PF-days. GPT-3 175B required approximately 87,000 PfH of compute.

OpenAI (2023) [GPT-4] (OpenAI, 2023). The GPT-4 technical report demonstrates that scaling laws enable accurate capability prediction: they fit $L(C) = aC^{-b} + c$ (with irreducible loss term c) using models trained with at most $1/10{,}000$th the compute of GPT-4, and accurately predicted GPT-4's final loss on an internal codebase. Although the specific parameters were not disclosed, we extracted data from their Figure 1 using automatic circle detection and fit scaling laws. A pure power law yields $f \approx 0.067$ ($R^2 = 0.96$), while including an irreducible loss term $E \approx 0.92$ bits/word yields $f \approx 0.120$ ($R^2 = 0.9999$). The latter is consistent with Chinchilla's exponent on reducible loss, while the former is substantially larger than other claimed data in the literature. The compute range spans approximately 10 orders of magnitude (from 10^{-10} to 1 relative to GPT-4).

Hoffmann et al. (2022) [Chinchilla] (Hoffmann et al., 2022). Introduced the two-term loss decomposition $L(N, D) = E + A/N^{\alpha} + B/D^{\beta}$ with irreducible loss $E = 1.69$. At compute-optimality, the reducible loss scales as $(L - E) \propto C^{-\gamma}$ with $\gamma = \alpha\beta/(\alpha + \beta) \approx 0.154$. Crucially, this exponent applies only to the reducible portion; fitting a pure power law to their data yields $f \approx 0.05$, consistent with other work. Chinchilla established the "20 tokens per parameter" guideline ($N^* \propto C^{0.5}$).

DeepSeek-AI (2024) (Bi et al., 2024). Proposed using non-embedding FLOPs per token (M) rather than parameter count (N) as the model scale metric, arguing this gives better extrapolation from small to large models. They find optimal allocation $M_{\text{opt}} \propto C^{0.524}$. The loss scaling exponent $f \approx 0.048$ was extracted from their Figure 5 by digitizing pixel coordinates and fitting a power law ($R^2 = 0.983$).

Charles et al. (2025) [DiLoCo] (Charles et al., 2025). Studied scaling laws for distributed training with local optimization. At Chinchilla-optimal allocation ($D = 20N$, hence $C = 120N^2$), their parameter scaling $L \propto N^{-0.095}$ converts to $L \propto C^{-0.048}$ on compute.

Dey et al. (2025) [CompleteP] (Dey et al., 2025). Compared standard μP initialization to their proposed CompleteP, finding $f = 0.051$ for μP and $f = 0.056$ for CompleteP—a modest improvement in scaling exponent from better initialization.

Gadre et al. (2025) [Gemstones] (McLeish et al., 2025). Systematically varied width and depth to study scaling in a 2D architecture space. A key finding is that the fitted allocation exponent is sensitive to methodology, ranging from $a = 0.46$ to $a = 0.80$ depending on fitting choices. The loss exponent $f \approx 0.052$ was extracted from their Figure 21 (convex hull points) and is consistent across compute scales ($R^2 = 0.9992$).

Porian et al. (2024) (Porian et al., 2024). Experimentally reconciled the Kaplan–Chinchilla discrepancy by identifying three factors: (1) FLOP counting conventions, (2) learning rate warmup scaling, and (3) optimizer hyperparameter tuning. Applying all corrections shifts the allocation exponent from Kaplan's $a \approx 0.88$ to Chinchilla's $a \approx 0.50$.

Qiu et al. (2025) [Muon μP] (Qiu et al., 2025a). Demonstrated that matrix-preconditioned optimizers (MUON, SHAMPOO) achieve consistent 1.3–1.4$\times$ compute speedups over ADAMW when using proper hyperparameter transfer via μP. The loss exponents $f \approx 0.049$–0.051 (extracted from their Table 5) are consistent across optimizers—the speedup manifests as a vertical shift in log-log space rather than a change in slope.

Comparison with this work. When fitting a single power law $L = a + eC^{-f}$ to our ADAMW data, we obtain $f = 0.044$ without saturation ($a = 0$) or $f = 0.065$ with fitted saturation ($a \approx 0.99$). Both values lie within the range of the literature. The sensitivity to the saturation parameter (Table 9) underscores the difficulty of comparing exponents across studies with different fitting methodologies. Our broken power law analysis reveals that a single exponent inadequately captures the loss-compute relationship: the small-scale exponent $c \approx 0.115$ governs initial rapid improvement, while the large-scale exponent $f \approx 0.032$ reflects slower asymptotic gains. This two-regime structure suggests that literature exponents $f \approx 0.05$ may represent an average over distinct scaling phases.

F BASELINING PROCEDURE

We describe our systematic hyperparameter search procedure for fair comparison across optimizers. A rigorous baselining methodology is essential when comparing optimizers, as performance differences can easily be confounded by suboptimal hyperparameter choices for one or more methods.

F.1 SEARCH STRATEGY

For each optimizer and model size, we perform a two-stage search over learning rate and weight-decay. The first stage uses a coarse grid to identify the region of good performance, followed by a finer search to locate the optimum.

Coarse learning rate search. We first search over learning rates on a log scale with factor-of-2 spacing:

$$\gamma^* \in \{\cdots, 2^{-14}, 2^{-13}, 2^{-12}, \cdots, 2^{-4}, 2^{-3}\}. \tag{30}$$

The upper and lower bounds of the search are truncated when the loss behavior begins to consistently grow or become unstable, and it contains a local minimum with at least 2 data points on either side.

Fine learning rate search. Once the coarse search identifies the best learning rate $\bar{\gamma}$, we perform a finer search using factors of 1.25:

$$\gamma^* \in \{\bar{\gamma}/1.25^2, \bar{\gamma}/1.25, \bar{\gamma}, \bar{\gamma} \times 1.25, \bar{\gamma} \times 1.25^2\}. \tag{31}$$

We continue this refinement until we have at least two data points on either side of the local minimum, ensuring robust identification of the optimal learning rate.

Weight-decay grid. For ADAMW, ADEMAMIX, and other fixed-weight-decay optimizers, we search:

$$\lambda \in \{0, 0.01, 0.03, 0.1, 0.3, 1.0\}. \tag{32}$$

For ADANA variants with decaying weight-decay, $\lambda(t) = \omega/(t + t_{\text{wd}})$ where $t_{\text{wd}} = T/10$, we search ω:

$$\omega \in \{1.0, 2.0, 4.0, 8.0\}. \tag{33}$$

In practice, we find $\omega = 4.0$ to be robust across model sizes and optimizers.

F.2 OTHER TRAINING DETAILS

Learning Rate Schedule and Batch. Given a peak learning rate γ^*, we use linear warmup for $T/50$ steps starting $0.01 \times \eta$ at the start of training followed by cosine decay (Loshchilov and Hutter, 2017) to $0.1 \times \gamma^*$ of the maximum learning rate at the end of training. The batch size is fixed at 32 sequences $\times$ 2048 tokens = 65,536 tokens per step.

Gradient Accumulation. We employ gradient accumulation over multiple microbatches to simulate larger effective batch sizes, following the implementation of Semenov et al. (2025). For each training iteration, we accumulate gradients over K microbatches (default $K = 1$) by dividing the loss by K before the backward pass, ensuring the accumulated gradient represents the average rather than the sum. The cross-entropy loss excludes padding tokens via `ignore_index`, with automatic normalization by the count of non-padding tokens only. The forward pass uses `bfloat16` precision, but model parameters, optimizer state, and accumulated gradients are maintained in `float32`. Because accumulation occurs in `float32`, no specialized numerical precision techniques (e.g., Kahan summation) are employed.

Weight-Decay. All algorithms use independent weight decay (Loshchilov and Hutter, 2019), and we do not apply weight decay to embeddings and layer-norm parameters. We use and compare two different types of weight-decay, either constant $\lambda(t) = \omega/T$ or log-time decaying $\lambda(t) = \omega/(T/10 + t)$ where ω is a hyperparameter. See Section 2.3 for more details.

Table 15: **Training hyperparameters.** Fixed values across all experiments unless otherwise noted.

Parameter	Value
Sequence length	2048
Batch size (sequences)	32
Tokens per batch	65,536
Vocabulary size	50,304
Warmup fraction	0.02 (2% of total steps)
Final LR fraction	0.10
Precision	bfloat16
Optimizer state precision	float32
Gradient clipping	0.5 (global norm)

Chinchilla-optimal training. Following the Chinchilla scaling laws (Hoffmann et al., 2022), we train each model for a number of steps proportional to its parameter count, ensuring approximately 20 tokens per parameter. Specifically, for a model with N total parameters:

$$\text{iterations} = \left\lfloor \frac{20 \times N}{65536} \right\rfloor \tag{34}$$

where 65,536 is the number of tokens per batch. This ensures that all models in our scaling study are trained in the compute-optimal regime.

Weight-Decay Exclusions The following parameter groups are excluded from weight-decay regularization:

- Token embeddings
- All LayerNorm/RMSNorm scale parameters
- All bias parameters

This follows standard practice (Loshchilov and Hutter, 2019) and prevents weight-decay from interfering with normalization dynamics. Applying weight-decay to normalization parameters can destabilize training by preventing the model from learning appropriate feature scales.

Compute Budget. All experiments with batch size 32 and context length 2048 up to 2.62B parameters can be run using 4 H100 of each 80G memory using Distributed Data Parallel and if necessary gradient accumulation. Per-model compute usage is described in Appendix F.2.

Table 16: **Per-model run compute.** GPU hours for AdamW per model size.

Model Size	GPU Hours	GPU Type
45.7M	0.4	A100
70.4M	0.9	A100
141.0M	3.7	A100
254.0M	14	A100
423.7M	17	H100
664.1M	41	H100
1.41B	173	H100
2.62B	626	H100

F.3 Learning Rate Scaling Laws

A key component of fair comparison across model scales is determining how hyperparameters—particularly the learning rate—should scale with model size. We fit power-law scaling rules to the

optimal learning rates found at each model size, enabling extrapolation to larger scales without exhaustive hyperparameter search.

F.3.1 FITTING METHODOLOGY

For each optimizer, we collect the top-K learning rates (ranked by final validation loss) at each model size. We use $K = 5$ to ensure robustness against noise in individual runs. Each learning rate receives a weight: the best learning rate receives weight K, the second-best receives weight $K - 1$, and so on.

We fit a saturated power law of the form:

$$\gamma^*(P) = a \cdot (b + P)^d, \tag{35}$$

where P denotes the number of non-embedding parameters, and a, b, d are fitted coefficients with constraints $a, b > 0$. The saturation term b accounts for the observation that optimal learning rates do not diverge as model size approaches zero; instead, they saturate to a finite value.

The fit is performed by minimizing a weighted mean-squared error loss in log-space:

$$\mathcal{L} = \frac{\sum_i w_i^2 \cdot P_i \cdot (\log \gamma_i^* - \log \hat{\gamma^*}(P_i))^2}{\sum_i w_i^2 \cdot P_i}, \tag{36}$$

where w_i is the weight assigned to each data point and the factor of P_i gives additional emphasis to larger models where extrapolation accuracy matters most. Here γ_i^* are the observed peak LR and $\hat{\gamma^*}$ are the predicted LR. We optimize using Adagrad for 200,000 steps, enforcing positivity constraints on a and b via exponential reparameterization.

F.3.2 ENOKI MODEL LEARNING RATE FITS

For the Enoki architecture (Section C.1), we fit learning rate scaling laws using data from model sizes with 6 to 24 attention heads, corresponding to 7M to 510M non-embedding parameters. These fits were used to select learning rates for larger-scale runs (up to 41 heads):

$$\text{ADAMW:} \quad \gamma^* = 1.28 \times 10^1 \cdot (1.67 \times 10^4 + P)^{-0.515}, \tag{37}$$

$$\text{DANA-MK4:} \quad \gamma^* = 5.60 \times 10^1 \cdot (5.71 \times 10^3 + P)^{-0.708}, \tag{38}$$

$$\text{ADEMAMIX:} \quad \gamma^* = 4.51 \cdot (6.71 \times 10^3 + P)^{-0.503}, \tag{39}$$

$$\text{MUON:} \quad \gamma^* = 2.19 \cdot (5.64 \times 10^4 + P)^{-0.417}. \tag{40}$$

Several patterns emerge from these fits. First, the exponent d varies significantly across optimizers, ranging from -0.42 for MUON to -0.71 for DANA-MK4. This indicates that different optimizers have fundamentally different scaling behavior: DANA variants require more aggressive learning rate reduction at large scales, while MUON maintains relatively high learning rates. Second, the saturation parameter b is typically on the order of 10^3 to 10^4, indicating that the power-law regime dominates for models larger than approximately 10M parameters.

F.3.3 ADANA, DANA-MK4 AND DANA-STAR-MK4 LEARNING RATE FITS BY κ

For ADANA and DANA-STAR-MK4, the spectral dimension parameter κ affects the optimal learning rate scaling. We fit separate scaling laws for different κ values to understand this dependence. The results are shown in Figures 15 and 16.

For DANA-STAR-MK4:

$$\kappa = 0.75: \quad \gamma^* = 4.10 \times 10^1 \cdot (3.03 \times 10^4 + P)^{-0.661}, \tag{41}$$

$$\kappa = 0.85: \quad \gamma^* = 2.23 \times 10^2 \cdot (2.40 \times 10^6 + P)^{-0.724}. \tag{42}$$

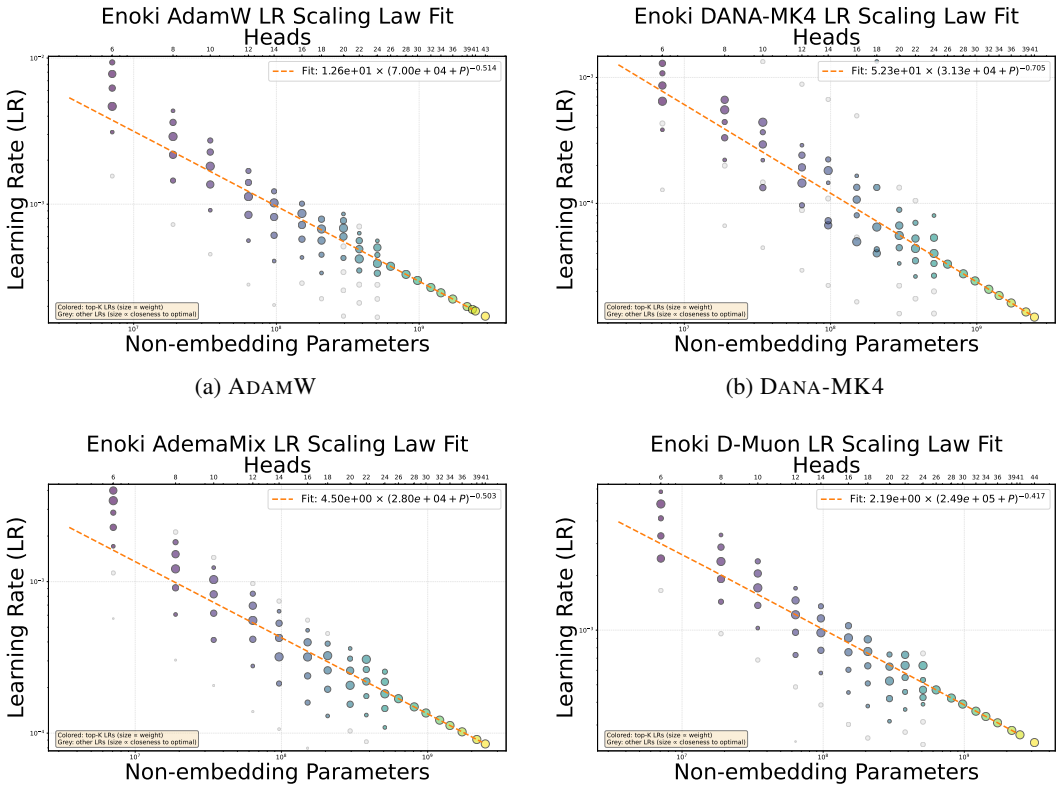
For ADANA:

$$\kappa = 0.75: \quad \gamma^* = 1.27 \times 10^1 \cdot (9.55 \times 10^3 + P)^{-0.624}, \tag{43}$$

$$\kappa = 0.80: \quad \gamma^* = 1.76 \times 10^1 \cdot (3.66 \times 10^4 + P)^{-0.624}, \tag{44}$$

$$\kappa = 0.85: \quad \gamma^* = 2.25 \times 10^1 \cdot (1.22 \times 10^5 + P)^{-0.618}, \tag{45}$$

$$\kappa = 0.9: \quad \gamma^* = 6.89 \times 10^1 \cdot (2.95 \times 10^4 + P)^{-0.667}. \tag{46}$$



(a) ADAMW (b) DANA-MK4

(c) ADEMAMIX (d) MUON

Figure 14: **Peak learning rate γ^* scaling for Enoki models.** Each panel shows the optimal learning rates (circles, sized by weight) and the fitted power-law curve (dashed line) for a single optimizer. The fits shown here include data from all scales (6–41 heads) and may differ slightly from the equations above, which were fit using only data up to 24 heads and used to extrapolate learning rates for larger runs.

For DANA-MK4:

$$\kappa = 0.85: \quad \gamma^* = 2.30 \times 10^1 \cdot (1.46 \times 10^4 + P)^{-0.627}. \tag{47}$$

(48)

The higher κ value leads to a larger saturation parameter b, indicating that the power-law regime begins at larger model sizes. The exponents are similar between the two κ values for each optimizer family, suggesting that κ primarily affects the learning rate magnitude rather than the scaling behavior.

F.3.4 ADAMW AND ADEMAMIX WITH DECAYING WEIGHT-DECAY

With decaying logarithmic-time weight-decay, ADAMW (DW) and ADEMAMIX (DW) have the following LR fits:

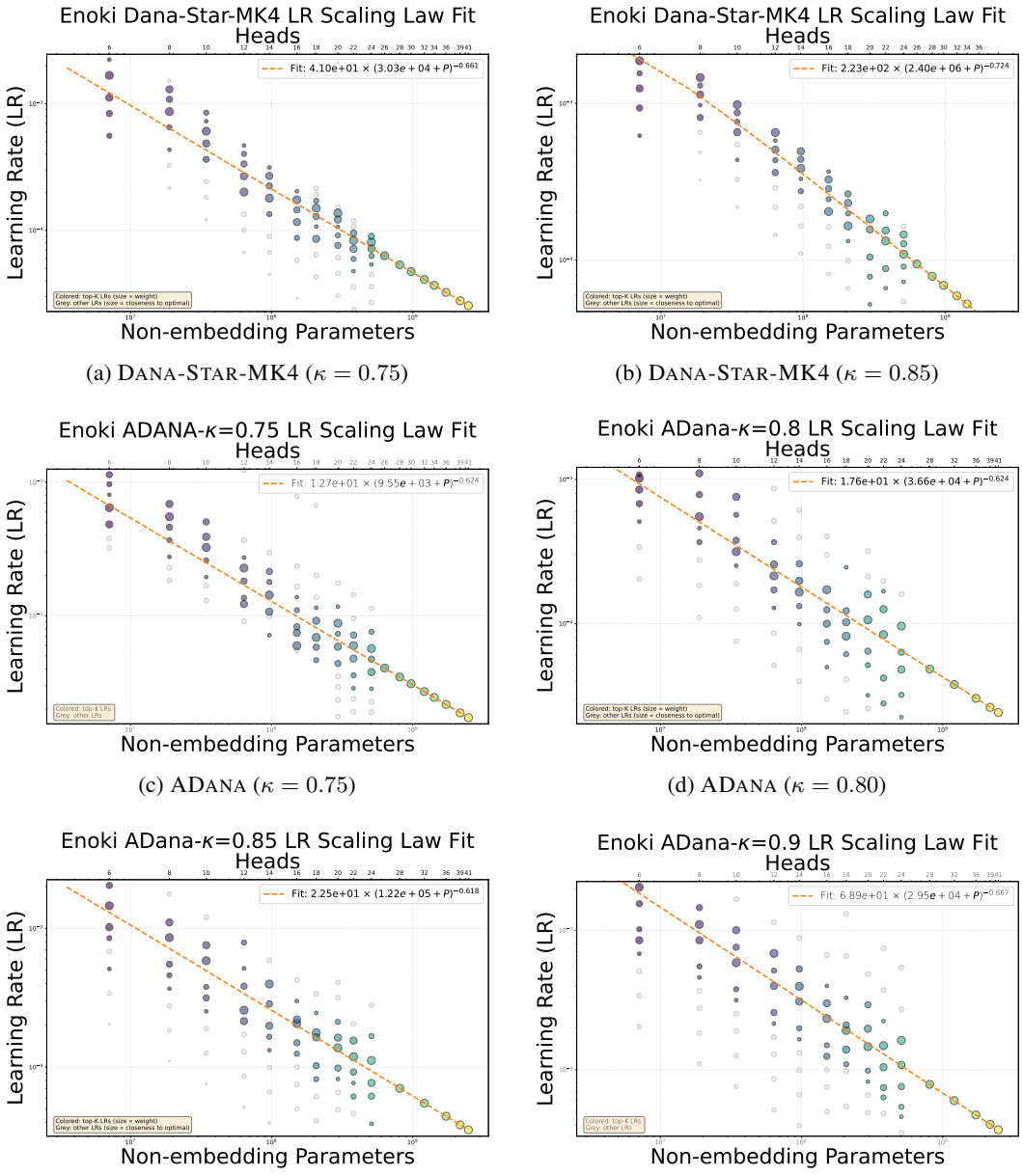
$$\text{ADAMW (DW)}: \quad \gamma^* = 6.57 \times 10^1 \cdot (9.83 \times 10^3 + P)^{-0.608}, \tag{49}$$

$$\text{ADEMAMIX (DW)}: \quad \gamma^* = 7.28 \cdot (8.43 \times 10^3 + P)^{-0.525}. \tag{50}$$

The peak learning rate fits are shown in Figure 17.

F.3.5 ADANA VARIANTS USED IN THE ABLATION STUDY IN APPENDIX D.9

Below we report the LR fits of scaled ADANA variants used in the ablation Appendix D.9 and reported in Table 8.



(a) DANA-STAR-MK4 ($\kappa = 0.75$) (b) DANA-STAR-MK4 ($\kappa = 0.85$)

(c) ADANA ($\kappa = 0.75$) (d) ADANA ($\kappa = 0.80$)

(e) ADANA ($\kappa = 0.85$) (f) ADANA ($\kappa = 0.9$)

Figure 15: **Peak learning rate γ^* scaling for ADANA and DANA-STAR-MK4 by κ.** Top row: DANA-STAR-MK4 with different κ values. Middle and bottom rows: ADANA (without τ estimator) with different κ values. Colored circles indicate top-K learning rates used in the fit; grey circles show other explored learning rates with size proportional to closeness to optimal.

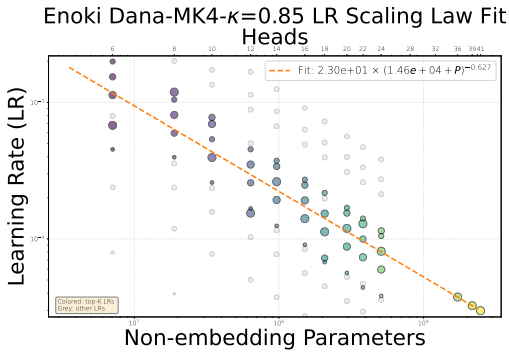
$$\text{ADANA 3:} \quad \gamma^* = 2.38 \times 10^1 \cdot (8.13 \times 10^3 + P)^{-0.657}, \tag{51}$$

$$\text{ADANA 5:} \quad \gamma^* = 4.57 \times 10^1 \cdot (1.47 \times 10^4 + P)^{-0.627}, \tag{52}$$

$$\text{ADANA 7:} \quad \gamma^* = 2.58 \cdot (6.04 \times 10^3 + P)^{-0.533}. \tag{53}$$

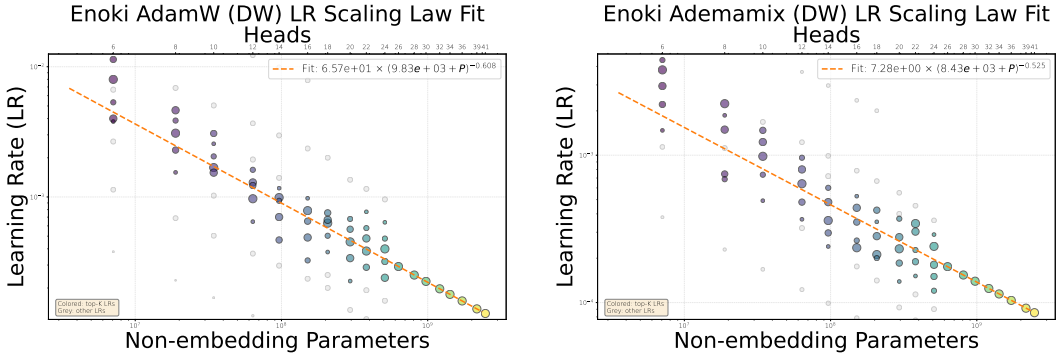
F.3.6 EXTRAPOLATION AND SENSITIVITY ANALYSIS

The learning rate scaling laws enable principled extrapolation to model sizes beyond those directly tuned. For the Enoki architecture at 2.4B parameters (40 heads), the predicted learning rates are:



(a) DANA-MK4 ($\kappa = 0.85$)

Figure 16: Peak learning rate γ^* scaling for DANA-MK4, $\kappa = 0.85$.



(a) ADAMW (Decaying Weight-decay)

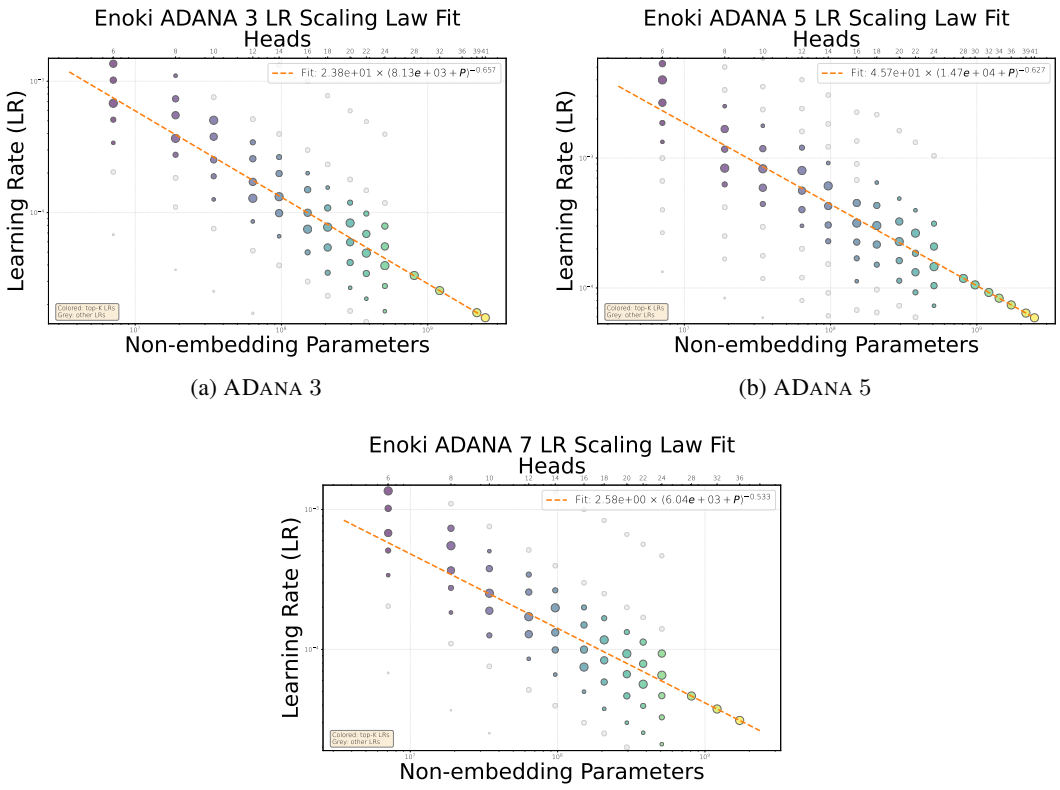
(b) ADEMAMIX (Decaying Weight-decay)

Figure 17: Peak learning rate γ^* scaling on Enoki models for ADAMW and ADEMAMIX using logarithmic-time weight-decay.

Optimizer	**Predicted LR γ^***
ADAMW	1.91×10^{-4}
ADEMAMIX	8.70×10^{-5}
DANA-STAR-MK4	2.63×10^{-5}
MUON	2.70×10^{-4}
DANA-MK4	1.30×10^{-5}

While these point estimates allow us to run large-scale experiments with minimal hyperparameter search, practitioners need to understand the uncertainty in these predictions and their downstream impact on compute savings claims. We present a three-stage sensitivity analysis: (1) bootstrap confidence intervals on the fitted parameters, (2) loss curvature analysis, and (3) propagation of uncertainties to compute savings estimates.

Bootstrap Confidence Intervals. We quantify uncertainty in the fitted LR scaling parameters using model-size level bootstrap resampling. For each of $N = 100$ bootstrap samples, we resample the set of M model sizes with replacement, drawing exactly M samples (e.g., if we have data at heads $\{6, 8, 10, 12, 14, 16, 18, 20, 22, 24\}$, we draw 10 sizes with replacement, so a bootstrap sample might include $\{6, 8, 8, 12, 14, 14, 14, 20, 22, 24\}$ with some sizes appearing multiple times and others omitted). For each sampled size, we retain all of its top-K LRs with their associated weights. We then refit the power law $\gamma^*(P) = a \cdot (b + P)^d$ using weighted MSE, where weights are the product of the top-K rank weight (best LR gets weight K, second-best gets $K - 1$, etc.) squared and the parameter count, giving more emphasis to larger models.



(a) ADANA 3

(b) ADANA 5

(c) ADANA 7

Figure 18: Peak learning rate γ^* scaling for optimizers used in the ablation study Table 8 on Enoki models. We report only optimizers that were scaled beyond 24 heads.

This yields distributions over the fitted parameters (a, b, d) and, critically, over the predicted learning rates at extrapolation scales. The 95% confidence intervals for predicted learning rates are quite tight, even at 2.4B parameters (40 heads):

Optimizer	**Predicted LR**	**95% CI**	**Relative Width**
ADAMW	1.91×10^{-4}	$[1.89, 1.92] \times 10^{-4}$	$\pm 0.7\%$
DANA-MK4	1.30×10^{-5}	$[1.28, 1.32] \times 10^{-5}$	$\pm 1.5\%$

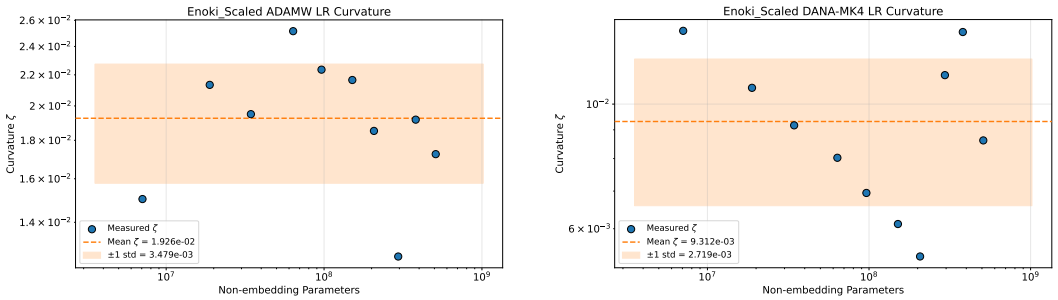
The narrow confidence intervals indicate that the power-law fits are highly constrained by the available data. The exponent d is particularly well-determined: for ADAMW, $d = -0.514$ with 95% CI $[-0.551, -0.495]$; for DANA-MK4, $d = -0.704$ with 95% CI $[-0.724, -0.684]$.

Loss Curvature. Learning rate errors translate to loss increases via the curvature of the loss landscape around the optimal LR. We characterize this by fitting a parabola in log-LR space at each model size:

$$L(\log \gamma^*) \approx L^*(P) + \zeta(P) \cdot (\log \gamma^* - \log \bar{\gamma}(P))^2, \tag{54}$$

where $\zeta(P)$ is the local curvature, γ^* is the observed peak learning rate, and $\bar{\gamma}(P)$ is the optimal predicted peak learning rate. We extract ζ at each model size by fitting to all LR sweep data (not just the top-K).

Figure 19 shows the measured curvatures across model sizes. Due to systematic variations that do not follow a simple power law (visible in the scatter), we report the average curvature rather than fitting a scaling law:



(a) ADAMW (b) DANA-MK4

Figure 19: **LR curvature across model sizes.** Local curvature ζ (measuring sensitivity to LR errors) as a function of model size. Points show measured curvatures from parabolic fits to LR sweep data; the dashed horizontal line shows the mean curvature with shaded ± 1 standard deviation band. The scatter does not follow a simple power law, so we use the average curvature for sensitivity calculations.

Optimizer	$\bar{\zeta}$	**Std Dev**
ADAMW	1.93×10^{-2}	3.5×10^{-3}
DANA-MK4	9.3×10^{-3}	2.7×10^{-3}

These curvatures are modest, meaning the loss landscape around the optimal LR is relatively flat. For a relative LR error ε, the expected loss increase is $\Delta L = \bar{\zeta} \cdot (\log(1+\varepsilon))^2$. Even a 50% learning rate error results in an expected loss increase of only:

$$\Delta L \approx \bar{\zeta} \cdot (\log 1.5)^2 \approx 0.019 \times 0.16 \approx 0.003 \text{ nats}, \tag{55}$$

which is negligible compared to typical loss differences between optimizers (~ 0.05–0.1 nats).

Propagation to Compute Savings Confidence Intervals. We translate loss uncertainty into compute savings uncertainty using the slope of the loss scaling law at the end of training. From $L = a + b \cdot C^{-c}$, the marginal compute cost of a loss improvement is:

$$\frac{dC}{dL} = -\frac{C}{c(L-a)}. \tag{56}$$

Thus, a loss increase ΔL (from LR error) corresponds to a relative compute increase of:

$$\frac{\Delta C}{C} = \frac{\Delta L}{|c|(L-a)}. \tag{57}$$

For the Enoki architecture at the 2.4B scale, with $L \approx 2.51$, $a \approx 0.20$, and $c \approx -0.043$ (from the loss scaling law fits in Section E), we have $|c|(L-a) \approx 0.10$. Combined with the bootstrap LR confidence intervals (± 1–2%) and the curvature $\bar{\zeta} \approx 0.015$, the expected loss increase from LR uncertainty is:

$$\Delta L \approx \bar{\zeta} \cdot (\log 1.02)^2 \approx 0.015 \times 0.0004 \approx 6 \times 10^{-6} \text{ nats}. \tag{58}$$

This translates to a relative compute uncertainty of:

$$\frac{\Delta C}{C} \approx \frac{6 \times 10^{-6}}{0.10} \approx 0.006\%. \tag{59}$$

This negligible uncertainty confirms that the compute savings claims in the main text are robust to LR extrapolation error. Even under pessimistic assumptions (50% LR error), the compute penalty would be approximately $0.003/0.10 \approx 3\%$—still far less than the compute savings achieved by the better optimizers.

G QWEN3 EXPERIMENTS

To validate that our optimizer findings generalize beyond the Enoki architecture, we conduct experiments on Qwen3, a modern SwiGLU-based transformer with different design choices.

G.1 QWEN3 MODEL ARCHITECTURE

The Qwen3 model features SwiGLU activation in the MLP (Shazeer, 2020), RMSNorm instead of LayerNorm (Zhang and Sennrich, 2019), elementwise attention output gating (Qiu et al., 2025b), and RoPE positional embeddings. The head dimension is fixed at 128 following the Qwen3 architecture.

The Qwen3 scaling rule (# of heads = heads):

$$\text{n_layer} = 2 \times \text{heads}, \quad \text{n_embd} = \text{heads} \times 128. \tag{60}$$

Our implementation differs from the official Qwen3 architecture in several respects: we do not employ Grouped Query Attention (GQA), as our goal is to test optimizer properties rather than parameter efficiency; we do not tie embedding weights (larger Qwen3 models also do not tie them); and we use 2048-token context length rather than the extended contexts typical of Qwen models (at least 32K tokens).

SwiGLU activation. The MLP uses the SwiGLU (Swish-Gated Linear Unit) activation function (Shazeer, 2020):

$$\text{MLP}(x) = W_{\text{down}} \left(\text{SiLU}(xW_{\text{gate}}) \odot xW_{\text{up}}\right) \tag{61}$$

with hidden dimension $3 \times n_{\text{embd}}$ to maintain comparable parameter counts. SwiGLU combines the Swish activation with a gating mechanism, using three linear projections instead of two. This architecture has become standard in modern open-weight LLMs: LLaMA 2+ (Touvron et al., 2023a;b), Mistral (Jiang et al., 2023), OLMo (Groeneveld et al., 2024), and Qwen (Yang et al., 2025) all employ SwiGLU in their feed-forward layers. The gating mechanism provides greater expressiveness and improved capacity to model complex relationships compared to standard ReLU or GELU activations.

Elementwise attention output gating. Following Qiu et al. (2025b), a sigmoid gate is applied to the attention output:

$$\text{Attn}_{\text{gated}}(X) = \text{Attn}(X) \odot \sigma(XW_g) \tag{62}$$

where $W_g \in \mathbb{R}^{n_{\text{embd}} \times (n_{\text{head}} \cdot d_h)}$ are learned parameters. This mechanism provides three key benefits: (1) it introduces non-linearity between the value projection W_V and output projection W_O, which otherwise form a low-rank linear mapping that limits expressiveness; (2) the sigmoid activation produces sparse gating scores (mean ≈ 0.12), creating input-dependent (query-dependent) sparsity that filters irrelevant context; and (3) this sparsity eliminates the "attention sink" phenomenon where initial tokens disproportionately accumulate attention scores. Empirically, gated attention also improves training stability, enabling larger learning rates and better length generalization.

Parameter count formulas. For Qwen3 with elementwise gating:

$$\text{attn} = 5 \times n_{\text{embd}} \times (n_{\text{head}} \cdot d_h) + 2 \times d_h \tag{63}$$

$$\text{mlp} = 9 \times n_{\text{embd}}^2 \tag{64}$$

$$\text{per_layer} = \text{attn} + \text{mlp} + 2 \times n_{\text{embd}} \tag{65}$$

$$\text{total} = n_{\text{layer}} \times \text{per_layer} + n_{\text{embd}} + 2 \times n_{\text{embd}} \times V \tag{66}$$

The factor of 5 in the attention term accounts for the doubled query projection (which includes the gate) plus key, value, and output projections. The MLP factor of 9 reflects the $3\times$ hidden dimension with three projection matrices (gate, up, down). The additional $2 \times d_h$ accounts for the QK-norm parameters, and the extra n_{embd} term accounts for the final RMSNorm layer.

G.2 COMPARISON WITH ENOKI SCALING

Different scaling rules produce architectures with substantially different depth-to-width ratios, which can affect both optimization dynamics and final model quality. Table 17 summarizes the scaling rules used in our experiments.

Table 17: **Comparison of scaling rules.** Each rule fixes certain aspect ratios while scaling others. The Qwen3 rule produces deeper models than Enoki for equivalent parameter counts.

Rule	n_{layer}	n_{embd}	d_h	**Depth/Width Ratio**
Enoki	$3h/4$	$64h$	64 (fixed)	3/256
Qwen3	$2h$	$128h$	128 (fixed)	1/64

Table 18: **Qwen3 results.** Final validation loss for different optimizers on Qwen3 models. Best loss per row in bold.

		ADANA		DANA-STAR-MK4		DANA-MK4	
Size	**ADAMW**	**κ0.85**	**ADEMAMIX**	**κ0.75**	**κ0.85**	**κ0.75**	**κ0.85**
176M	3.228	**3.182**	3.214	3.199	3.202	3.185	3.183
338M	3.017	**2.965**	2.999	2.980	2.977	2.971	**2.965**
588M	2.871	2.816	2.868	2.839	2.820	2.834	**2.815**
947M	2.740	2.696	2.730	2.709	2.699	2.707	**2.692**
1.44B	2.644	2.608	2.641	2.620	2.609	2.618	**2.601**
2.09B	2.572	2.531	2.566	2.543	2.530	2.542	**2.527**
2.47B	2.536	2.496	3.226[†]	2.512	2.496	2.501	**2.493**

[†]Training instability (outlier).

A key observation is that Qwen3 models are generally deeper than Enoki models at comparable parameter counts. For a given number of heads h, Qwen3 has $2h$ layers while Enoki has only $3h/4$ layers. Since deeper models can express more complex compositional functions but may also exhibit greater sensitivity to learning rate and more challenging optimization landscapes (Wortsman et al., 2023), this architectural difference has implications for optimizer selection.

G.3 QWEN3 SCALING RESULTS

Figure 20 and Table 19 present scaling law fits for Qwen3.

All optimizers achieve good fits ($R^2 > 0.99$) when excluding heads < 7. DANA-MK4 ($\kappa = 0.75$) achieves the highest R^2 (0.999). The $\kappa = 0.85$ variants (both DANA-STAR-MK4 and DANA-MK4) consistently achieve lower absolute losses than their $\kappa = 0.75$ counterparts, with DANA-MK4 $\kappa = 0.85$ achieving the best loss at most scales (Table 18). ADEMAMIX exhibits training instability at 17 heads (loss diverges to 3.23 from expected $\sim$2.5), preventing a reliable power law fit.

Figure 21 shows the fit residuals for Qwen3 using the Chinchilla compute formula ($6ND$). All DANA variants achieve excellent fits with residuals well within ± 0.01.

Comparison with Enoki scaling exponents. The broken power law structure is broadly similar between architectures. For Enoki (Table 13, shared $a = 0.106$), the large-scale exponent $f \approx 0.029$–0.030 is nearly identical across optimizers; for Qwen3, $f \approx 0.030$–0.032 is likewise consistent. The small-scale exponent c shows more variation: Enoki achieves $c \approx 0.211$–0.218, while Qwen3 ranges from $c = 0.151$ (DANA-MK4 $\kappa = 0.75$) to $c = 0.199$ (DANA-STAR-MK4 $\kappa = 0.85$). This suggests Enoki exhibits slightly steeper initial scaling improvement. Despite these differences, the relative ordering of optimizers is preserved: DANA variants consistently outperform ADAMW on both architectures, with the compute savings persisting at scale.

G.4 QWEN3 LEARNING RATE SCALING

For Qwen3, the smallest model sizes (below 100M non-embedding parameters) deviate from power-law scaling and are excluded from the fit. The resulting fits are:

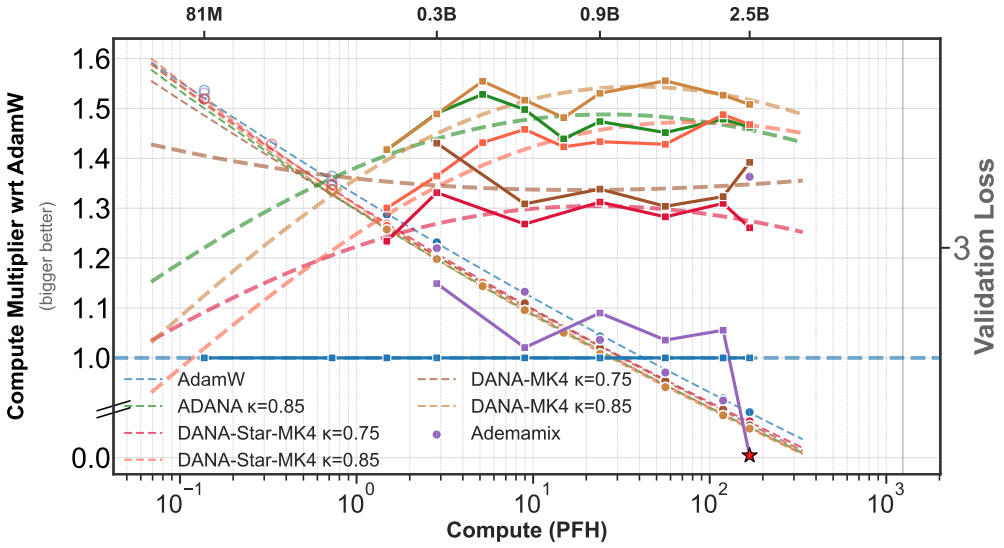


Figure 20: **Qwen3 architecture scaling.** *Left:* Compute savings relative to AdamW. *Right:* Validation loss vs compute with broken power law fits. ADANA ($\kappa = 0.85$), DANA-STAR-MK4 ($\kappa = 0.75$, 0.85), and DANA-MK4 ($\kappa = 0.75$, 0.85) are compared against ADAMW. ADEMAMIX exhibits training instability at 17 heads (marked with star). DANA-MK4 $\kappa = 0.85$ achieves the best loss at most scales.

Table 19: **Qwen3 scaling parameters.** Broken power law fits $L = a + bC^{-c} + eC^{-f}$ with shared $a = 0.108$ using Chinchilla compute formula. Fits exclude heads < 7. Because the broken power law has five free parameters and Qwen3 has fewer data points than Enoki, we initialize the optimization using values informed by the Enoki fits: $b_0 = 0.40$, $c_0 = 0.20$, $e_0 = 2.50$, $f_0 = 0.03$.

Optimizer	b	c	e	f	R^2
ADAMW	0.403	0.164	2.66	0.032	0.9995
ADANA (κ=0.85)	0.402	0.190	2.61	0.030	0.9999
DANA-STAR-MK4 (κ=0.75)	0.403	0.188	2.62	0.030	0.9997
DANA-STAR-MK4 (κ=0.85)	0.403	0.199	2.62	0.031	0.9998
DANA-MK4 (κ=0.75)	0.401	0.151	2.61	0.032	0.9994
DANA-MK4 (κ=0.85)	0.403	0.197	2.61	0.030	0.9999
ADEMAMIX	—	—	—	—	—

$$\text{ADAMW:} \quad \gamma^* = 2.16 \times 10^4 \cdot (7.20 \times 10^4 + P)^{-0.873}, \tag{67}$$

$$\text{DANA-STAR-MK4 } (\kappa{=}0.85)\text{:} \quad \gamma^* = 9.39 \times 10^1 \cdot (1.71 \times 10^4 + P)^{-0.701}, \tag{68}$$

$$\text{ADANA } (\kappa{=}0.85)\text{:} \quad \gamma^* = 3.17 \cdot (3.99 \times 10^4 + P)^{-0.553}, \tag{69}$$

$$\text{DANA-MK4 } (\kappa{=}0.85)\text{:} \quad \gamma^* = 1.20 \cdot (3.44 \times 10^4 + P)^{-0.501}. \tag{70}$$

The ADAMW fit exhibits the largest exponent ($d \approx -0.87$), while the DANA variants show more moderate exponents ($d \approx -0.50$ to -0.70).

G.5 KEY FINDINGS

The Qwen3 experiments confirm that optimizer improvements transfer across architectures—ADANA, DANA-STAR-MK4, and DANA-MK4 variants consistently outperform ADAMW on both Enoki and Qwen3. Several architecture-specific patterns emerge:

1. **Hardened variant outperforms on Qwen3.** Unlike Enoki where ADANA achieves the best scaling exponent, DANA-MK4 ($\kappa = 0.85$) achieves the best final validation loss at most scales on Qwen3 (Table 18). This may reflect greater sensitivity to κ in the deeper Qwen3 architecture, where the hardened variant's reduced κ-sensitivity provides an advantage.
2. **Star variant (τ estimator) has limited impact at this scale.** The DANA-STAR-MK4 variants do not substantially outperform DANA-MK4 on Qwen3, consistent with Enoki results. The τ probability estimator appears most beneficial at larger scales or with sparser gradient distributions.
3. **ADEMAMIX exhibits training instability.** At 17 heads ($\sim$2.5B parameters), ADEMAMIX diverges (loss 3.23 vs. expected $\sim$2.5), preventing reliable power law fits. All DANA variants remain stable across the full scale range.
4. **Learning rate scaling differs between architectures.** The LR exponent for ADAMW is -0.87 on Qwen3 versus -0.49 on Enoki (Section F.3), while DANA variants show more moderate exponents (-0.50 to -0.70) on both. This suggests architecture-specific LR tuning is necessary, though DANA variants are less sensitive to this choice.
5. **Scaling law structure differs but relative rankings persist.** Qwen3 exhibits larger small-scale exponents c and smaller large-scale exponents f compared to Enoki, yet DANA variants consistently outperform ADAMW on both architectures.

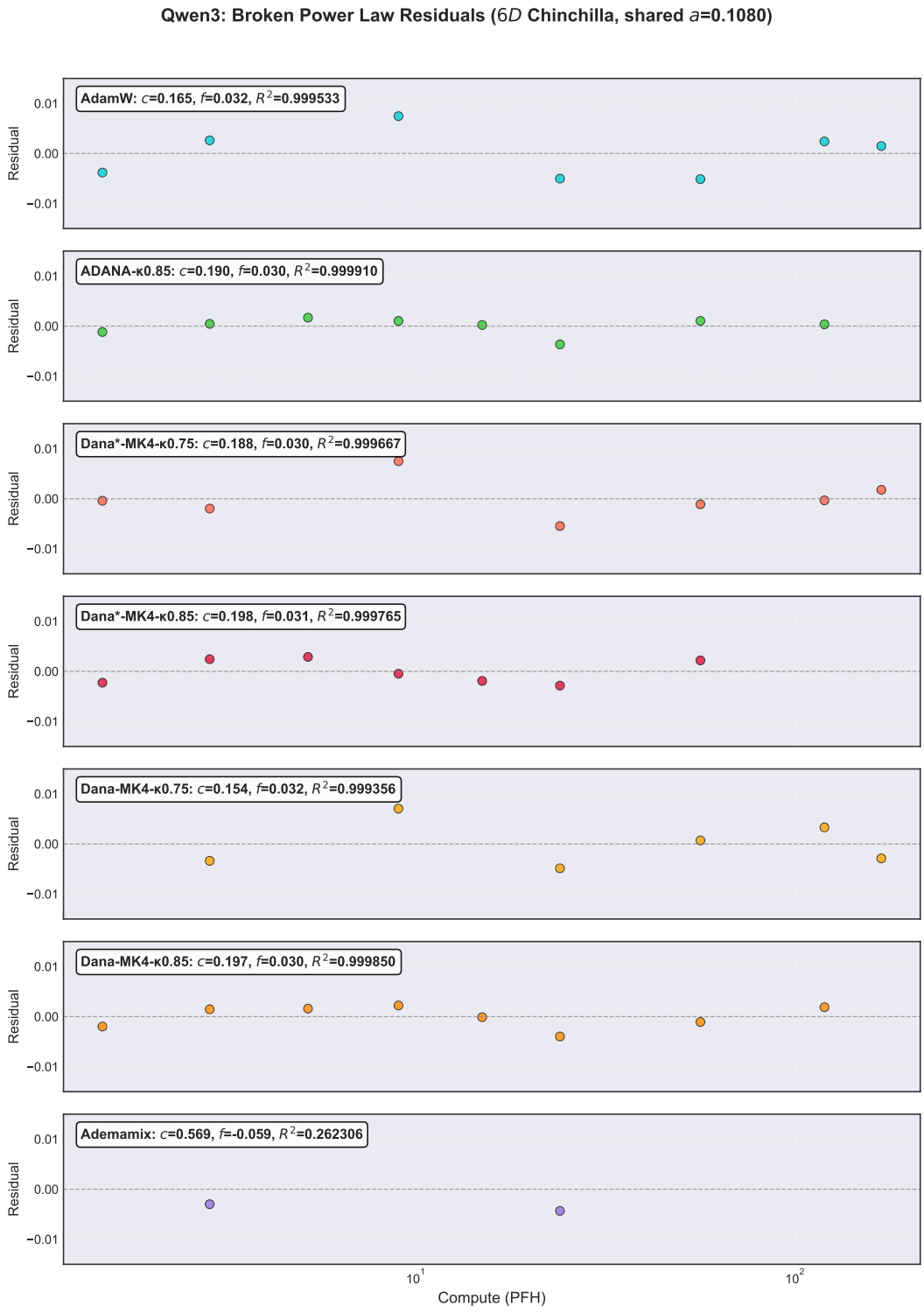
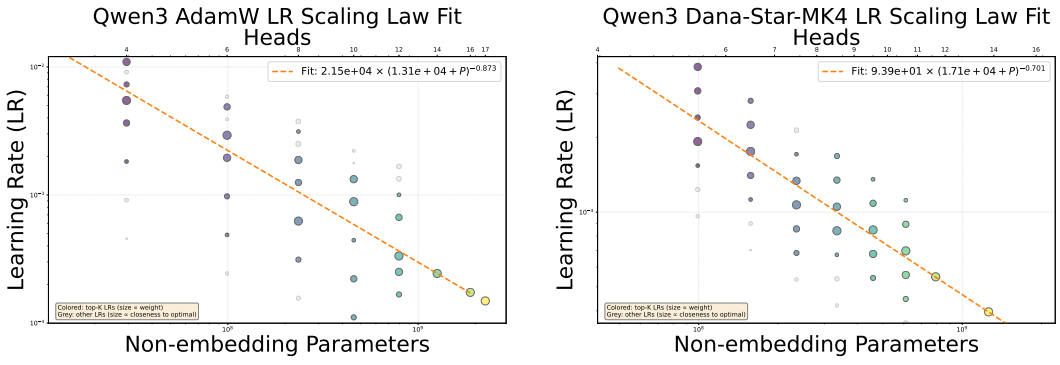
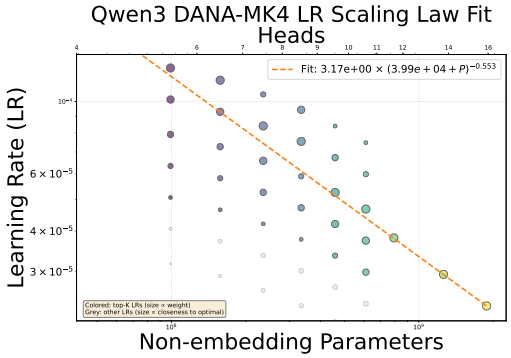


Figure 21: **Qwen3 broken power law fit residuals.** Residuals from fitting $L = a + bC^{-c} + eC^{-f}$ with shared $a = 0.108$ using Chinchilla compute formula ($6ND$). Fits use heads ≥ 7. All DANA variants achieve excellent fits ($R^2 > 0.999$), while ADEMAMIX shows poor fit quality due to training instability.

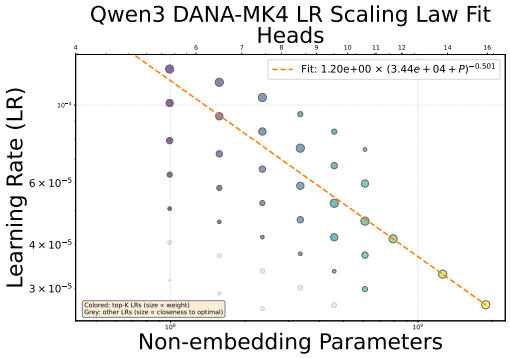


(a) AdamW

(b) Dana-Star-MK4 $\kappa = 0.85$



(c) ADana $\kappa = 0.85$



(d) Dana-MK4 $\kappa = 0.85$

Figure 22: **Peak learning rate scaling for Qwen3 models.** Scales below 100M non-embedding parameters are excluded from the fit.

H IMPACT OF BATCH SIZE

In this section, we discuss the role of batch size in ADANA. We describe our experimental setup, discuss the theoretical implications of batch size for logarithmic time scheduling, and demonstrate that ADANA maintains its scaling advantages at larger batch sizes.

Our primary experiments use a batch size of 32 sequences (approximately 65K tokens per batch with sequence length 2048). This small-batch regime offers several advantages: it is more FLOP-efficient (Marek et al., 2025), requires less memory, and allows for more frequent parameter updates. Larger batch sizes are typically motivated by hardware utilization on multi-GPU systems rather than optimization benefits.

β_2 and batch size stability. Following Brown et al. (2020); Touvron et al. (2023a), our ADAMW baseline uses $\beta_2 = 0.95$ rather than the default $\beta_2 = 0.999$. Liu et al. (2019) found that $\beta_2 = 0.98$ improves stability when training with large batch sizes, and Chen et al. (2020) found that $\beta_2 = 0.95$ reduced loss spikes compared to $\beta_2 = 0.999$. We do not modify the ADAMW β_2 in our experiments; these findings serve as context for potential instabilities when $\beta_2 \to 1$.

Note that under our logarithmic time framework, ADANA uses $\beta_2(k) = 1 - \delta/k$, which sends $\beta_2 \to 1$ as training progresses. One motivation for the large-batch experiments in this section is to verify that this asymptotic behavior does not introduce instabilities in the large batch setting, where prior work has found β_2 close to 1 to be problematic.

Batch size and gradient noise. Under the hypothesis that the limiting noise in language model training arises from feature learning on Zipfian-distributed data (see Appendix M), we expect the correct way to scale batch is through a batch-dependent rescaling of the γ_3 damping parameter rather than standard noise reduction arguments. Specifically, if gradient noise were purely due to sampling variance, it would decrease as $1/\sqrt{B}$ with batch size B. However, for language data, the noise from learning semantic concepts—which exhibits power-law structure characterized by Hilberg's exponent (see Appendix M)—persists across batch sizes, so while larger batches do reduce noise, this motivates keeping the same logarithmic time schedule while adjusting γ_3 by a batch-dependent constant.

H.1 IMPACT OF BATCH SIZE ON THE DAMPING SCHEDULE $\alpha(t)$

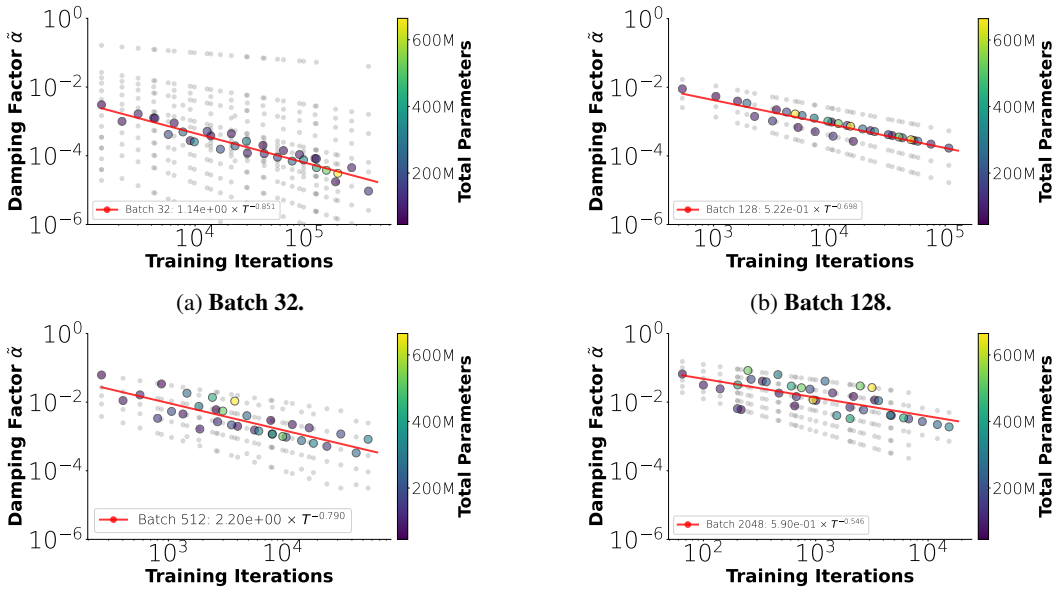
In Figure 5a (main text) we studied the dependence of the damping factor $\tilde{\alpha}$ on training time and model scale in the context of the DANA-CONSTANT-type damping schedule $\alpha(t) = \tilde{\alpha}t$. In Figure 23, we extend this analysis by varying both the iterations, model sizes, and batch size (number of sequences seen at each iteration) between 32 and 512.

For a fixed batch size, the optimal $\tilde{\alpha}$ continues to scale as a power law in the number of iterations T and remains approximately independent of model size. However, increasing batch size increases the optimal $\tilde{\alpha}$ allowed. This is consistent with the idea that increasing batch size decreases the gradient noise, hence improving stability and alleviating the need for a small damping schedule $\alpha(t)$. Note that in the limit of large batch sizes we expect Stochastic Nesterov to be stable and hence $\tilde{\alpha} \equiv 1$ to be stable.

The empirical relationship shown in Figure 23 suggests that the optimal $\gamma_3 = \tilde{\alpha} \cdot T^{1-\kappa}$ scales as a power law in batch size, i.e., $\tilde{\alpha} \propto B^{\eta}$ for some exponent $\eta > 0$. Based on our fits, increasing batch size from 32 to 512 (a factor of 16) requires approximately a 2.5× increase in $\tilde{\alpha}$ to maintain optimal performance and implies an exponent of $\eta \approx 0.33$.

H.2 BATCH SIZE 256 RESULTS

We conducted a sweep at batch size 256 (approximately 524K tokens per batch) to verify that the scaling improvements of ADANA and DANA-MK4 transfer to larger batch regimes. Results are presented in Figure 24, and show that ADAMW performance is increased substantially at small scales, that MUON benefits even more substantially, but that ADANA catches up to this performance around $1B$ parameters. We note that this is without incorporating batch adjustments to the momentum term. A main observation is that ADANA performs worse that ADAMW for very small compute budget but its performance increases with scale and improves over ADAMW starting medium scale compute. An



(a) **Batch 32.** (b) **Batch 128.**

(c) **Batch 512.** (d) **Batch 2048.**

Figure 23: **Impact of batch size on $\alpha(t)$.** The optimal damping constant $\tilde{\alpha}$ in the schedule $\alpha(t) = \tilde{\alpha}t$ scales as a power law in iterations T. Increasing batch size from 32 to 2048 increases the optimal $\tilde{\alpha}$, consistent with reduced gradient noise at larger batches.

interpretation is that at very small compute budget and large batch, the number of iterations to run is very small which may impact results: 1743 iterations for batch 256 against 13950 for batch 32 on Heads $= 6$ Enoki model. At moderate scales Heads $= 24$, batch 256 performs a larger number of iterations of 25335.

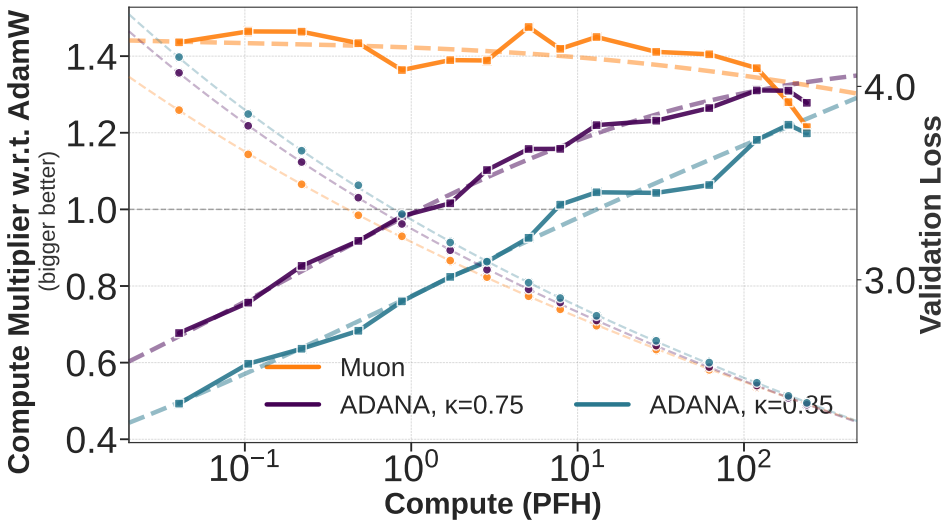
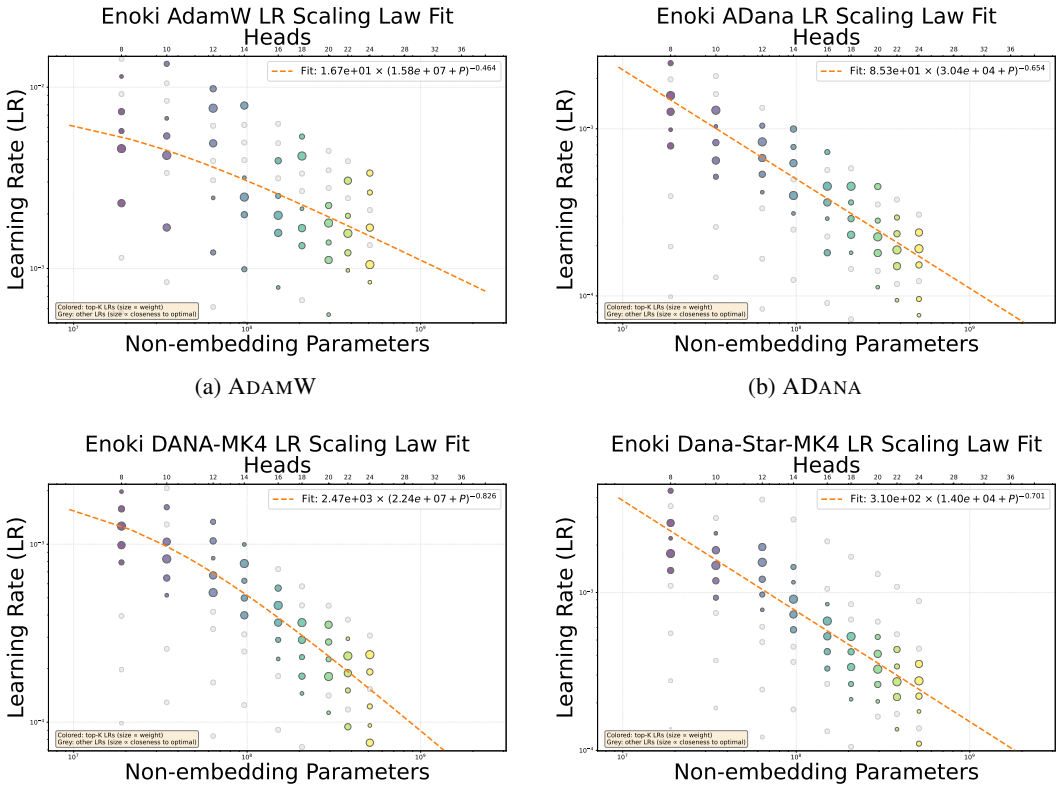


Figure 24: **Batch size 256 scaling comparison.** Validation loss and compute multipliers with respect to AdamW (constant WD) for Muon and ADANA with $\kappa \in \{0.75, 0.85\}$ using logarithmic time weight-decay. Batch size is 256 with sequence length 2048. The compute efficiency gains of ADANA persist at this larger batch size.



(a) AdamW (b) ADana (c) Dana-MK4 (d) Dana-Star-MK4

Figure 25: **LR scaling at batch size 512.** Optimal learning rate as a function of model size for AdamW, ADana, Dana-MK4, and Dana-Star-MK4 at batch size 512. The fitted saturated power laws $\gamma^* = a(b + P_{\text{non-emb}})^d$ are: AdamW: $\gamma^* = 16.7\,(1.58 \times 10^7 + P)^{-0.464}$; ADana ($\kappa = 0.85$): $\gamma^* = 85.3\,(3.04 \times 10^4 + P)^{-0.654}$; Dana-MK4 ($\kappa = 0.85$): $\gamma^* = 2.47 \times 10^3\,(2.24 \times 10^7 + P)^{-0.826}$; Dana-Star-MK4 ($\kappa = 0.85$): $\gamma^* = 310\,(1.40 \times 10^4 + P)^{-0.701}$.

H.3 Batch Size 512 with Rescaled γ_3

For batch size 512 (approximately 1M tokens per batch), we rescale γ_3 by a factor of 2.5, derived from the empirical fit in Figure 23. We re-baseline AdamW at this level, and re-fit the scaling rules for the learning rate. The resulting figures are provided in Figure 25 and 26.

H.4 Discussion

These results suggest that at larger batch sizes, both ADana and AdamW continue to scale, although at small model scales AdamW outperforms ADana, whereas at larger scales the slope of the scaling law fit appears even better for ADana with large batch. This is consistent with the Hilberg hypothesis discussed in Appendix M.

The hardened variants (Dana-MK4) provide additional stability margin at larger batch sizes. We have not observed instability issues at batch size 512 that would suggest $\beta_2 \approx 1$ is problematic in this regime, consistent with our choice of the log-time schedule $\beta_2(k) = 1 - \delta/k$ which naturally adapts to the training horizon.

The key practical implication is that when scaling batch size, practitioners should scale γ_3 (or equivalently $\tilde{\alpha}$) according to the power-law relationship shown in Figure 23, rather than keeping these hyperparameters fixed. This differs from standard batch size scaling rules for learning rate (Goyal et al., 2017) because the damping parameter controls the stability of the long-momentum term rather than the gradient step size.

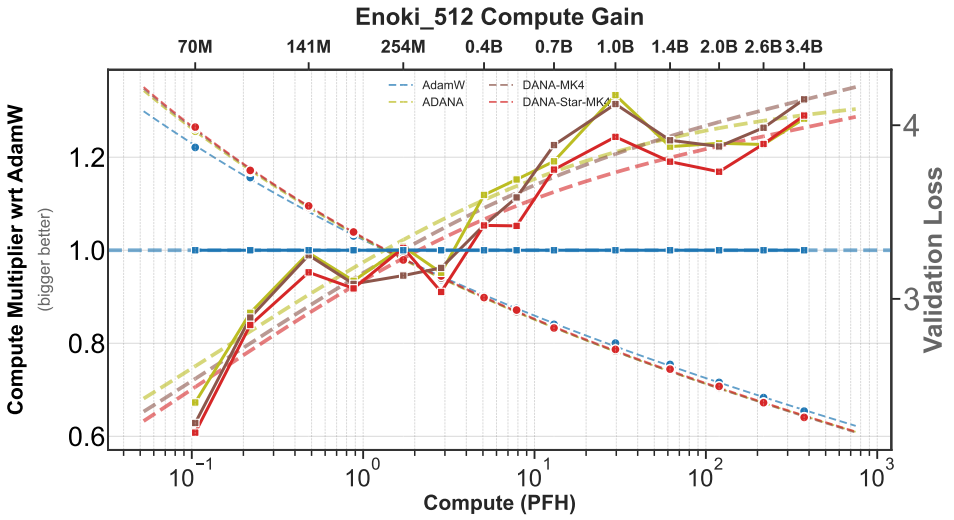


Figure 26: **Batch size 512 scaling comparison.** Validation loss comparing ADAMW, ADANA, DANA-MK4, and DANA-STAR-MK4 at batch size 512 with $\tilde{\alpha} = 2.5$.

I MEASURING THE DATA EXPONENT

In the Power-Law Random Features (PLRF) model and its Mixture-of-Experts extension (MOE-PLRF, see Appendix B), the data covariance matrix has eigenvalues decaying as $\lambda_j \propto j^{-2\rho}$, where the exponent 2ρ characterizes the effective dimensionality of the data. From the theoretical analysis in (Ferbach et al., 2025), the optimal choice of the momentum exponent is $\kappa = 1/(2\rho)$, which can be interpreted as the spectral dimension of the problem. When $2\rho > 1$, the data is approximately low-dimensional and acceleration is possible; when $2\rho < 1$, the data becomes effectively high-dimensional.

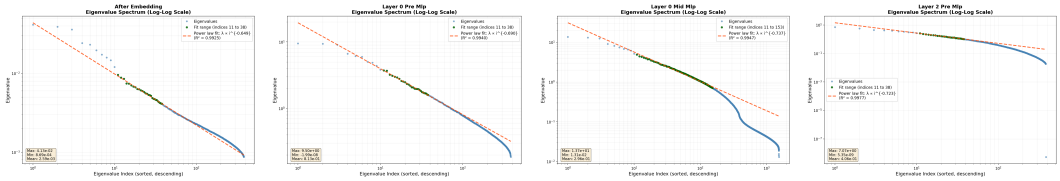
While the activation covariance spectra measured in transformers are not identical to the PLRF covariance structure, we expect a meaningful relationship to exist. To investigate this, we measure the eigenvalue distributions of activation covariances at different layers of our transformer models, both before and after training. We fit power laws of the form $\lambda_j \propto j^{-2\rho}$ to these spectra and extract the corresponding exponent.

We observe the following behaviors of the data exponent at different layers of the transformers:

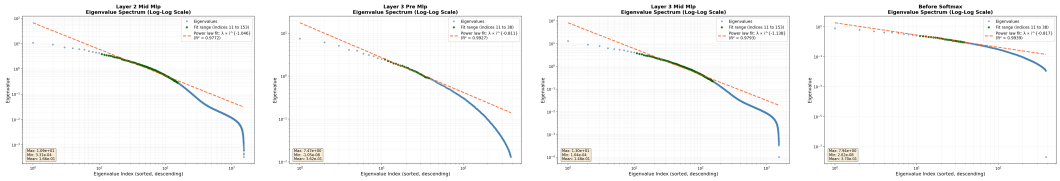
- The data exponent 2ρ increases with layer depth.
- The data exponent 2ρ decreases during training.

In particular, the largest data exponents (especially those above the high-dimensional threshold $2\rho = 1$) are reached at initialization for deep layers. After training, or for shallower layers, the exponent is often below the high-dimensional line.

Notably, the optimal observed κ for the ADANA algorithm in our experiments ($\kappa \approx 0.75$–0.85) corresponds to an effective spectral dimension consistent with measured exponents $2\rho \approx 1.2$–1.3 in the activation covariance spectra. This provides empirical support for the connection between the theoretical PLRF analysis and practical transformer optimization.



(a) Layer 0: After embedding (b) Layer 0: Pre-MLP (c) Layer 0: Mid-MLP (d) Layer 2: Pre-MLP

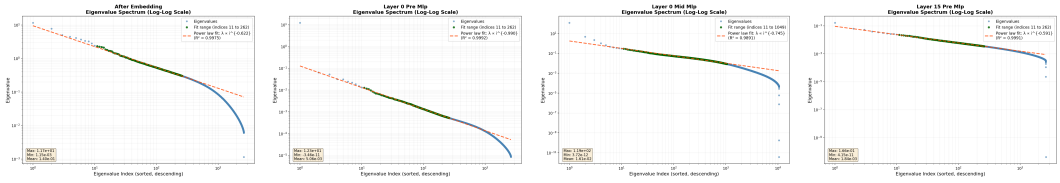


(e) Layer 2: Mid-MLP (f) Layer 3: Pre-MLP (g) Layer 3: Mid-MLP (h) Final: Before softmax

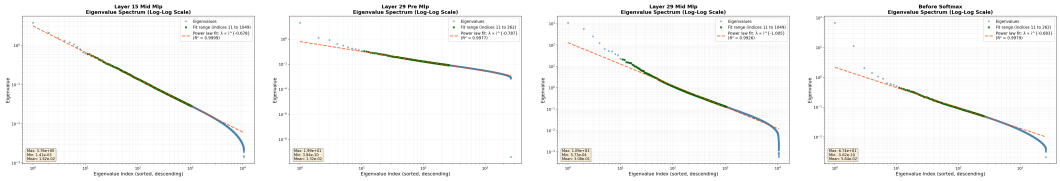
Figure 27: **Eigenvalue distributions at different layers before training (24-head model).** Each subplot shows the eigenvalue spectrum of the activation covariance matrix at a specific layer, with a power-law fit $\lambda_j \propto j^{-2\rho}$. The fitted exponent 2ρ is shown in each panel.

J CORRESPONDENCE WITH ADEMAMIX

In this section, we analyze the connection between ADANA Algorithm 1 and ADEMAMIX Algorithm 6. We show the existence of a scaling rule for ADEMAMIX hyperparameters β_1 and α such that ADEMAMIX bears strong connections with ADANA. In particular, our analysis reveals that ADEMAMIX's warmup schedules implicitly approximate the generalized Nesterov's momentum schedule, providing a theoretical foundation for its empirical success, showed in Figures 6 and 10 using our proposed scaling rule for hyperparameters. The remaining gap in performance between ADEMAMIX and ADANA can further be explained from the ablation study Appendix D.9.

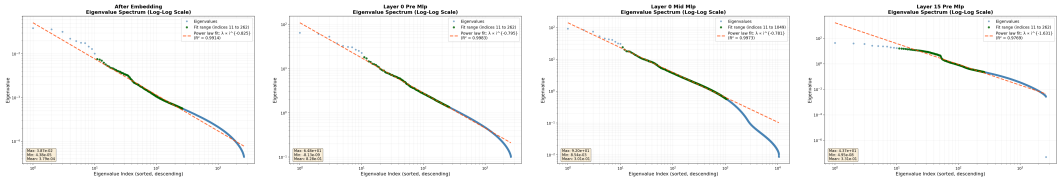


(a) Layer 0: After embedding (b) Layer 0: Pre-MLP (c) Layer 0: Mid-MLP (d) Layer 15: Pre-MLP

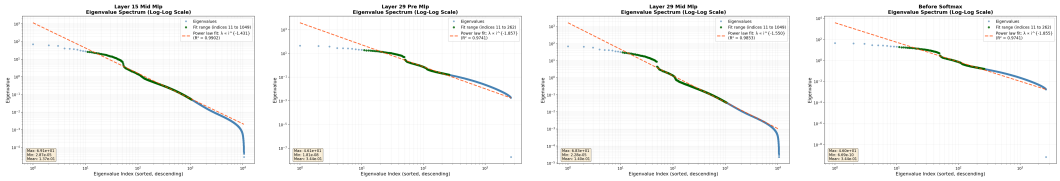


(e) Layer 15: Mid-MLP (f) Layer 29: Pre-MLP (g) Layer 29: Mid-MLP (h) Final: Before softmax

Figure 28: **Eigenvalue distributions at different layers after training (41-head model).** After training, the power-law exponents tend to be smaller (closer to the high-dimensional threshold) compared to initialization.



(a) Layer 0: After embedding (b) Layer 0: Pre-MLP (c) Layer 0: Mid-MLP (d) Layer 15: Pre-MLP



(e) Layer 15: Mid-MLP (f) Layer 29: Pre-MLP (g) Layer 29: Mid-MLP (h) Final: Before softmax

Figure 29: **Eigenvalue distributions at different layers before training (41-head model).** Deeper layers exhibit larger power-law exponents at initialization, suggesting they operate in a more low-dimensional regime.

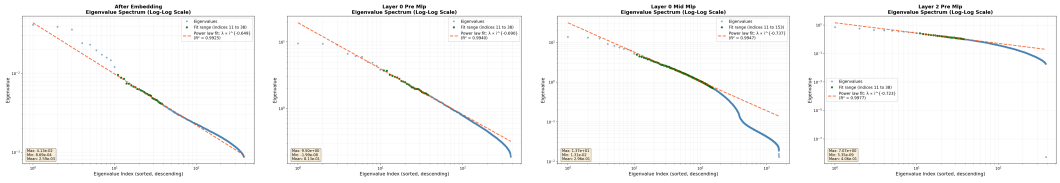
J.1 The AdeMamix Algorithm

In the original AdeMamix algorithm (see Algorithm 6), Pagliardini et al. (2025) use four main hyperparameters: the long-range momentum EMA β_1, the short-range momentum EMA β_3, the second moment EMA β_2, the mixing coefficient α. In particular, AdeMamix originally uses constant values for the short and long-range momentum, respectively $\beta_3(t) \equiv \beta_3,\ \beta_1(t) \equiv \beta_1$, the second moment $\beta_2(t) \equiv \beta_2$, and the mixing coefficient $\alpha(t) \equiv \alpha$.

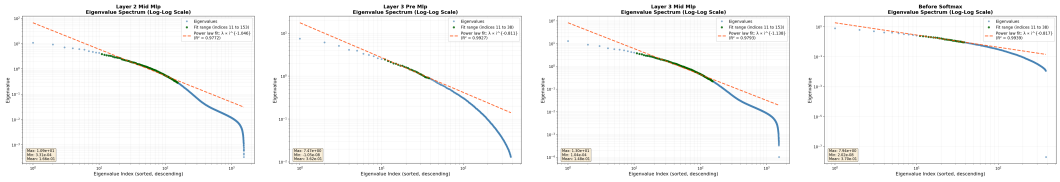
However, in practice Pagliardini et al. (2025) observed that training with constant long-range momentum was unstable. To address this, they proposed two warmup schedules up to time T_α and T_{β_1} for the mixing coefficient $\alpha(t)$ and long momentum $\beta_1(t)$ respectively:

$$\beta_1(t) = \min\left\{\exp\left(\frac{\log\beta_3 \cdot \log\beta_1}{(1-t/T_{\beta_1})\log\beta_1 + (t/T_{\beta_1})\log\beta_3}\right), \beta_1\right\}, \tag{71}$$

$$\alpha(t) = \frac{t}{T_\alpha}\alpha. \tag{72}$$

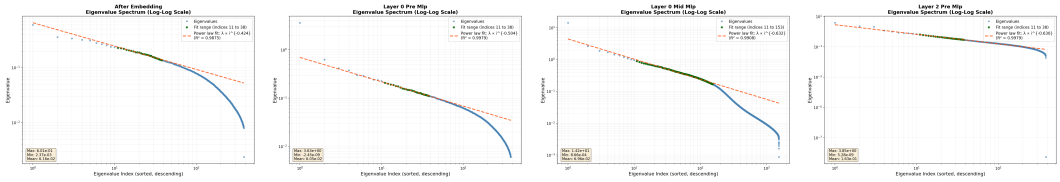


(a) Layer 0: After embedding (b) Layer 0: Pre-MLP (c) Layer 0: Mid-MLP (d) Layer 2: Pre-MLP

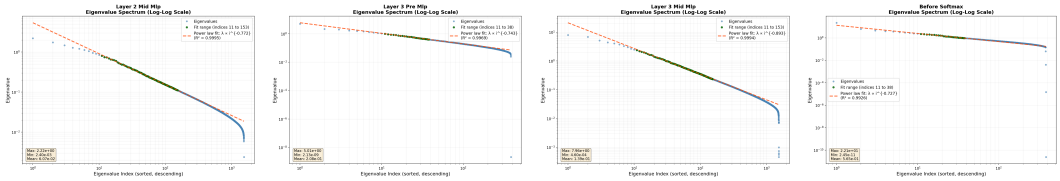


(e) Layer 2: Mid-MLP (f) Layer 3: Pre-MLP (g) Layer 3: Mid-MLP (h) Final: Before softmax

Figure 30: **Eigenvalue distributions at different layers before training (6-head model).** Smaller models show similar qualitative trends but with generally smaller exponents.



(a) Layer 0: After embedding (b) Layer 0: Pre-MLP (c) Layer 0: Mid-MLP (d) Layer 2: Pre-MLP



(e) Layer 2: Mid-MLP (f) Layer 3: Pre-MLP (g) Layer 3: Mid-MLP (h) Final: Before softmax

Figure 31: **Eigenvalue distributions at different layers after training (6-head model).** After training, the spectra become more uniform, with reduced power-law exponents.

The motivation for the $\beta_1(t)$ is to increase linearly the half-time decay of the EMA linearly throughout training. They additionally typically set the warmup horizon T_{α,β_1} to the total number of training iterations T.

J.2 Equivalence of Schedules

In this section, we show that under a particular scaling of α, β_1 and β_3, and under the warmup schedules AdemaMix is approximately equivalent to the ADana algorithm. More precisely we will explain the following:

Correspondence between AdemaMix and ADana: Let a training horizon T, constant δ (typically $\delta = 8$) and $\kappa \in (0, 1)$. Set AdemaMix hyperparameters as $\beta_1 = 1 - \delta/T$ (logarithmic-time long momentum), $\beta_3 \in (0, 1)$ any constant (typically $\beta_3 = 0.9$ in practice, but $\beta_3 = 0.0$ for the correspondence), mixing coefficient $\alpha = T^{1-\kappa}$ and warmup schedules $T_\alpha = T_{\beta_1} = T$. Then AdemaMix is approximately equivalent to ADana with DANA-CONSTANT-type schedule $\alpha(t) = T^{-\kappa} \times t$.

In the following we show the two main equivalence on schedules between ADEMAMIX and ADANA: the long-momentum schedule $\beta_1(t)$ and mixing coefficient $\alpha(t)$.

J.2.1 LOG-TIME MOMENTUM EXPONENTIAL MOVING AVERAGE (EMA) $\beta_1(t)$.

We first show in Theorem J.1 that by setting $\beta_1 = 1 - \delta/T$s, the warmup schedule (71) asymptotically approximates the ADANA Nesterov-style logarithmic-time momentum schedule $\beta_1(t) = 1-\delta/(\delta+t)$.

Lemma J.1 (ADEMAMIX schedule approximates DANA schedule)**.** *Let $\delta > 0$, $0 < \beta_3 < 1$ (short momentum), and define for any $T > 0$, $\beta_1 = 1 - \delta/T$ (target long momentum). Suppose that $0 < \beta_1 < 1$ and define the* ADEMAMIX *schedule*

$$\beta_1^{\text{ADEMAMIX}}(t) = \min\left\{\exp\left(\frac{\log\beta_3 \cdot \log\beta_1}{(1-t/T)\log\beta_1 + (t/T)\log\beta_3}\right), \beta_1\right\}.$$

Then as $t, T \to \infty$ with $0 < t \leq T$, we have

$$1 - \beta_1^{\text{ADEMAMIX}}(t) = \frac{\delta}{t}\big(1 + \mathcal{O}_t(1) + \mathcal{O}_T(1)\big).$$

Proof. Since $\beta_1 = 1 - \delta/T$, we can write $\log(\beta_1) = -\delta/T \cdot (1 + \mathcal{O}_T(1))$. Substituting into the schedule formula, for $0 < t \leq T$:

$$\begin{aligned}
\beta_1^{\text{ADEMAMIX}}(t) &= \exp\left(\frac{\log(\beta_3)\cdot(-\delta/T)(1+\mathcal{O}_T(1))}{(t/T)\log(\beta_3)\big(1+\mathcal{O}_t(1)+\mathcal{O}_T(1)\big)}\right) \\
&= \exp\left(\frac{-\delta/t\cdot(1+\mathcal{O}_T(1))}{1+\mathcal{O}_t(1)+\mathcal{O}_T(1)}\right) \\
&= 1 - \frac{\delta}{t}\big(1+\mathcal{O}_t(1)+\mathcal{O}_T(1)\big).
\end{aligned}$$

□

Theorem J.1 shows that setting with $\beta_1 = 1 - \delta/T$, the ADEMAMIX warmup schedule converges to the ADANA logarithmic time schedule $\beta_1(t) = 1 - \delta/(\delta + t)$ as $t, T \to \infty$. This is precisely the schedule that enables acceleration when δ is chosen sufficiently large (Ferbach et al., 2025). δ is typically set to 8 in our experiments.

J.2.2 MIXING COEFFICIENT $\alpha(t)$ SCHEDULE.

The original ADEMAMIX paper proposes a constant schedule $\alpha(t) \equiv \alpha$. Within the GENERALIZED NESTEROV/ DANA framework (22), a constant $\alpha(t) \equiv \alpha$ corresponds to using DANA-DECAYING with power exponent $\kappa = 1$. In particular, the momentum schedule does not increase fast enough to allow for acceleration.

However, to avoid instabilities, Pagliardini et al. (2025) typically use a linear warmup schedule $\alpha(t) = \alpha \times \frac{t}{T}$. As explained in Appendix D.7, this schedule recovers the DANA-CONSTANT schedule when scaling the constant α appropriately. In particular, setting $\alpha = T^{1-\kappa}$ recovers the damped schedule $\alpha(t) = (1+t)^{1-\kappa}$ around $t = T$ and is predicted to match the performance DANA-DECAYING schedule $\alpha(t) = (1+t)^{1-\kappa}$ around $t = T$ (Ferbach et al., 2025).

J.3 EMPIRICAL VERIFICATION OF $\alpha(t)$ SCALING: SATURATION ON LONG TRAINING RUNS

In Figure 5a we we fit the optimal damping factor (for DANA-CONSTANT-type damping schedule $\alpha(t) = \tilde{\alpha} \times t$) with respect to training time on different model scales and tokens to parameter ratios. While in Figure 5a we see that the optimal $\tilde{\alpha}$ follows a power-law dependence on T, training far beyond the compute optimal regime reveals a saturation of the exponent κ, as shown in Figure 33. This effect is particularly visible for smaller models, where the power-law behavior eventually degrades at very long training horizons.

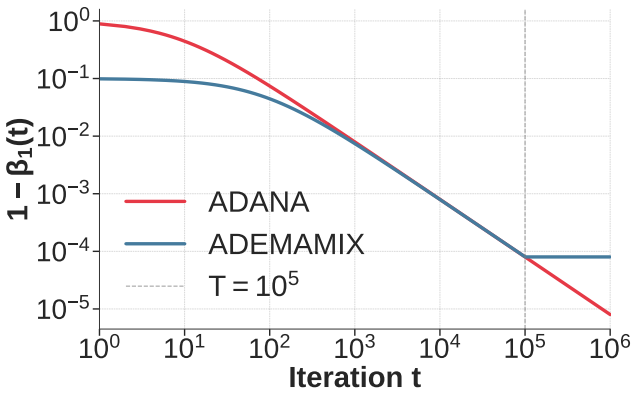
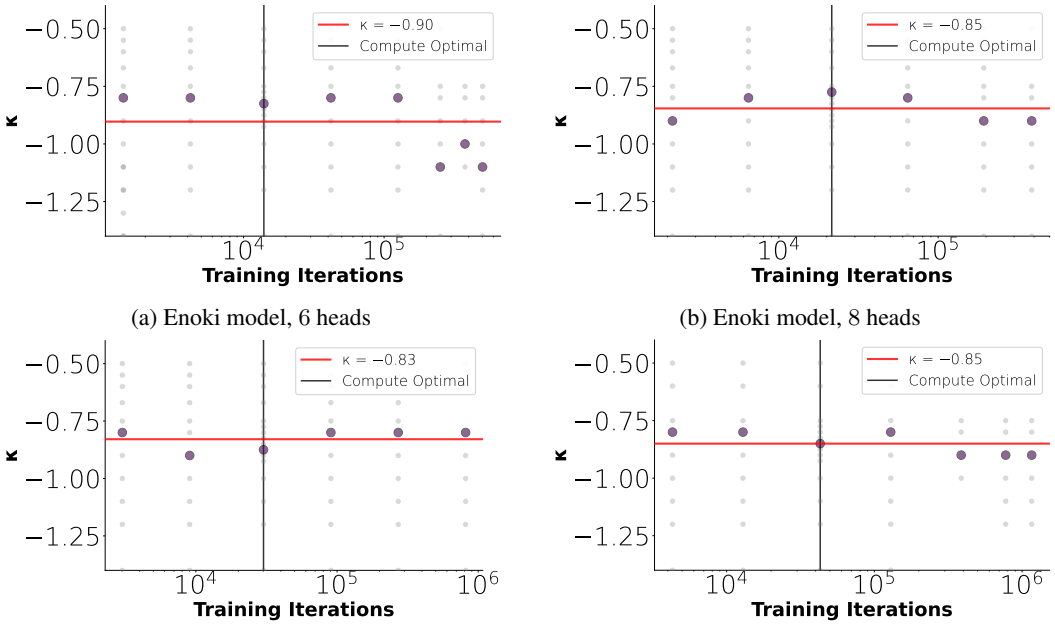


Figure 32: Comparison of long-range momentum schedules for ADANA $1 - \beta_1(t) = \frac{\delta}{\delta+t}$ and for ADEMAMIX using warmup $T_{\beta_1} = T$, $\beta_3 = 0.9$, $\beta_1 = 1 - \frac{\delta}{T}$ with $\delta = 8$ and $T = 10^5$. While both schedules differ at the start of training they match later in training. Note that after $t \geq T$, ADEMAMIX's schedule becomes constant $\beta_1(t) = \beta_1 = 1 - \frac{\delta}{T}$.



(a) Enoki model, 6 heads

(b) Enoki model, 8 heads

(c) Enoki model, 10 heads

(d) Enoki model, 12 heads

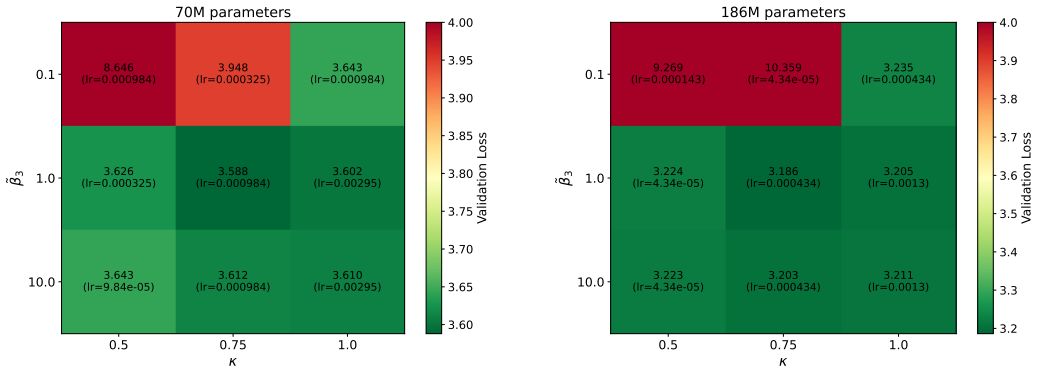
Figure 33: **Evolution of optimal $\kappa \stackrel{\text{def}}{=} \log_T(\tilde{\alpha})$ for different model sizes when training far beyong the compute optimal point.** The power-law relationship $\tilde{\alpha} \approx T^{-\kappa}$ holds across different model scales and training horizons but breaks when training for too long (see heads 6, 8 especially.

J.4 Conclusion of the Ablation in Appendix D.9

Due to the close relationship between ADANA and ADEMAMIX, under the hyperparameters scaling noted above, it is natural to ask how their performance compare. In Appendix D.9, we answer this question by conducting an ablation study on ADANA schedules to understand why ADEMAMIX underperforms ADANA in Figure 1. The main conclusion is that the DANA-CONSTANT-type $\alpha(t) = T^{-\kappa} \times t$ schedule contributes to degrading performance while the short momentum EMA $\beta_3 = 0.9$ tends to increase performance compared to $\beta_3 = 0.0$, probably due to improved stability. Finally $\beta_2 = 0.999$ constant as used in ADEMAMIX may have a limited impact compared to logarithmic-time ADANA $\beta_2(t) = 1 - \frac{\delta}{\delta+t}$ although it seems to increase instability at scale.

J.5 Optimality of logarithmic-time schedules

In Appendix J.5, we conducted a joint hyperparameter search for ADEMAMIX by jointly sweeping κ, the learning rate γ and $\tilde{\beta}_3$ where ADEMAMIX hyperparameters are $\beta_1 = 0.9,\ \beta_2 = 0.999$, $\beta_3 = 1 - \tilde{\beta}_3 \times \frac{\delta}{T}$ and $\alpha = T^{1-\kappa}$. We see that for both model sizes 70M and 186M parameters, the optimal loss is attained at $\kappa = 0.75,\ \tilde{\beta}_3 = 1$. In particular, the choice $\tilde{\beta}_3 = 1$ reveals that ADANA's logarithmic-time parametrization $\beta_3 = 1 - \frac{\delta}{T}$ is optimal.



(a) 8 heads.

(b) 14 heads.

Figure 34: **ADEMAMIX hyper-parameters sweep.** Optimal damping exponent κ and long momentum $\beta_3 = 1 - \tilde{\beta}_3 \times \frac{\delta}{T}$ of ADEMAMIX for two different model sizes. Learning rate is tuned over a grid of 5 values with relative ratio 3. In particular, ADANA's scaling rules $\tilde{\beta}_3 = 1,\ \kappa = 0.75$ is optimal for both sizes.

J.6 Discussion

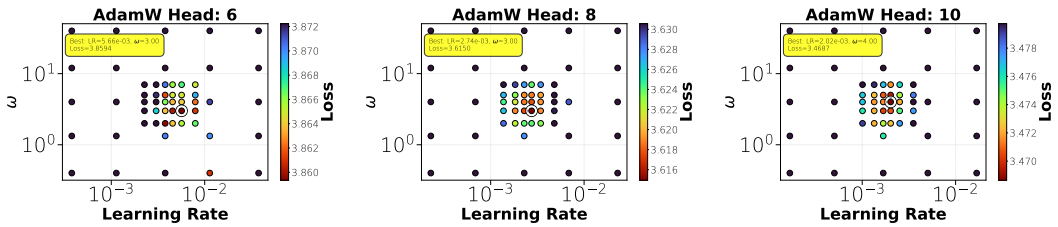
The connection between ADANA and ADEMAMIX demonstrates that the key insight—long-term memory with appropriate scaling—can be implemented in multiple ways. ADANA's approach of using time-dependent β_1, β_2 values is more direct and requires fewer hyperparameters, while ADEMAMIX achieves similar effects through its three-buffer structure with appropriate warmup scheduling.

For practitioners, we offer the following recommendations. When using ADEMAMIX, setting $\beta_1 = 1 - \delta/T$ (long momentum) and tuning $\alpha = T^{1-\kappa}$yields strong performance. When starting fresh, ADANA is simpler with fewer hyperparameters to tune and achieves better final performance. The relationship $\beta_1 = 1 - \delta/T$ and $\alpha = T^{1-\kappa}$ in ADEMAMIX provides the bridge between the two frameworks.

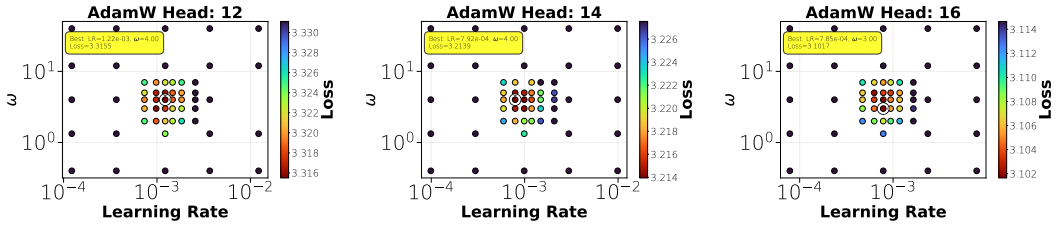
K Additional Weight Decay Details

Intuition for Logarithmic-Time Weight-Decay We previously motivated logarithmic-time schedules by the mismatch between the exponential forgetting induced by constant $\beta_1, \beta_2, \lambda$ and the increasing time scale over which meaningful updates occur during training.

A complementary perspective is obtained by assuming approximate power-law training dynamics. Concretely, suppose the loss $\mathcal{R}(\theta_t)$ and related quantities (such as expected gradients) evolve smoothly with iteration t, in a way that is well approximated by power-law scaling. In this regime, these quantities change slowly on multiplicative time scales (e.g., between t and $2t$), rather than on fixed additive time intervals. This behavior is commonly observed in language modeling, where losses, gradient norms, and related statistics exhibit approximate power-law trends over the course of training.

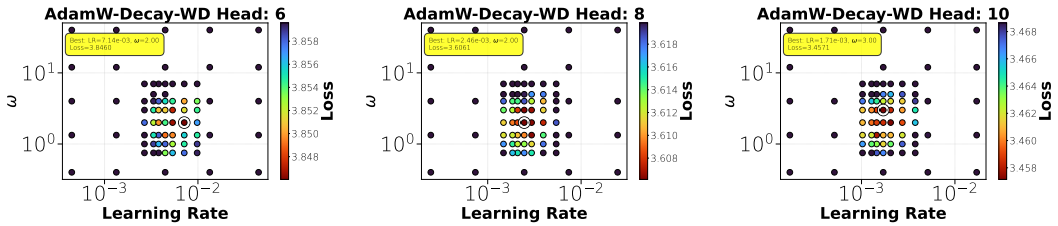


(a) 6 heads (b) 8 heads (c) 10 heads

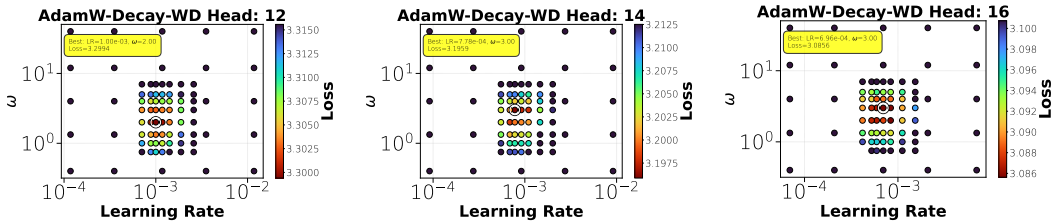


(d) 12 heads (e) 14 heads (f) 16 heads

Figure 35: **Validation loss heatmaps for constant weight-decay (AdamW) across different model sizes.** Each heatmap shows the final validation loss as a function of learning rate and weight-decay coefficient ω.



(a) 6 heads (b) 8 heads (c) 10 heads



(d) 12 heads (e) 14 heads (f) 16 heads

Figure 36: **Validation loss heatmaps for time-decaying weight-decay (AdamW-decaying-wd) across different model sizes.** Compared to the constant weight decay heatmaps in Figure 35, the optimal loss values are consistently lower across all scales.

Consider to simplify constant learning rate ($\gamma(t) \equiv 1$). We can approximately rewrite the independent weight-decay update (Independent-WD) in closed form as

$$\theta_t \approx \exp(-\sum_{s=0}^{t-1} \lambda_s) \left(\theta_0 - \gamma^* \sum_{s=0}^{t-1} g_s \exp(\sum_{r=0}^{s} \lambda_r) \right). \tag{73}$$

This expression makes explicit how weight decay reweights past gradient contributions. If the scaled gradients g_s are approximately constant on any fixed relative time window (e.g., $[t/2, t]$), then weight decay should attenuate updates over such windows by a constant factor to avoid degeneracy. This requires $\sum_{r=t/2}^{t} \lambda_r$ to be constant uniformly in t. Under this condition, a harmonic schedule emerges naturally:

$$\lambda_t = \frac{\omega}{t} \tag{74}$$

Approximation to Logarithmic Time Weight-Decay In practice, we use an approximation to logarithmic-time decaying by adding a saturation term $T/10$ to the denominator $\lambda(t) = \frac{\omega}{T/10+t}$ instead of $\lambda(t) = \frac{\omega}{t}$ in the ADANA implementation Algorithm 1 and more generally in all instances of decaying weight-decay in our experiments. Therefore, in effect the weight-decay schedule is only in logarithmic time for a constant fraction of T at the end of training. On the other hand, at the start of training $t \ll T$, the weight-decay is approximately constant $\lambda(t) \approx \frac{10\omega}{T}$ and 10 times larger than with constant weight decay $\lambda(t) = \frac{\omega}{T}$ for the same renormalized factor ω. Despite this approximation, the qualitative behavior of time-decaying weight decay is preserved. Early updates experience stronger regularization while later updates (corresponding to rarer patterns in Zipfian data) experience weaker regularization. This temporal redistribution is what leads to the improved performance we observe empirically.

Additional Details on Sweeps from Figure 7. In Figures 35 and 36 we show the joint sweeps on peak LR γ^* and weight-decay renormalized factor ω at each head size of the Enoki Model and with optimizer ADAMW. We perform sweeps for respectively constant weight-decay $\lambda(t) = \frac{\omega}{t}$ in Figure 35 and logarithmic time decaying $\lambda(t) = \frac{\omega}{T/10+t}$ weight-decay in Figure 36. The learning rate schedule $\gamma(t)$ is the same as in Table 2 (linear warmup and cosine decay).

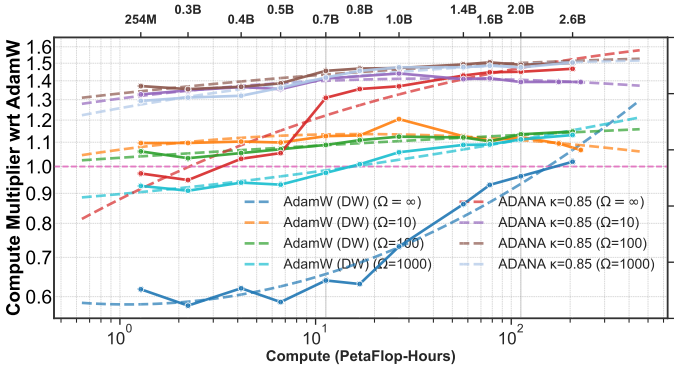


Figure 37: **Warm-up in Weight-decay.** We consider a weight-decay schedule of the form $\lambda(t) = \frac{\omega}{T/\Omega+t}$ and vary $\Omega \in \{10, 100\}$, with $\Omega = \infty$ corresponding to $\lambda(t) = \omega/t$. The parameter Ω acts as a warm-up for weight decay, keeping it approximately constant early in training before transitioning to a $1/t$ decay. We find that choosing $\Omega = 10$ or 100 has little effect on the performance of ADANA or ADAMW (DW), while consistently improving performance relative to ADAMW with constant weight decay. In contrast, the performance of ADAMW (DW) degrades substantially when using the pure ω/t schedule, whereas ADANA remains stable under this schedule, at least for larger model sizes but does worse for smaller sizes. It appears that ADANA is less sensitive to the choice of Ω.

L BUILDING ADANA: ADDITIONAL DETAILS & THEORY

This section develops the intuition behind ADANA (Alg. 1) and why each component is necessary in order for ADANA to improve upon standard ADAM. We additionally discuss some problems/challenges that arise when attempting to add log-time schedules for the 1st/2nd moment buffers of ADAM and how ADANA mitigates these problems.

Suppose $\mathcal{R} : \mathbb{R}^d \to \mathbb{R}$ is the expected risk with loss function $\mathcal{L}$ that takes in parameters $\theta \in \mathbb{R}^d$ and data x. Consider the following learning problem:

$$\min_{\theta \in \mathbb{R}^d} \mathcal{R}(\theta) \stackrel{\text{def}}{=} \mathbb{E}_x[\mathcal{L}(\theta; x)], \tag{75}$$

where we denote the stochastic gradient $g_{t+1} \stackrel{\text{def}}{=} \nabla \mathcal{L}(\theta_t, x_{t+1})$.

Before going into details about each component of ADANA, let us start from one of the widely used algorithms in machine learning, ADAM (Alg. 2). Given initializations $m_0, \theta_0, v_0 \in \mathbb{R}^d$, ADAM

generates an independent data sample x_{t+1} and updates by

$$\begin{aligned} &(1^{st}\ \textit{mom.}) \quad m_{t+1} = \beta_1 m_t + (1-\beta_1) g_{t+1} \\ &(2^{nd}\ \textit{mom.}) \quad v_{t+1} = \beta_2 v_t + (1-\beta_2) g_{t+1}^2 \\ &(\textit{param.}) \quad \theta_{t+1} - \theta_t - \gamma \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon}. \end{aligned}$$

Here $\beta_1, \beta_2 \in (0,1)$ are the 1st and 2nd moment hyperparameters, $\gamma > 0$ is the learning rate, and $\epsilon > 0$ is a numerical stability constant. Moreover we use the notation that $g_{t+1}^2 = g_{t+1} \odot g_{t+1}$. A common practical hyperparameter choice is to set β_1 and β_2 to be fixed constants independent of model size and iteration count, t.

Throughout this discussion, we will refer to the notion of *outscaling*, which we define below.

Outscaling. A *training regime*, $t \asymp d^\ell$, $\ell > 0$, is a scaling of iterations (or samples) to parameters. There are many examples of training regimes, e.g., the *proportional regime* ($t \asymp d$) or the *compute-optimal regime*, in which one selects the ℓ that yields the best loss under a fixed compute budget. Now suppose the loss under an algorithm follows the scaling law

$$\mathcal{R}(t,d) \stackrel{\text{def}}{=} \mathcal{R}(\theta_t, d) \asymp t^{-\sigma} + d^{-\tau} \quad \text{and suppose } t \asymp d^\ell,\ \ell > 0 \text{ is a training regime.} \tag{76}$$

Then under this training regime, the loss satisfies $\mathcal{R}(d^\ell, d) \asymp d^{-\min\{\ell\sigma, \tau\}}$. We call the absolute exponent on d, the *loss exponent*. For a given training regime, we say an algorithm *outscales* another algorithm if the loss exponent is larger.

We emphasize that this notion of outscaling differs in one key aspect from the more traditional notion of "acceleration" from optimization theory. Acceleration is typically formulated for a fixed-dimensional problem with constants that can have large d-dependence (e.g., $\|\theta_0 - \theta^\star\|^2$). In other words, acceleration generally denotes outperformance when $t \to \infty$ and $d = O(1)$.

While model architectures and training methods have advanced rapidly, it has long been unclear whether innovations in *optimization algorithms* could fundamentally change the loss exponents Hestness et al. (2017). Some evidence suggests that major advances like the ADAM optimizer Kingma and Ba (2015) primarily improve the constants in the scaling law rather than improving its exponent Hestness et al. (2017).

L.1 BUILDING ADANA: BENEFIT OF LOG-MOMENTUM AND IMPORTANCE OF DAMPING SCHEDULE $\alpha(t)$

We begin by providing intuitive insight into the benefits of a log-time schedule for β_1. In particular, we explain how combining a ***log-time*** β_1 ***schedule with an*** $\alpha(t)$ ***damping schedule*** is theoretically grounded, leading to compute-efficiency gains that persist as model size increases and to improved loss-scaling exponents relative to SGD.

This section begins by building intuition directly from two optimization algorithms: stochastic gradient descent with momentum,

$$m_{t+1} = \beta_1 m_t + (1-\beta_1) g_{t+1} \quad \text{and} \quad \theta_{t+1} = \theta_t - \gamma(g_{t+1} + m_{t+1}) \tag{SGD-M}$$

and a naive implementation of stochastic Nesterov's accelerated method Nesterov (1983)

$$\theta_{t+1} = y_t - \gamma g_{t+1}, \quad y_{t+1} = \theta_{t+1} + \mu_t(\theta_{t+1} - \theta_t), \quad \text{where} \quad \mu_t = \frac{t}{t+3} = 1 - \frac{3}{t+3}. \tag{S-NAG}$$

L.1.1 CONSTANT β_1 MAY NOT ACCELERATE OR OUTSCALE SGD

Consider a simple momentum-based stochastic algorithm SGD-M that updates by

$$\begin{aligned} &(\textit{mom.}) \quad m_{t+1} = \beta_1 m_t + (1-\beta_1) g_{t+1} \\ &(\textit{param.}) \quad \theta_{t+1} = \theta_t - \gamma g_{t+1} - m_{t+1}. \end{aligned}$$

Assuming that $m_0 = 0$, we can write $m_{t+1} = (1-\beta_1)\sum_{j=0}^{t} \beta_1^j g_{t+1-j}$. In this sense, gradients which are from a long time ago will get killed exponentially fast. In small batch setting, the stochastic

gradient g_t does not change very much on short-time horizons. Note this is not the case in the full-batch setting. Therefore, the momentum parameter $m_{t+1} \approx g_{t+1}$ (in small batch) and we see that the parameter update $\theta_{t+1} \approx \theta_t - \gamma g_{t+1} - \text{constant}(\beta_1) \times g_{t+1}$, that is, SGD-M with constant β_1 (in small batch setting) becomes SGD with a different learning rate. This observation has been proven in specific settings (see e.g., Paquette and Paquette (2021); Kidambi et al. (2018); Sebbouh et al. (2021); Zhang et al. (2019)).

L.1.2 Benefits and issues with adding log-momentum, $\beta_1(t) = 1 - \delta(\delta + t)^{-1}$

It is well known from deterministic optimization that *log momentum* is beneficial for non–strongly convex objective functions. In particular, this corresponds to choosing $\beta_1(t) \asymp (1+t)^{-1}$. This idea is a key component of the celebrated Nesterov accelerated gradient method and explains, in part, why the algorithm achieves acceleration in the deterministic setting. This acceleration is desirable for us. However, long momentum of this form suffers from stability issues when transitioning from deterministic gradients to (small-batch) stochastic gradients (see, e.g., Thm. 7 in Even et al. (2021)).

Instability using S-NAG. We now explain the source of this instability and how it arises in a naive stochastic implementation of Nesterov's accelerated method. Consider the stochastic version of Nesterov's accelerated method Nesterov (1988), defined by the updates

$$\theta_{t+1} = y_t - \gamma \nabla \mathcal{L}(y_t), \quad y_{t+1} = \theta_{t+1} + \mu_t(\theta_{t+1} - \theta_t), \quad \text{where } \mu_t = \frac{t}{t+3}. \tag{77}$$

Here $\gamma > 0$ is the learning rate, and for simplicity we write $\mathcal{L}(y_t) \overset{\text{def}}{=} \mathcal{L}(y_t, x_{t+1})$ to denote a stochastic gradient. Note that Nesterov's accelerated method is originally formulated using full-batch (deterministic) gradients; here, we replace the deterministic gradient with a stochastic one. This constitutes the most naive stochastic variant of Nesterov's accelerated method. We refer to this formulation as *Two-sequence Nesterov*.

An equivalent formulation of *Two-sequence Nesterov* (see Section O.1) is what we call *Extra-gradient Nesterov* (see also Defazio et al. (2025)). The updates take the form

$$m_{t+1} = \mu_{t-1} m_t + g_{t+1} \quad \text{and} \quad \Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \mu_t m_{t+1}), \quad \text{where } g_{t+1} = \nabla \mathcal{L}(\Phi_t, x_{t+1}). \tag{78}$$

We will use this formulation going forward. To build intuition for why instability arises when full-batch gradients are replaced with stochastic ones, suppose that

$$\nabla \mathcal{L}(\Phi_t, x_{t+1}) = \nabla \mathcal{R}(\Phi_t) + \epsilon \quad \text{where } \epsilon > 0 \text{ is some fixed error in the gradient.}$$

In practice, $\epsilon \to 0$, though typically at a slow rate. Unraveling the recursion for m_{t+1}, the contribution of the noise term ϵ takes the form

$$m_{t+1} = \underbrace{\sum_{k=1}^{t} \left(\prod_{i=k+1}^{t} \mu_{i-1} \right) \nabla \mathcal{R}(\Phi_k)}_{\text{(unravel full gradient)}} + \epsilon \cdot \sum_{k=1}^{t} \left(\prod_{i=k+1}^{t} \mu_{i-1} \right) \overset{\text{def}}{=} G_t + \epsilon \cdot \sum_{k=1}^{t} \left(\prod_{i=k+1}^{t} \mu_{i-1} \right).$$

Specializing to $\mu_{t-1} = \frac{t}{t+3}$ yields

$$m_{t+1} = G_t + \epsilon \cdot \frac{t+3}{4}.$$

The key observation is therefore that ***noise in the stochastic gradient accumulates in*** m_{t+1}. From an order-of-magnitude perspective, the stochastic gradient satisfies $g_{t+1} = \nabla \mathcal{R}(\Phi_t) + \epsilon$, while the momentum term behaves as $m_{t+1} = G_t + \epsilon \cdot \frac{t+3}{4}$. Since $\mu_t \approx 1$, these two contributions have very different dependence on ϵ in the update for Φ_t. Because there is only ***one learning rate*** γ controlling both terms, the method is expected to be unstable. This instability was formally established for the PLRF model (Section B) in Theorem 7 of Even et al. (2021).

Finally, we note that there are many equivalent formulations of Nesterov's method. A detailed discussion of these equivalences can be found in Section O.1 and Table 21. Of particular relevance—given its connection to the β_1 parameter in AdamW —is what we refer to as *Exponential Moving Average*

(EMA) Nesterov:

$$p_{t+1} = \mu_{t-1} p_t + (1-\mu_t) g_{t+1} \text{ and } \Phi_{t+1} = \Phi_t - \gamma\big(g_{t+1} + \frac{\mu_t}{1-\mu_{t-1}} \cdot p_{t+1}\big),$$
$$\text{where } p_{t+1} \approx \frac{m_{t+1}}{1-\mu_{t-1}}, g_{t+1} = \nabla \mathcal{L}(\Phi_t, x_{t+1}).$$

This formulation is not exactly equivalent to the previous ones, but becomes asymptotically equivalent as $t \to \infty$. Importantly, all of these variants suffer from the same gradient noise accumulation issue. This formulation corresponds to ADANA with $\beta_2 = 1$.

Generalized Nesterov: Fixing instability of log-momentum and adding a damped term. To mitigate the accumulation of noise from stochastic gradients, we introduce the ***Generalized Nesterov's accelerated method***:

$$m_{t+1} = \mu_{t-1} m_t + g_{t+1} \quad \text{and} \quad \Phi_{t+1} = \Phi_t - \gamma\big(\underbrace{g_{t+1}}_{\text{grad. term}} + \underbrace{\hat{\alpha}_t \cdot m_{t+1}}_{\text{momentum term}}\big)$$
$$\text{where } g_{t+1} = \nabla \mathcal{L}(\Phi_t) \text{ and } \mu_t = \frac{t}{t+3}.$$

Here, $\hat{\alpha}_t$ is an additional learning rate parameter that must be carefully tuned. We present this generalization using the *Extra-gradient Nesterov* formulation, but we also provide the corresponding generalization for the *EMA Nesterov* formulation due to its similarities with ADANA. Related formulations have appeared previously, e.g., in Defazio et al. (2024); Pagliardini et al. (2025); Defazio et al. (2025); Varre and Flammarion (2022); Flammarion and Bach (2015).

	Momentum Update	**Parameter Update**
Standard Nesterov-Extra grad. (102)	$m_{t+1} = \mu_{t-1} m_t + g_{t+1}$	$\Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \mu_t \cdot m_{t+1})$
Generalized Nesterov-Extra grad. (104)	$m_{t+1} = \mu_{t-1} m_t + g_{t+1}$	$\Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \hat{\alpha}_t \cdot m_{t+1})$
Standard Nesterov-EMA (103)	$p_{t+1} = \mu_{t-1} p_t + (1-\mu_{t-1}) g_{t+1}$	$\Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \frac{\mu_t}{1-\mu_{t-1}} \cdot p_{t+1})$
Generalized Nesterov-EMA(108)	$p_{t+1} = \mu_{t-1} p_t + (1-\mu_{t-1}) g_{t+1}$	$\Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \alpha(t) \cdot p_{t+1})$

In both formulations, we replace μ_t in (102) with $\hat{\alpha}_t$, or equivalently replace $\frac{\mu_t}{1-\mu_{t-1}}$ with $\alpha(t)$. Since $\lim_{t\to\infty} \mu_t = 1$, we may treat μ_t as effectively equal to one in the asymptotic regime.

Before proceeding, we examine the effect of the additional hyperparameter $\hat{\alpha}_t$ on Nesterov's accelerated method; a summary is provided in the table below.

Damping factor $\hat{\alpha}_t$	**Damping factor $\alpha(t)$**	**Behavior**
$\hat{\alpha}_t = \mu_t \approx 1$	$\alpha(t) = \frac{\mu_t}{1-\mu_{t-1}} \approx \frac{t}{3}$	Standard Nesterov
$\hat{\alpha}_t < \mu_t$	$\alpha(t) < \frac{\mu_t}{1-\mu_{t-1}} \approx \frac{t}{3}$	**Damped steps (more conservative)**
$\hat{\alpha}_t > \mu_t$	$\alpha(t) > \frac{\mu_t}{1-\mu_{t-1}} \approx \frac{t}{3}$	Amplified steps (more agressive)
$\hat{\alpha}_t = 0$	$\alpha(t) = 0$	No momentum contribution/reduces to SGD

Amplifying the momentum term (i.e., choosing $\alpha(t) \gtrsim \frac{t}{3}$) further increases the disparity between the magnitudes of the gradient and momentum contributions. In particular, setting $\alpha(t) = \frac{\mu_t}{1-\mu_{t-1}}$ recovers standard Nesterov acceleration, which is known to be unstable in the stochastic setting. Since our primary concern is stability—specifically, the accumulation of stochastic gradient noise in the momentum term—we instead seek to ***dampen the momentum contribution*** in the parameter update. This motivates focusing on the regime $\hat{\alpha}_t < \mu_t \approx 1$.

At the same time, excessive damping causes the method to behave like standard SGD, forfeiting acceleration and scaling benefits. This raises the central question:

*Is there a choice of $\alpha(t)$ that achieves both **stability** and **acceleration** at scale?*

The deliberate choice to *dampen* the momentum term—so that the gradient and momentum contributions remain of comparable order in the parameter update rule—is precisely the motivation behind ADANA (Adaptive Damped Nesterov Acceleration).

L.1.3 BENEFIT OF LOG-MOMENTUM: CHOOSING A DAMPING SCHEDULE, $\alpha(t)$, THAT ENABLES ACCELERATION AS MODEL SIZES INCREASE

The main contribution of this section is to motivate the choice of the damping coefficient scaling schedule $\alpha(t)$ in ADANA:

$$\boldsymbol{\alpha(t) = (1+t)^{1-\kappa}}, \tag{79}$$

where κ is a quantity that is tuned, but appears transferable across scales.

Motivation for damping scaling schedule, $\alpha(t)$. The specific form of $\alpha(t)$ in (79) is motivated by a theoretically grounded analysis of DANA on the PLRF model (see Ferbach et al. (2025)). The DANA algorithm (see Alg. 8) updates according to the generalized Nesterov (extra-gradient) form

$$m_{t+1} = \left(1 - \frac{\delta}{t+\delta}\right) m_t + g_t \quad \text{and} \quad \Phi_{t+1} = \Phi_t - \gamma(g_t + \tilde{\gamma}(d)(1+t)^{-\kappa} m_{t+1}).$$

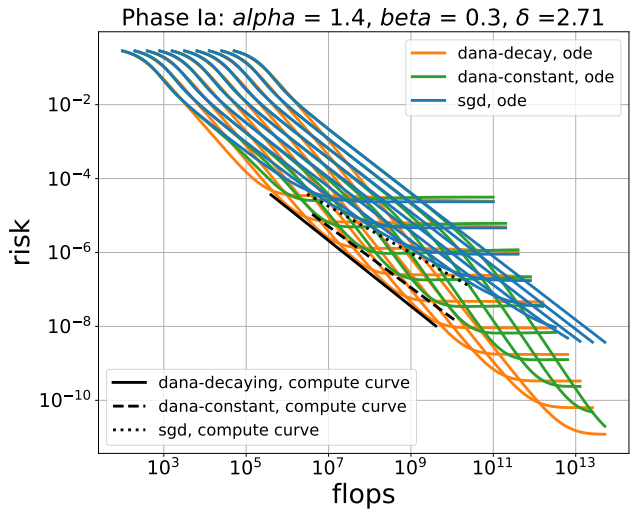
In particular, Ferbach et al. (2025) studies damping schedules of the form $\hat{\alpha}(t;d) \asymp \tilde{\gamma}(d)(1+t)^{-\kappa}$, where $\tilde{\gamma}$ may depend on the parameter dimension d. They establish the following results on a power-law random features (PLRF) model with data generated from a power-law distribution with data exponent ρ and target exponent η, satisfying $2\rho + 2\eta > 0$ and $2\rho > 1$ (see Section B):

- The optimal damping scaling schedule is $\hat{\alpha}(t) \asymp (1+t)^{-1/(2\rho)}$. Moreover, DANA with this schedule outscales SGD under the Chinchilla training regime[12], achieving both acceleration and compute-efficiency gains. This variant is referred to as DANA-DECAYING. Among schedules of the form $\hat{\alpha}_t = \tilde{\gamma}(d)(1+t)^{-\kappa}$, this choice is optimal in the sense that it yields the best loss scaling exponent. See also Yarotsky and Velikanov (2025) for related algorithm.
- A dimension-dependent but iteration-independent schedule $\hat{\alpha}_t \asymp \frac{1}{d}$ is also introduced. DANA with this schedule, referred to as DANA-CONSTANT, outscales SGD for nearly all choices satisfying $2\rho + 2\eta > 1$ and $2\rho > 1$, although it does not scale as favorably as DANA-DECAYING (see also Varre and Flammarion (2022) for related algorithms). This is evidenced in Figure 38a where DANA-decay outscales DANA-constant which in turns outscales SGD in some (ρ, η) region of the PLRF.
- When $\kappa \geq 1$ in $\alpha(t)$, DANA-DECAYING does not accelerate or outscale over SGD. In particular, Ferbach et al. (2025) show that $\kappa = 1$, on the PLRF, was equivalent to doing Schedule Free-SGD Defazio et al. (2024). This suggest that we should not expect ADANA to outscale (i.e., improve as the model size grows) if $\kappa = 1$. Figure 38b shows that Schedule-Free does indeed not outscale SGD on the PLRF.

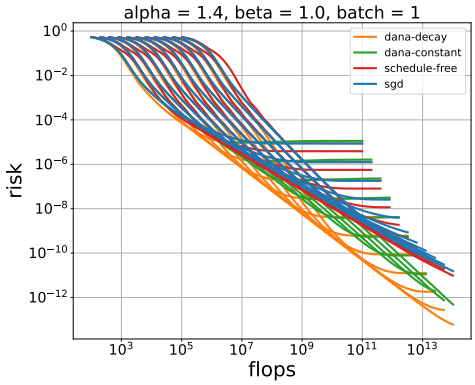
Motivated by these findings on the PLRF model—which is widely studied due to its tractability and its ability to mimic scaling properties observed in LLMs—we adopt $\boldsymbol{\alpha(t) = (1+t)^{1-\kappa}}$ ***in* ADANA**. The apparent discrepancy by a factor of $(1+t)$ relative to DANA arises precisely from the t-scaling that appears when transitioning from the generalized extra-gradient Nesterov formulation (104), (DANA), to the generalized EMA Nesterov formulation (108), (ADANA).

Choosing κ in the damping schedule: Practical guidance. On the PLRF model, the optimal value of κ depends on spectral properties of the data, in particular the power-law exponent. Consequently, theory predicts that κ should remain unchanged as the problem dimension grows, making it a transferable hyperparameter across scales. In practice, the data power-law exponent is unknown and κ must be tuned. Empirically (see Figure 4 & Figure 5a), we find that κ is indeed transferable across model sizes and relatively easy to tune. In our experiments on transformer architectures trained on FineWeb data, effective values of κ lie in the range **0.8-0.9**, with **0.85** performing best across all scales considered in this work. Moreover, κ appears to transfer well across architectures, performing robustly with minimal tuning when moving from Enoki models to Qwen.

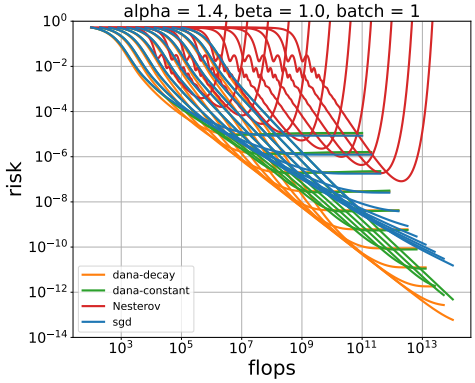
[12] A *training regime* is defined by a scaling $t \asymp d^{\ell}$, $\ell > 0$, relating the number of iterations (or samples) to the parameter dimension. Examples include the *proportional regime* ($t \asymp d$) and the *compute-optimal regime*, in which ℓ is chosen to minimize loss under a fixed compute budget. Suppose the loss obeys the scaling law $\mathcal{R}(t,d) \overset{\text{def}}{=} \mathcal{R}(\theta_t, d) \asymp t^{-\sigma} + d^{-\tau}$, and $t \asymp d^{\ell}$. Then $\mathcal{R}(d^{\ell}, d) \asymp d^{-\min \ell\sigma, \tau}$. We refer to the absolute exponent on d as the *loss exponent*. For a given training regime, one algorithm *outscales* another if it achieves a larger loss exponent.



(a) Figure 7 in Ferbach et al. (2025). Dana-decay and Dana-constant have faster compute-optimal scaling laws than SGD showing significant and growing improvement at the larger scales.



(b) Figure 10d in Ferbach et al. (2025). Schedule-free has a correspondence with using $\kappa = 1$ which has the same scaling laws as SGD.



(c) Figure 10f in Ferbach et al. (2025). Original Nesterov schedules (full-batch) are unstable in the stochastic setting. They correspond to using $\kappa = 0$, i.e. no damping.

Figure 38: Figures from Ferbach et al. (2025). Comparison of DANA-decay, DANA-constant, SGD, Schedule-Free SGD and Nesterov on the PLRF model. Here α, β correspond to respectively ρ, η. The risk is shown as a function of compute in flops, computed as $6 \times$ # iterations $\times$ dimension.

Summary: Benefits of log-momentum and damping schedule.

Recommended schedule (used in ADANA):

$$\beta_1(t) = \delta(\delta + t)^{-1} \quad \text{and} \quad \alpha(t) = (1 + t)^{1-\kappa}.$$

Motivation. This schedule is motivated by theoretical analyses of damped Nesterov-type methods on the PLRF model Ferbach et al. (2025), where sublinear damping of the momentum term helps control stochastic noise accumulation while retaining acceleration-like behavior.

Practical guidance for choosing κ.

- $\kappa < 1$ is essential: Small values of κ strengthen the momentum contribution and may improve acceleration, but can increase sensitivity to stochastic gradient noise. Choosing a good κ is important, but it is easily tunable on smaller models; a range of κ give similar performance with no dimensioning returns as model sizes grow.
- $\kappa \geq 1$ should lead to no performance gains: larger values over-damp the method and recover SGD-like behavior on the PLRF.
- Empirically, κ appears relatively *stable* and *transferable across scales*: in our experiments, values in the range $\kappa \in [0.75, 0.9]$ performed well across model sizes, with $\kappa \approx 0.85$ serving as a robust default. This range of κ's all give similar performance and compute gains persist (and do not diminish) as the model size grows on LLMs.

Interpretation. Adding log-momentum of the form $\beta_1(t) = \delta(\delta + t)^{-1}$ can help improve performance of an algorithm at scale, but one must also include a damping schedule for stability. The damping schedule $\alpha(t) = (1 + t)^{1-\kappa}$ provides a practical interpolation between unstable Nesterov acceleration and overly conservative SGD, offering a principled way to stabilize log-momentum while preserving potential compute-efficiency gains in large-scale training.

L.2 SCHEDULING THE 2ND MOMENT BUFFER

We have seen that introducing log-momentum together with an appropriate damping schedule can accelerate stochastic momentum methods (at least on PLRF). We now turn to incorporating this idea into ADAM. In particular, our objective is to determine how the second-moment parameter β_2 should be scheduled when the first-moment parameter β_1 follows a log-time schedule.

Goal. *Assuming β_1 uses a log-time schedule and a damping coefficient $\alpha(t)$, choose β_2 so as to achieve compute-efficiency gains in* ADAM *across the scaling ladder.*

In doing so, we will also highlight several potential pathologies that arise under different choices of β_2.

A naive approach to incorporating log-momentum into ADAM is to set $\beta_1 = 1 - \delta(\delta + t)^{-1}$ while keeping β_2 fixed at an absolute constant in $(0, 1)$. However, this choice is empirically unstable even in simple settings such as the Mixture-of-Experts PLRF model (Section B). As we will later show, it is also provably unstable in the sparse-gradient regime (see Section L.2.1). We observe similar instability on the Enoki scaling ladder when using a constant β_2 together with a log-time β_1 (see Figure 3).

In practice, it is common to choose $\beta_2 > \beta_1$ Orvieto and Gower (2025), which fails under the naive log-momentum implementation above. This naturally suggests a second candidate: setting β_2 to follow the same log-time schedule as β_1. We refer to this variant as LOG-ADAMW (Alg. 3), defined by

$$\begin{aligned} m_{t+1} &= \left(1 - \frac{\delta}{t+\delta}\right) m_t + \frac{\delta}{t+\delta} \cdot g_{t+1} \\ v_{t+1} &= \left(1 - \frac{\delta}{t+\delta}\right) v_t + \frac{\delta}{t+\delta} \cdot g_{t+1}^2 \\ \theta_{t+1} &= \theta_t - \gamma \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon}. \end{aligned} \tag{80}$$

Unfortunately, both variants—ADAM with log-time β_1 and fixed β_2, as well as LOG-ADAMW with log-time schedules for both moments—are unstable in the presence of sparse gradients. To make this instability precise, we next introduce a simple necessary condition for convergence. All proofs of theorems in this section can be found in Section P.

L.2.1 SPARSE GRADIENTS AND A NECESSARY CONDITION FOR CONVERGENCE

One of the advantages of ADAM (and AdaGrad Duchi et al. (2011)), as highlighted in the original ADAM paper by Kingma and Ba (2015), is its ability to handle sparse gradients. Sparsity in stochastic gradients, however, can also cause optimization algorithms to diverge. As we will show, this issue arises for ADAM when using a logarithmic-time schedule for β_1, and it motivates placing β_2 on a logarithmic-time schedule as well. In this section, we develop the theoretical foundations explaining why log-time β_1 fails in the presence of sparse gradients and why choosing β_2 in log-time can partially—but not fully—mitigate this problem.

To model sparse gradients, consider a stochastic gradient oracle $\mathcal{O}$ which, for any parameter $\theta \in \mathbb{R}^d$, outputs a stochastic gradient $\mathcal{G} \sim \mathcal{O}(\theta)$ satisfying, for some constant $M > 0$,

$$\mathbb{E}[\|\mathcal{G}\|^2] \le M \quad \text{and} \quad \mathbb{E}[\mathcal{G}] = \nabla \mathcal{R}(\theta).$$

We induce sparsity via a *sparse oracle* $\mathcal{O}_p$, which generates sparse stochastic gradients $\mathcal{G}' \sim \mathcal{O}_p$ according to

$$\mathcal{G}' = B \cdot \mathcal{G} \quad \text{where} \quad \mathcal{G} \sim \mathcal{O}(\theta) \quad \text{and} \quad B \sim \text{Bernoulli}(p).$$

Here $B \sim \text{Bernoulli}(p)$ is a Bernoulli random variable with probability p of being 1 and probability $(1-p)$ of being 0. Therefore, the value of $p \in [0,1]$, represents the probability that the stochastic gradient is *not sparse*.

Now let us suppose that our gradients g_t in ADAM are generated from our sparse oracle $\mathcal{O}_p$ and we perform the ADAM parameter update, $Y_t \overset{\text{def}}{=} \frac{m_t}{\sqrt{v_t}+\epsilon}$. Even though the gradients g_t may be zero due to sparsity, because ADAM has cumulative 1st/2nd moments m_t (and v_t), the parameters still evolve and change. If sparsity exists and the 1st moment accumulates enough (such as in log-momentum), then the updates θ_t will not decrease the loss and $\|\theta_t\|$ can grow and grow causing non-convergence.

To make this behavior explicit, let T_t be the times at which nonzero gradients occur and define the cumulative sum of ADAM updates between those times

$$Z_\ell \overset{\text{def}}{=} \sum_{t=T_\ell}^{T_{\ell+1}-1} Y_t, \quad \ell \ge 1 \quad \text{where } Y_t \overset{\text{def}}{=} \frac{m_t}{\sqrt{v_t}+\epsilon} \text{ is the ADAM update.} \tag{81}$$

Theorem L.1. *Suppose $\beta_1(t) = 1 - \delta/(t+1)$ for some $\delta > 1$ and let $\beta_2(t) \in (0,1)$ be **any** monotone sequence. Let the gradients $g_t \sim \mathcal{O}_p$ be generated by our sparse gradient oracle and define Z_j as in (81). Then the family of random variables $\{Z_j : j \ge 1, p \in (0,1), \epsilon > 0\}$ is unbounded, i.e.,*

$$\limsup_{p \to 0} \sup_{j \ge 1, \epsilon > 0} \mathbb{E}[\|Z_j\|] = \infty.$$

In fact, it follows that along a subsequence of $p \to 0$, $\|Z_2\| \to \infty$.

Intuitively, this result says that as the probability of sparse gradients increase, the likelihood that the updates of LONG ADAM, with any monotonically decreasing $\beta_2 \in (0,1)$ sequence, can grow without bound. In particular, if sparse gradients are sufficiently common (small p), the iterates in LOG-ADAMW may diverge before one sees another nonzero gradient. This immediately yields the following corollary.

Corollary L.2 (Divergence of LOG-ADAMW for Lipschitz functions)**.** *Suppose $\mathcal{R} : \mathbb{R}^d \to \mathbb{R}$ is Lipschitz. Let the gradients $g_t \sim \mathcal{O}_p$ be generated by the sparse oracle and $\{\theta_t\}$ be iterates generated by Alg. 3 (or with $\beta_2 \in (0,1)$ which is monotonically decreasing sequence) using these sparse gradients. Let T_2 be the time at which the second nonzero gradient occurs. Then there is a sequence of $p \to 0$ so that*

$$\|\theta_{T_2}\| \to \infty.$$

Hence ADAM with $\beta_1(t) = 1 - \delta(1+t)^{-1}$ is essentially uniformly bad; on *every* problem with a sufficiently sparse oracle, the iterates can become arbitrarily large. In particular, LOG-ADAMW is unstable for any Lipschitz loss whose minimizer does not lie at infinity.

Necessary condition for convergence. As a consequence of Corollary L.2 (and Theorem L.1), we get a necessary condition for convergence of ADAM:

$$\text{(Necessary condition for convergence)} \quad \mathbb{E}[\|Z_j\|] \quad \text{is bounded.} \tag{82}$$

L.2.2 INTUITION FOR THEOREM L.1 AND EXAMPLES OF BAD CHOICES FOR β_2 WITH LOG-MOMENTUM.

We now provide intuition for the divergence result and examine several problematic choices of β_2 in the presence of log-momentum. For simplicity, we apply all arguments entry-wise and assume $\epsilon > 0$ is negligible.

Unraveling the ADAM updates with time-varying β_1 and β_2 yields the approximation

$$Y_t = \frac{m_t}{\sqrt{v_t}+\epsilon} \approx \sum_{s=1}^{t} \frac{(1-\beta_1(s))\big(\prod_{r=s+1}^{t}\beta_1(r)\big)g_s}{\sqrt{\sum_{s=1}^{t}(1-\beta_2(s))\big(\prod_{r=s+1}^{t}\beta_2(r)\big)(g_s^2)}}. \tag{83}$$

ADAM with constant β_1 and β_2. Let us first understand the case when β_1 and β_2 are fixed constants (independent of time). In this case, $\prod_{r=s+1}^{t}\beta_1 = \beta_1^{t-s}$ and similarly for $\prod_{r=s+1}^{t}\beta_2 = \beta_2^{t-s}$. Thus,

$$Y_t \approx \frac{1-\beta_1}{\sqrt{1-\beta_2}} \times \sum_{s=1}^{t} \frac{\beta_1^{t-s} g_s}{\sqrt{\sum_{s=1}^{t}\beta_2^{t-s} g_s^2}}.$$

Because of the constant β_1 and β_2, the tail history of g_s gets small exponentially fast and thus, only the most recent gradients really matter. The small batch means that the gradients do not change very much over this recent history. Thus, we can replace this changing g_s with the last gradient g_t, which implies that $g_t/\sqrt{g_t^2} \le 1$. Now we need some summability of the β_1, β_2 to ensure that the update is bounded. As a result, we get the following.

Theorem L.3 (ADAM with constant β_1, β_2 satisfies necessary condition)**.** *If $\beta_1, \beta_2 \in (0,1)$ are fixed constants independent of t and $\beta_1^2 < \beta_2$, the family of random variables $\{Z_\ell : \ell \ge 1, p \in (0,1), \epsilon > 0\}$ satisfies the uniform first-moment bound*

$$\mathbb{E}[|Z_\ell|] \le \frac{1-\beta_1}{(\sqrt{1-\beta_2})\big(1-\frac{\beta_1}{\sqrt{\beta_2}}\big)^2}.$$

While this does not guarantee convergence, it explains why standard ADAM handles sparse gradients reasonably well, consistent with Kingma and Ba (2015). However, constant β_1 does not yield the scaling-law improvements associated with log-momentum.

Bad choices for β_2 with log-momentum. Let us now suppose that $\beta_1(t) = (1-\delta/(t+1))$, which implies that $\prod_{r=s+1}^{t}(1-\delta/(r+1)) \asymp \big(\frac{s}{t}\big)^{\delta}$. To understand what happens in this case, let us focus only on what happens between the first non-zero gradient and the 2nd non-zero gradient (i.e., $t \in [T_1, T_2-1]$). For convenience, we can assume that $T \overset{\text{def}}{=} T_1$ and the time between the first and 2nd non-zero gradients is the same as the time to get the first non-zero gradient, i.e., $T = T_1 \asymp T_2 - T_1$. In particular, we expect $T = 1/p$ where p is the probability of a non-spare gradient. If sparse gradients happen, then T is large.

Fix $t \in [T_1, T_2-1] = [T, M\cdot T - 1]$ where the constant $M > 1$. Since $t \in [T, MT-1]$, we know that only one non-zero gradient has occurred from the collection of gradients $\{g_s\}_{s=1}^{t}$. Let X_T be this non-zero gradient. Thus, we have that

$$m_t = \sum_{s=1}^{t}(1-\beta_1(s))\Big(\prod_{r=s+1}^{t}\beta_1(r)\Big)g_s = \tfrac{1}{T}\times X_T \times \big(\tfrac{T}{t}\big)^{\delta}$$

$$v_t = \sum_{s=1}^{t}(1-\beta_2(s))\Big(\prod_{r=s+1}^{t}\beta_2(r)\Big)g_s^2 = (X_T)^2(1-\beta_2(T))\prod_{r=T+1}^{t}\beta_2(r)$$

$$\Rightarrow \quad Y_t = \frac{\frac{1}{T}\times X_T \times \big(\frac{T}{t}\big)^{\delta}}{|X_T|\sqrt{(1-\beta_2(T))\prod_{T+1}^{t}\beta_2(r)}} \quad \Rightarrow \quad |Z_1| = \sum_{t=T}^{MT-1}\frac{\frac{1}{T}\times\big(\frac{T}{t}\big)^{\delta}}{\sqrt{(1-\beta_2(T))\prod_{T+1}^{t}\beta_2(r)}}.$$

It follows by a Riemann approximation (provided $\delta > 1$) that for large T (or sparse gradients), $\frac{1}{T}\sum_{t=T}^{MT-1}\left(\frac{T}{t}\right)^{\delta} \approx \int_0^{M-1}(1+u)^{-\delta}\,\mathrm{d}u < \infty$. Since the numerator of $|Z_1|$ is order 1, it means that if $|Z_1|$ is to be bounded, then we need the denominator to be also order 1. Before continuing, we discuss below two choices for β_2 which fail the necessary condition 82.

Bad choice 1: Log-momentum and constant β_2**.** Let us suppose that β_2 is a constant. Then we see that $\prod_{r=T+1}^{t}\beta_2 = \beta_2^{t-T}$. Since $\beta_2^{t-T} \to 0$ as $t \to \infty$, the denominator of $|Z_1|$ is going to 0, causing $|Z_1|$ to be infinity. Thus ADAM with $\beta_1(t) = 1 - \delta(t+1)^{-1}$ and β_2 constant is divergent, i.e., doesn't satisfy our necessary condition in (82).

Bad choice 2: Log-momentum and $\beta_2 = 1 - c/(t+1)^2$**.** Now suppose we take an aggressive $\beta_2(t) = 1 - c/(t+1)^2$. Then we see that

$$(1-\beta_2(T))\prod_{r=T+1}^{t}\beta_2(r) \asymp \frac{1}{T^2}, \quad \text{as we know} \prod_{r=T+1}^{t}\beta_2(r) = \mathcal{O}(1).$$

Putting this together gives that $|Z_1| = \sum_{t=T}^{MT-1}\left(\frac{T}{t}\right)^{\delta}$. To make this summation convergent for large T, requires a $1/T$. Multiplying and dividing by $1/T$ results in

$$|Z_1| = T \times \frac{1}{T}\sum_{t=T}^{MT-1}\left(\frac{T}{t}\right)^{\delta} \asymp T \asymp \frac{1}{p}.$$

This again blows up as $p \to 0$ and ADAM fails to converge with this choice of β_2.

L.2.3 BEST CHOICE FOR β_2 IN THE SPARSE GRADIENT SETTING, $\beta_2(t) = 1 - \delta(1+t)^{-1}$

These examples motivate choosing

$$\beta_2 = 1 - \delta/(t+1).$$

We provide the follow result in this case.

Theorem L.4. *If* $\beta_1(t) = \beta_2(t) = (1 - \frac{\delta}{t})$ *and* $\delta > 2$*, the family of random variables* $\{Z_\ell : \ell \geq 1, p \in (0,1), \epsilon > 0\}$ *satisfies the first-moment bound,*

$$\mathbb{E}[|Z_\ell|] \leq \frac{C}{\sqrt{p}}, \quad \textit{where } C > 0 \textit{ is a constant.}$$

This bound is substantially better than the $1/p$ divergence observed with more aggressive β_2 schedules, making this choice the *least sensitive* to sparsity among log-time schedules.

However, even this choice does not fully resolve the issue: both LOG-ADAMW and ADANA may still diverge under sparse gradients. This theorem suggests that we still need to correct for sparsity by multiplying our updates by $\sqrt{p}$. Since the probability p is unknown, we need to construct an online estimate of it. This motivates the development of DANA-STAR, a variant of ADANA explicitly designed to correct for sparsity by estimating the unknown probability p online via a quantity denoted τ. We discuss this and other variants of ADANA in Section N.

Summary: Log-time β_2 **and sparsity.** When using log-time momentum $\beta_1(t) = 1 - \delta/(t+1)$, the choice of the second-moment schedule β_2 plays a critical role in stability under sparse gradients. Among log-time schedules, setting

$$\beta_2(t) = 1 - \delta/(t+1)$$

is the *least sensitive to sparsity*: it yields the weakest divergence rate of the accumulated updates between nonzero gradients among potential β_2 schedules. While this choice does not fully eliminate divergence under extreme sparsity, it substantially mitigates the instability observed with constant or faster-decaying β_2 schedules.

L.3 Explanation for the log-time

We finish this section by providing an explanation for the term *log-time*.

Consider the AdamW updates (with indepdent weight-decay) on time $\tau = \log(\delta + t)$, with constant learning rate γ^* (the schedule $\gamma(t) = 1$), $\beta_1 = \beta_2 = 1 - \delta$, $\lambda = \omega$:

$$\begin{aligned} \textit{(mom.)} \quad & m_{\tau+1} = (1-\delta)m_\tau + \delta g_{\tau+1}, \\ \textit{(2}^{nd}\textit{ moment)} \quad & v_{\tau+1} = (1-\delta)v_\tau + \delta g_{\tau+1}^2, \\ \textit{(param.)} \quad & \theta_{\tau+1} = \theta_\tau - \gamma^* \cdot \frac{m_{\tau+1}}{\sqrt{v_{\tau+1}} + \epsilon} - \omega \cdot \theta_\tau. \end{aligned} \tag{84}$$

We can rewrite these updates with $\Delta\tau = 1$ as

$$\begin{aligned} \textit{(mom.)} \quad & (m_{\tau+1} - m_\tau)\Delta\tau = \delta(g_{\tau+1} - m_\tau)\Delta\tau, \\ \textit{(2}^{nd}\textit{ moment)} \quad & (v_{\tau+1} - v_\tau)\Delta\tau = \delta(g_{\tau+1}^2 - v_\tau)\Delta\tau, \\ \textit{(param.)} \quad & (\theta_{\tau+1} - \theta_\tau)\Delta\tau = -\left(\gamma^* \frac{m_{\tau+1}}{\sqrt{v_{\tau+1}} + \epsilon} - \omega\theta_\tau\right)\Delta\tau. \end{aligned} \tag{85}$$

Now we can do the change of variable from τ to t and replace $\Delta\tau \approx \frac{\Delta t}{\delta + t}$ to obtain

$$\begin{aligned} \textit{(mom.)} \quad & (m_{t+1} - m_t)\frac{\Delta t}{\delta + t} \approx \delta(g_{t+1} - m_t)\frac{\Delta t}{\delta + t}, \\ \textit{(2}^{nd}\textit{ moment)} \quad & (v_{t+1} - v_t)\frac{\Delta t}{\delta + t} \approx \delta(g_{t+1}^2 - v_t)\frac{\Delta t}{\delta + t}, \\ \textit{(param.)} \quad & (\theta_{t+1} - \theta_t)\frac{\Delta t}{\delta + t} \approx -\left(\gamma^* \cdot \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} - \omega \cdot \theta_t\right)\frac{\Delta t}{\delta + t}. \end{aligned} \tag{86}$$

Finally we recover

$$\begin{aligned} \textit{(mom.)} \quad & m_{t+1} \approx (1 - \frac{\delta}{\delta + t})m_t + \frac{\delta}{\delta + t} g_{t+1}, \\ \textit{(2}^{nd}\textit{ moment)} \quad & v_{t+1} \approx (1 - \frac{\delta}{\delta + t})v_t + \frac{\delta}{\delta + t} g_{t+1}^2, \\ \textit{(param.)} \quad & \theta_{t+1} \approx \theta_t - \frac{\gamma^*}{\delta + t} \cdot \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} - \frac{\omega}{\delta + t} \cdot \theta_t. \end{aligned} \tag{87}$$

Note the additional factor on the learning rate $\frac{\gamma^*}{\delta+t}$ which in practice we ignored. Indeed, intuitively we want to apply logarithmic time updates only on the averaging updates (weight-decay and momentum) and not the gradient descent part of the update. This is equivalent to adding a factor $\gamma^* \times \tau$ to the learning rate update in logarithmic time.

M The Hilberg Hypothesis and Logarithmic Time

In this section, we provide a speculative but motivated connection between the Hilberg exponent from information theory and the logarithmic time schedules used in AdaNa. The central idea is that the rate-limiting factor in language model training is feature learning on power-law distributed data, and the Hilberg exponent characterizes precisely how difficult this learning problem is.

M.1 The Hilberg Exponent

Following Shannon (1951) we model natural language as a stationary stochastic process $(X_t)_{t\in\mathbb{Z}}$ taking values in a finite alphabet $\mathcal{A}$. The *block entropy* $H(X_1^n) = H(X_1, \ldots, X_n)$ measures the total uncertainty in n consecutive symbols, and the *Shannon entropy rate* is defined as $h =$

$\lim_{n\to\infty} H(X_1^n)/n = \lim_{n\to\infty} H(X_n \mid X_1^{n-1})$, where the second equality holds for stationary processes (Shannon, 1948). The entropy rate h captures the irreducible per-symbol uncertainty that remains even when the predictor has access to unbounded context.

Shannon (1951) estimated the entropy of English through human guessing experiments at various context lengths, observing that per-symbol uncertainty decreases substantially as more context is provided. Hilberg (1990) reanalyzed Shannon's data and proposed that block entropy grows sublinearly as $H(X_1^n) \sim n^\beta$ with $\beta \approx 0.5$, which would imply an entropy rate of $h = 0$. This original conjecture is now considered too strong. Following Crutchfield and Feldman (2003) and Ebeling and Nicolis (1991), the modern *relaxed Hilberg hypothesis* retains a positive entropy rate and posits that

$$H(X_1^n) = h \cdot n + A \cdot n^\beta + o(n^\beta), \tag{88}$$

so that the per-symbol encoding rate—the bits per symbol required by an optimal compressor with access to n symbols of context—decays as $r(n) = H(X_1^n)/n = h + An^{\beta-1} + o(n^{\beta-1})$ (Dębowski, 2015). The *Hilberg exponent* $\beta \in (0, 1)$ thus characterizes how quickly compression rates converge to the entropy rate as context grows.

Takahira et al. (2016) measured β empirically using PPM compression (Cleary and Witten, 1984) on corpora up to 7.8 GB across six languages (English, French, German, Japanese, Chinese, and Korean). They introduced a stretched exponential ansatz $r(n) = \exp(An^{\beta-1} + h')$ that better fits the empirical compression curves, and found $\beta \approx 0.88$ with remarkably small cross-language variance (SD $= 0.034$). This consistency suggests that the Hilberg exponent is a *language universal*, independent of the particular language or writing system.

The universality of β suggests that while entropy rate h measures how hard it is to *predict* text, β measures how hard it is to *learn* to predict text—and all human languages appear equally hard to learn.

M.2 The Hilberg Hypothesis: Systematic Noise in Logarithmic Time

To connect the Hilberg exponent to optimization, we work in *logarithmic time* $\tau = \log t$. This change of variables is natural for two reasons: (1) the momentum schedule $\beta_1(t) = 1 - \delta/t$ becomes a constant effective momentum in log-time, and (2) the Hilberg exponent describes how information accumulates across scales, where each unit of log-time corresponds to one scale.

We hypothesize that within each unit of logarithmic time (i.e., from t to $2t$), the model is learning features at a particular frequency scale. The gradient noise during this period is *systematic*, not independent: the model lacks certain features needed to predict tokens at this scale, and this manifests as correlated errors throughout the log-time window. The model makes the same mistakes repeatedly because it is missing the same features.

Concretely, let $\sigma(\tau)$ denote the scale of the systematic error per unit of log-time. Based on the Hilberg exponent, this error decays as

$$\sigma(\tau) \sim t^{\beta-1} = e^{\tau(\beta-1)}, \tag{89}$$

where $\beta \approx 0.88$ is the Hilberg exponent. In physical time, this is $t^{-0.12}$—a slow decay reflecting the difficulty of learning rare features.

This model differs fundamentally from standard stochastic optimization assumptions. The noise is not IID across iterations; rather, it is perfectly correlated within each log-time unit (same missing features) but approximately independent across log-time units (different frequency scales). This correlation structure means the noise does not average out within a log-time window.

Connection to damped Nesterov acceleration. Consider the damping schedule $\alpha(t) = (1+t)^{1-\kappa}$ used in ADANA. In logarithmic time, this becomes $\alpha(\tau) = e^{\tau(1-\kappa)}$. For $\kappa = 0$, the damping grows exponentially in log-time (full Nesterov acceleration); for $\kappa = 1$, the damping is constant (no acceleration).

The momentum buffer in log-time spans $O(1)$ units, accumulating the systematic errors from recent scales. Since the noise is correlated within each log-time unit, we cannot appeal to variance reduction from averaging. Instead, the contribution to the optimizer state per log-time unit has scale

$$\alpha(\tau) \cdot \sigma(\tau) \sim e^{\tau(1-\kappa)} \cdot e^{\tau(\beta-1)} = e^{\tau(\beta-\kappa)}. \tag{90}$$

For stability, this contribution should remain $O(1)$ as $\tau \to \infty$, which requires

$$\beta - \kappa \leq 0, \quad \text{i.e.,} \quad \boxed{\kappa \geq \beta}. \tag{91}$$

This is the quantitative statement of the Hilberg hypothesis: the damping exponent κ in ADANA should be at least as large as the Hilberg exponent β. With $\beta \approx 0.88$, this predicts $\kappa \geq 0.88$, and the optimal choice $\kappa = \beta$ exactly balances acceleration against the systematic noise.

Interpretation. The spectral dimension parameter κ encodes an assumption about the noise structure of the learning problem. When $\kappa = 0$, the algorithm assumes deterministic gradients and uses full acceleration. When $\kappa = 1$, the algorithm assumes persistent noise and forgoes acceleration entirely. The Hilberg hypothesis suggests that for natural language, the noise decays as $t^{\beta-1}$ with $\beta \approx 0.88$, and thus κ should be chosen close to β.

This provides a principled justification for why $\kappa \in [0.8, 0.9]$ works well across model scales: it reflects a fundamental property of language data (the Hilberg exponent) rather than a hyperparameter to be tuned per architecture.

M.3 DISCUSSION

The connection between the Hilberg exponent and the power-law damping factor that we use in combination with logarithmic time scheduling remains speculative and requires further theoretical development. However, we believe this perspective is valuable for several reasons:

1. It provides a principled explanation for why κ should be approximately constant across scales—it reflects properties of language data, not model architecture.
2. It explains why κ should be chosen close to the Hilberg exponent: our experiments use $\kappa \in [0.8, 0.9]$, which is consistent with $\beta \approx 0.88$.
3. It suggests that the dominant noise in language model training is less the standard sampling variance but rather the variance due to information-theoretic limits on how well the learned features can explain the next token.

N FAILURE MODES AND HARDENED VARIANTS OF ADANA: FULL DETAILS

This appendix provides complete details on the DANA-STAR, DANA-MK4, and DANA-STAR-MK4 algorithms that address the failure modes of logarithmic-time momentum under sparse gradients and inhomogeneous spectral dimensions.

N.1 DANA-STAR: ACCOUNTING FOR TIME SPARSITY

To address time sparsity, we introduce DANA-STAR, which incorporates a probability estimator τ that tracks how frequently each parameter receives meaningful updates. The key insight is that the effective time $t_{\texttt{eff}}$ experienced by a parameter should scale with the fraction of iterations in which it receives non-trivial gradient information.

Probability estimator, τ. We define an instantaneous estimate of the 'presence' of an update of a parameter

$$\tau_{\texttt{update}} = \frac{|g_{t+1}|}{|g_{t+1}| + \sqrt{v_{t+1}} + \epsilon}. \tag{92}$$

This quantity measures the magnitude of the current gradient relative to the accumulated second moment. When $|g| \gg \sqrt{v}$, we have $\tau_{\texttt{update}} \approx 1$, indicating that the parameter is receiving consistent updates. Conversely, when $|g| \ll \sqrt{v}$, we have $\tau_{\texttt{update}} \approx 0$, indicating sparse or rare updates. We now collect a time average of the presence of an update, to produce an estimate for the *probability* of an update.

$$\tau_{t+1} = \left(1 - \frac{\delta}{t+1}\right) \tau_t + \frac{\delta}{t+1} \tau_{\texttt{update}}. \tag{93}$$

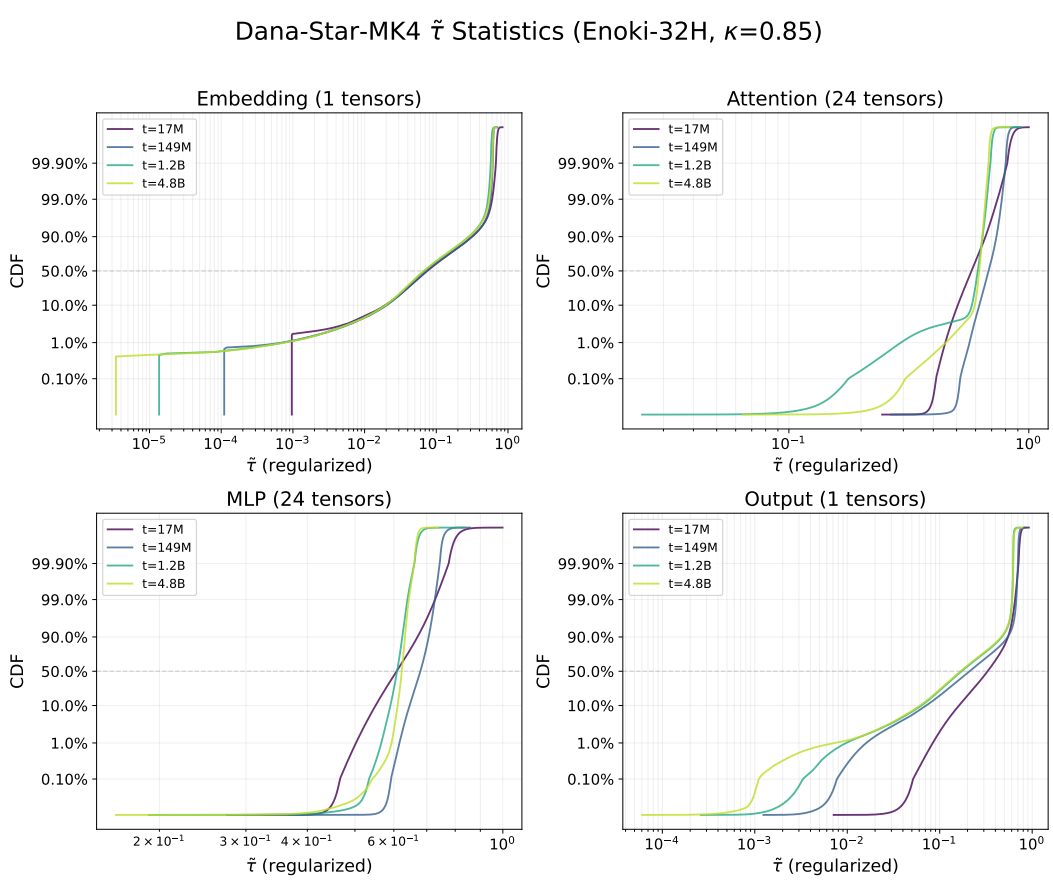


Figure 39: **Distribution of $\tilde{\tau}$ across parameter types.** Cumulative distribution of the regularized probability estimator $\tilde{\tau}$ for a 32-head Enoki model with $\kappa = 0.85$. Each panel shows a different parameter type: embedding (wte), attention, MLP, and output (lm_head). Curves show the distribution at different training stages from early (dark) to late (light). The embedding and output layers exhibit substantially lower τ values due to token-level sparsity, while attention and MLP layers have more concentrated distributions near $\tau \approx 1$.

Effective time and transformed probability. The effective time $t_{\texttt{eff}} = \max\{t \cdot \tau_{t+1}, 1\}$ rescales the iteration count by the update probability, ensuring that rarely-updated parameters do not accumulate excessive momentum. For frequently-updated parameters, $t_{\texttt{eff}} \approx t$ and DANA-STAR behaves like standard ADANA. For rarely-updated parameters, $t_{\texttt{eff}} \ll t$, reducing the momentum contribution.

We also define a clipped and transformed version of τ for use in the gradient scaling:

$$\tau_{\texttt{clip}} = \min\{\tau_{t+1}, 0.5\}, \quad \tilde{\tau}_{t+1} = \max\left\{\frac{\tau_{\texttt{clip}}}{1-\tau_{\texttt{clip}}}, \frac{1}{1+t}\right\}. \tag{94}$$

The clipping at 0.5 prevents numerical instability when τ approaches 1, and the floor at $1/(1+t)$ ensures the scaling remains well-behaved throughout training; note that this strategy can never accurately estimate a probability at time t when that probability is less than $1/(t+1)$. The transformed quantity $\tilde{\tau}_{t+1}$ enters the parameter update through a $\sqrt{\tilde{\tau}}$ scaling factor that modulates both the gradient and momentum terms. The transformation $\tau/(1-\tau)$ just corrects the clipped probability estimate to be 1 for $\tau = 1/2$, and could be ignored as a reasonable optimization of the algorithm.

The full DANA-STAR algorithm is presented in Alg 9.

Visualizing the τ distribution across layers. Figure 39 shows the cumulative distribution of the regularized $\tilde{\tau}$ values across different parameter types during training. The four panels display key

architectural components: the embedding layer (wte), attention weights, MLP weights, and the output projection (lm_head). Each curve shows the distribution at a different training stage, from early (dark) to late (light) training.

Several patterns emerge. First, the embedding and output projection layers exhibit substantially lower τ values compared to the dense attention and MLP layers, reflecting the inherent sparsity in token-level gradients due to Zipfian token frequencies. Second, the distributions shift rightward over training as the τ estimator accumulates more accurate statistics. Third, the heavy tails at low τ values (visible as the rapid decrease below 1%) indicate that a significant fraction of parameters receive sparse updates throughout training, justifying the need for the effective time scaling $t_{\texttt{eff}} = t \cdot \tau$.

N.2 DANA-STAR-MK4/ DANA-MK4: ADAPTING THE EXPONENT DURING TRAINING

While DANA-STAR addresses time sparsity, it still requires setting the exponent κ correctly. DANA-STAR-MK4 introduces an additional mechanism to adapt the effective exponent during training through SNR-based clipping. The key modification is that instead of directly using the momentum term scaled by $(1 + t_{\texttt{eff}})^{1-\kappa}$, we clip the effective scaling factor based on a signal-to-noise ratio estimate.

SNR-based clipping. Define the normalized momentum factor as

$$\texttt{norm} = \frac{\sqrt{\tilde{\tau}_{t+1}}}{\sqrt{v_{t+1}} + \epsilon}, \quad \texttt{mfac} = \frac{|m_{t+1}| \cdot \texttt{norm}}{\tilde{\tau}_{t+1}}. \tag{95}$$

The quantity $\texttt{mfac}$ measures the magnitude of the momentum relative to the second moment, serving as a proxy for the signal-to-noise ratio. We then define the clipped scaling factor

$$\alpha_{\texttt{fac}} = \min\left\{t_{\texttt{eff}}^{1-\kappa} \cdot \texttt{mfac}, \texttt{clipsnr}\right\}, \tag{96}$$

where $\texttt{clipsnr}$ is a hyperparameter controlling the maximum allowed scaling. This clipping prevents the momentum term from dominating when the gradient signal is weak relative to variance, providing automatic adaptation to the local effective dimension.

Visualizing the SNR clipping mechanism. Figure 40 illustrates the behavior of the key quantities in DANA-STAR-MK4 during training. We track four statistics averaged across all parameters:

- $\|m_t\|$: The norm of the first moment buffer. This quantity decays over time as the logarithmic time averaging accumulates more gradient history, even though the gradient norm itself does not decay. This motivates the need for the growing $(1 + t)^{1-\kappa}$ scaling factor.
- $\alpha_{\texttt{fac}}/\texttt{mfac}$: The effective exponent scaling. When no clipping occurs, this equals $t_{\texttt{eff}}^{1-\kappa}$ (shown as dashed line). When SNR clipping is active, this is reduced to $\texttt{clipsnr}/\texttt{mfac}$.
- $\texttt{mfac}$: The normalized momentum magnitude serving as our SNR proxy. This quantity captures the ratio of momentum signal to accumulated variance.
- $\alpha_{\texttt{fac}}$: The actual scaling factor applied to the momentum term after clipping.

Sign-based momentum update. The DANA-STAR-MK4 update takes the form

$$\theta_{t+1} = \theta_t - \tilde{\gamma}_2 \cdot g_{t+1} \odot \texttt{norm} - \tilde{\gamma}_3 \cdot \text{sign}(m_{t+1}) \odot (\tilde{\tau}_{t+1} \cdot \alpha_{\texttt{fac}} + |m_{t+1}| \odot \texttt{norm}). \tag{97}$$

The use of $\text{sign}(m_{t+1})$ rather than m_{t+1} directly provides additional robustness by decoupling the direction from the magnitude of the momentum contribution. This sign-based update is reminiscent of techniques used in signSGD and related methods, but here it is applied specifically to the logarithmic-time momentum term.

The full DANA-STAR-MK4 algorithm is presented in Algorithm 11. The SNR clipping provides automatic adaptation: when the signal-to-noise ratio is high (clean gradient signal), the full $(1 + t_{\texttt{eff}})^{1-\kappa}$ scaling is used; when the SNR is low (noisy signal), the momentum contribution is automatically reduced. This makes DANA-STAR-MK4 more robust to misspecification of κ and to heterogeneous effective dimensions across layers.

We also define the DANA-MK4 algorithm (see Algorithm 10) in which we set $\tau \equiv 1$ in DANA-STAR-MK4. This is used to illustrate separately the reduction in sensitivity to the choice of κ without the sparsity corrector.

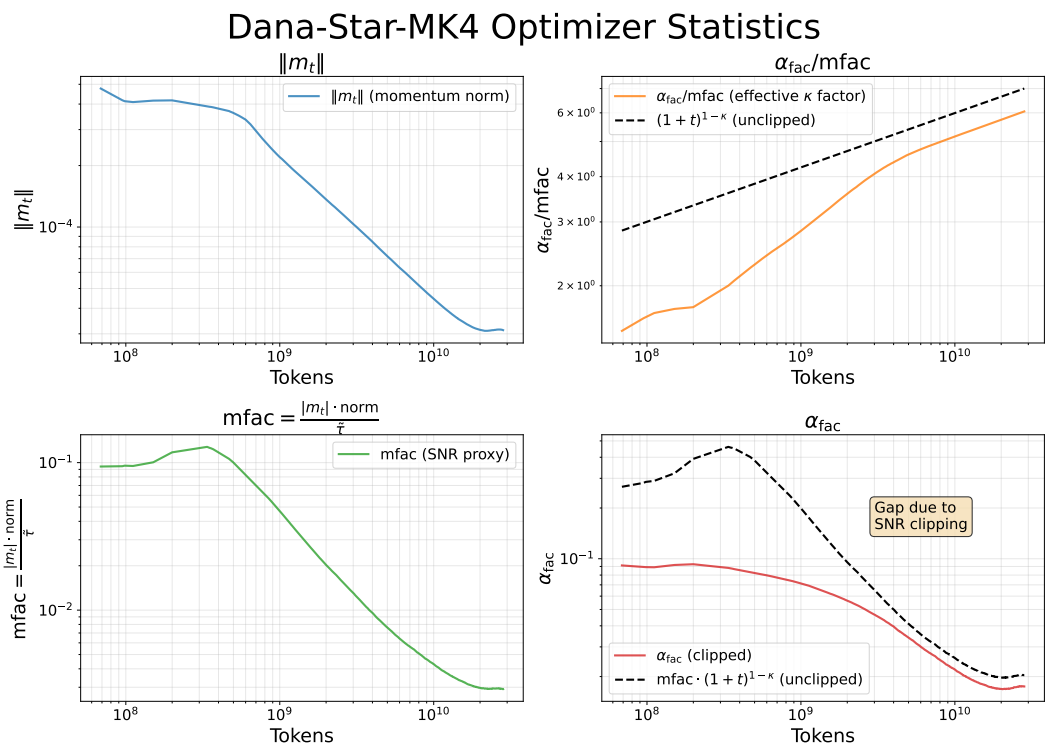


Figure 40: **SNR clipping statistics during training.** Optimizer statistics for a 32-head Enoki model with $\kappa = 0.85$ and `clipsnr` $= 0.125$. **Top left:** The momentum norm $\|m_t\|$ decays over time. **Top right:** The effective κ factor $\alpha_{\texttt{fac}}/\texttt{mfac}$ (solid) compared to the theoretical $(1+t)^{1-\kappa}$ schedule (dashed); the SNR clipping causes the actual schedule to fall below the theoretical curve. **Bottom left:** The `mfac` provides the SNR proxy. **Bottom right:** The $\alpha_{\texttt{fac}}$ shows the actual scaling applied after clipping; the gap from the dashed line ($\texttt{mfac} \cdot (1+t)^{1-\kappa}$) reflects the effect of SNR clipping.

Table 20: **ADANA variant comparison.** Each variant addresses specific failure modes while maintaining the core benefits of logarithmic time scheduling.

Variant	Key Feature
ADANA	Base logarithmic-time scheduling
DANA-MK4	+ SNR clipping (κ robustness)
DANA-STAR	+ τ estimator (sparse gradients)
DANA-STAR-MK4	+ both

N.3 COMPARISON OF VARIANTS

On standard transformer training, all variants perform similarly. The robust variants show their value in challenging settings with sparse gradients.

For most applications, the base ADANA algorithm suffices. DANA-MK4 adds stability with essentially no tradeoff (`clipsnr` $\in [0.25, 2.0]$ are effective and can be set and generally ignored). We do not have an obvious motivating use case for DANA-STAR or DANA-STAR-MK4, and they require additional memory overhead. It could be that at larger scales, batch sizes, or models that directly integrate sparsity (e.g. through MOE or gating mechanisms),these methods are important.

The computational overhead of the robust variants is minimal, requiring only one additional division per parameter per step for the τ estimator. Consequently, they can be used as a safe default when robustness is valued over the marginal computational savings.

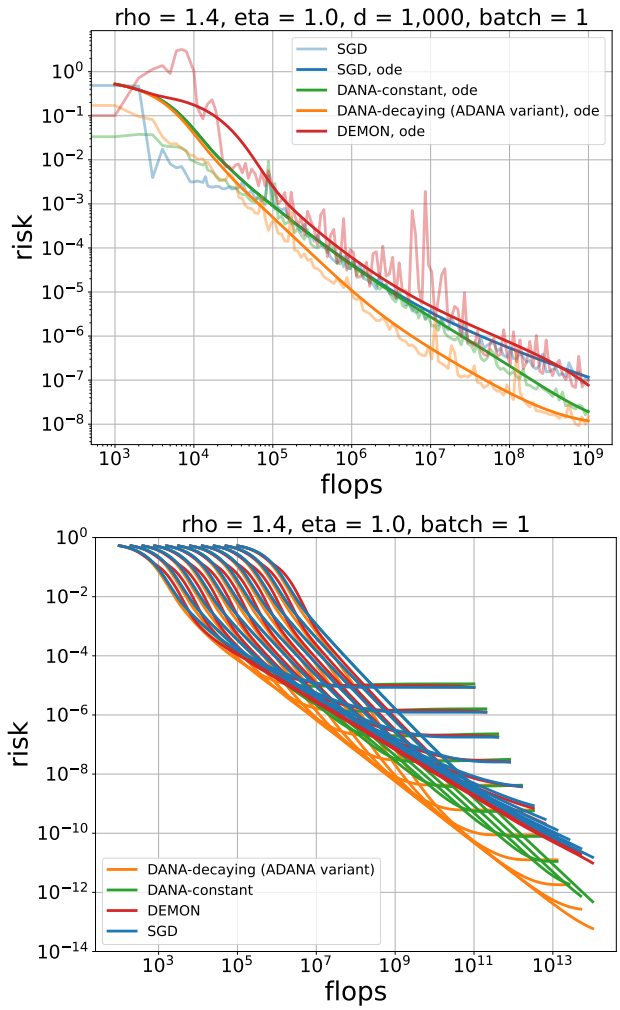


Figure 41: **Comparison of SGD, DANA (Alg. 8, see also Ferbach et al. 2025) with DEMON (Chen et al. 2019)** on the power-law random features model (Sec. B.1) with $\rho = 1.4$ and $\eta = 1.0$. Numerical setup: $D = \text{diag}(j^{-2\rho} : j = 1, \dots, d)$; SGD (blue curves) learning rate $\gamma = 0.5/\text{Tr}(D)$, DANA-constant (green) has $\gamma = 0.5/\text{Tr}(D)$, $\alpha(t) = 0.1/d \times (1+t)$, $\beta_1 = 1 - \delta/(1+t)$ where $\delta = \max\{2 - 1/\rho, (2\rho + 2\eta - 1)/\rho\} + 1$; DANA-decaying/ADANA variant (orange) same $\gamma(t)$ and β_1 as DANA-constant, $\alpha(t) = 0.1 \times (1+t)^{1-1/(2\rho)}$; DEMON (red) $\beta_{\text{init}} = 0.1$ and lr, $\tilde{\eta} = 0.5/\text{Tr}(D)$ (Chen et al. 2019); **Left:** algorithms were run for 10^6 steps with $d = 1000$, $v = 10000$; **Right:** algorithms run using the ODEs equivalences in (Ferbach et al. 2025) since DEMON is a version of Accelerated SGD (Jain et al., 2018) so it has an ODE equivalent; 10^9 iterations of algorithm, $d = \{100 \times 2^i\}$, $i = 1, \dots, 10$ and $v = 10 \times d$. ***DEMON (red) scales very closely with SGD (blue) and not does appear to accelerate*** for this (ρ, η). ***We see both* DANA*-decaying (motivation for* ADANA*) and* DANA*-constant accelerate.*** Note that it was difficult to get DEMON to converge as $\beta_{\text{init.}}$ needed to have small values; bigger β_{init} than 0.1 often diverged.

O DERIVATIONS FOR DIFFERENT FORMULATIONS OF NESTEROV'S ACCELERATED METHOD & GENERALIZED NESTEROV ACCELERATED METHOD.

In this section, we show how to derive different formulations for Nesterov's Accelerated Method and a generalized version – useful when the gradients are stochastic.

O.1 Standard Nesterov Formulations

We recall the naive stochastic Nesterov's accelerated method in (77), which we called **Two-sequence Nesterov:**

$$\theta_{t+1} = y_t - \gamma \nabla \mathcal{L}(y_t), \quad y_{t+1} = \theta_{t+1} + \mu_t(\theta_{t+1} - \theta_t), \quad \text{where} \quad \mu_t = \frac{t}{t+3}. \tag{98}$$

Note that Nesterov (1988); Beck and Teboulle (2009) use $1 - \frac{3}{t+3}$; however as noted in Lan (2012); Flammarion and Bach (2015), one can also use this formulation with $\mu_t = 1 - \frac{2}{t+1}$ to achieve the optimal rate of convergence for non-strongly convex, differentiable functions.

Look-ahead Nesterov. Our first goal is to rewrite this in terms of different set of variables and in particular, using a 'look-ahead' term. Define new variables

$$\boxed{v_{t+1} \stackrel{\text{def}}{=} \theta_{t+1} - \theta_t \quad \text{and} \quad \theta_t \stackrel{\text{def}}{=} \theta_t.}$$

Using the definition of y_{t+1} in (98), we have that $y_{t+1} = \theta_{t+1} + \mu_t v_{t+1}$ and $y_t = \theta_t + \mu_{t-1} v_t$. By definition of θ_{t+1} in (98) together with our v_{t+1},

$$\begin{aligned} v_{t+1} &= \theta_{t+1} - \theta_t = y_t - \gamma \nabla \mathcal{L}(y_t) - \theta_t \\ &= y_t - \gamma \nabla \mathcal{L}(y_t) - y_t + \mu_{t-1} v_t \\ &= \mu_{t-1} v_t - \gamma \nabla \mathcal{L}(y_t). \end{aligned}$$

Therefore, we have in our new variables (v_t, θ_t),

$$\boxed{\theta_{t+1} = \theta_t + v_{t+1}, \quad v_{t+1} = \mu_{t-1} v_t - \gamma \nabla \mathcal{L}(\theta_t + \mu_{t-1} v_t), \quad \text{where} \quad \mu_t = \frac{t}{t+3}.} \tag{99}$$

We call this formulation, *Look-ahead Nesterov.*

Extra-gradient Nesterov. We now rewrite (99) in terms of general DANA framework Ferbach et al. (2025) or an extra gradient. For this we define,

$$\boxed{\Phi_t \stackrel{\text{def}}{=} \theta_t + \mu_{t-1} v_t, \quad g_{t+1} \stackrel{\text{def}}{=} \nabla \mathcal{L}(\Phi_t), \quad \text{and} \quad m_t \stackrel{\text{def}}{=} \frac{-v_t}{\gamma}.}$$

Here again $\mu_t = \frac{t}{t+3}$. Since $\theta_{t+1} = \theta_t + v_{t+1}$ and $\theta_t = \Phi_t - \mu_{t-1} v_t$, we have that $\theta_{t+1} = \Phi_t - \mu_{t-1} v_t + v_{t+1}$. It follows that

$$\begin{aligned} \Phi_{t+1} &= \theta_{t+1} + \mu_t v_{t+1} = \Phi_t - \mu_{t-1} v_t + v_{t+1} + \mu_t v_{t+1} \\ &= \Phi_t - \mu_{t-1} v_t + (1 + \mu_t) v_{t+1} \\ (v_{t+1} = \mu_{t-1} v_t - \gamma g_t) \qquad &= \Phi_t - \mu_{t-1} v_t + (1 + \mu_t)(\mu_{t-1} v_t - \gamma g_{t+1}) \\ &= \Phi_t + \mu_t \mu_{t-1} v_t - (1 + \mu_t) \gamma g_{t+1}. \end{aligned}$$

Using that $m_t = -v_t/\gamma$ and $v_{t+1} = \mu_{t-1} v_t - \gamma g_{t+1}$, we have that

$$m_{t+1} = \frac{-v_{t+1}}{\gamma} = \frac{-\mu_{t-1} v_t}{\gamma} + g_{t+1} = \mu_{t-1} m_t + g_{t+1} \quad \text{and} \quad \mu_t \mu_{t-1} v_t = -\mu_t \mu_{t-1} \gamma m_t. \tag{100}$$

This yields that

$$\mu_t m_{t+1} = \mu_t \mu_{t-1} m_t + \mu_t g_{t+1} \quad \Rightarrow \quad \mu_t m_{t+1} - \mu_t g_{t+1} = \mu_t \mu_{t-1} m_t. \tag{101}$$

Combining (100) and (101), we have that

$$\begin{aligned} \Phi_{t+1} &= \Phi_t - \gamma \mu_t (m_{t+1} - g_{t+1}) - (1 + \mu_t) \gamma g_{t+1} \\ &= \Phi_t - \gamma \mu_t m_{t+1} - \gamma g_{t+1} \\ &= \Phi_t - \gamma (g_{t+1} + \mu_t m_{t+1}) \end{aligned}$$

Consequently, we have that

$$\boxed{\begin{aligned} &m_{t+1} = \mu_{t-1} m_t + g_{t+1} \quad \text{and} \quad \Phi_{t+1} = \Phi_t - \gamma (g_{t+1} + \mu_t m_{t+1}), \\ &\qquad \text{where } g_{t+1} = \nabla \mathcal{L}(\Phi_t) \text{ and } \mu_t = \frac{t}{t+3}. \end{aligned}} \tag{102}$$

We denote this formulation as *Extra-gradient Nesterov*.

EMA Nesterov. Since we used no properties that $g_{t+1} = \nabla\mathcal{L}(\Phi_t)$, we can redefine g_{t+1} in the generalized Nesterov formulation (102) so that

$$g_{t+1} = (1-\mu_{t-1})\tilde{g}_{t+1}.$$

Using this we get that the momentum update m_{t+1} becomes an exponential moving average which is similar to β_1 in ADAMW. Let us define

$$p_{t+1} = \mu_{t-1}p_t + (1-\mu_{t-1})g_{t+1}.$$

Thus we can think of $p_{t+1} \approx (1-\mu_{t-1})m_{t+1}$ and Φ_t remains the same as in (102). While this is not a direct equivalence, it is close and it better aligns with the implementation of β_1 in ADAMW. We call this formulation, *exponential moving average (EMA) Nesterov*, and its updates are

$$\boxed{p_{t+1} = \mu_{t-1}p_t + (1-\mu_{t-1})g_{t+1} \quad \text{and} \quad \Phi_{t+1} = \Phi_t - \gamma\left(g_{t+1} + \frac{\mu_t}{1-\mu_{t-1}}\cdot p_{t+1}\right)} \qquad (103)$$

$$\text{where} \quad m_{t+1} \approx (1-\mu_{t-1})p_{t+1}.$$

We summarize the results of the different formulations below.

Nesterov Version	Update Rules	Relationship
Extra-gradient Nesterov (102) (Φ_t, m_t)	$g_{t+1} = \nabla\mathcal{L}(\Phi_t)$ $m_{t+1} = \mu_{t-1}m_t + g_{t+1}$ $\Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \mu_t m_{t+1})$	
Look-ahead Nesterov (99) (θ_t, v_t)	$v_{t+1} = \mu_{t-1}v_t - \gamma\nabla\mathcal{L}(\theta_t + \mu_{t-1}v_t)$ $\theta_{t+1} = \theta_t + v_{t+1}$	$v_t = -\gamma m_t$ $\theta_t = \Phi_t + \mu_{t-1}\cdot\gamma\cdot m_t$
2-sequence Nesterov (98) (θ_t, y_t)	$\theta_{t+1} = y_t - \gamma\nabla\mathcal{L}(y_t)$ $y_{t+1} = \theta_{t+1} + \mu_t(\theta_{t+1} - \theta_t)$	$y_t = \Phi_t$ $\theta_t = \Phi_t + \mu_{t-1}\cdot\gamma\cdot m_t$
EMA Nesterov (103) (Φ_t, p_t)	$g_{t+1} = \nabla\mathcal{L}(\Phi_t)$ $p_{t+1} = \mu_{t-1}p_t + (1-\mu_{t-1})g_{t+1}$ $\Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \frac{\mu_t}{1-\mu_{t-1}}\cdot p_{t+1})$	Not an exact equivalence $p_{t+1} \approx \frac{m_{t+1}}{1-\mu_{t-1}}$

Table 21: We summarize the different formulations of standard Nesterov accelerated method, Section O.1. The relationship column shows how to go between the different parameters. Here $\gamma > 0$ is a learning rate and $\mu_t = \frac{t}{t+3}$.

O.2 GENERALIZED NESTEROV FORMULATION: ADDING A DAMPING FACTOR

As explained in Section L.1.2, when using stochastic gradients in Nesterov, (102), the noise from the gradients grows and dominates the parameter updates Φ_t. Since there is only one learning rate γ to control both the noise from the stochastic gradients in g_t and m_{t+1}, this causes divergence. Another way of stating this is that there is no choice of γ so that both g_{t+1} and m_{t+1} survive. You would end up driving the g_{t+1} to 0 (since m_{t+1} has more stochastic noise), but you need both of them for Nesterov's Accelerated Method to converge. To combat this, we introduce *Generalized Nesterov Accelerated Method*, which by choosing a new hyperparamter so that Nesterov is "damped". This allows for the algorithm to converge with stochastic gradients.

Generalized Extra-gradient Nesterov. The *Generalized Nesterov Acceleration Method* introduces a learning rate $\hat{\alpha}_t$ to scale the momentum term m_{t+1} in the parameter update rule. By choosing this learning such that $\hat{\alpha}_t \leq \mu_t$, we can damp the momentum term. Choosing $\hat{\alpha}_t > \mu_t$, amplifies momentum. Since we are concerned with the accumulation of noise from the stochastic gradients in the momentum term, we will want to dampen momentum, or equivalently, choose $\hat{\alpha}_t < \mu_t$. On the

other hand, we do not want to dampen too much, in which we would loose the acceleration effects we desire and become SGD. By choosing an appropriate $\hat{\alpha}_t$, Ferbach et al. (2025) show that you can accelerate *and* even outscale SGD on the power-law random features model (PLRF) (see Section B).

The *Generalized Nesterov Accelerated Method* updates as

$$\boxed{\begin{aligned} m_{t+1} &= \mu_{t-1} m_t + g_{t+1} \quad \text{and} \quad \Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \hat{\alpha}_t \cdot m_{t+1}) \\ &\text{where } g_{t+1} = \nabla \mathcal{L}(\Phi_t) \text{ and } \mu_t = \frac{t}{t+3}, \end{aligned}} \tag{104}$$

where $\hat{\alpha}_t$ is a learning rate that needs to be properly tuned.

	Momentum Update	Parameter Update	Hyperpameters
Standard Nesterov (102)	$m_{t+1} = \mu_{t-1} m_t + g_{t+1}$	$\Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \mu_t \cdot m_{t+1})$	$g_{t+1} = \nabla \mathcal{L}(\Phi_t), \quad \mu_t = \frac{t}{t+3}$
Generalized Nesterov (104)	$m_{t+1} = \mu_{t-1} m_t + g_{t+1}$	$\Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \hat{\alpha}_t \cdot m_{t+1})$	$g_{t+1} = \nabla \mathcal{L}(\Phi_t), \ \mu_t = \frac{t}{t+3}$

Here we replaced μ_t in (102) with $\hat{\alpha}_t$. Note that $\lim_{t\to\infty} \mu_t = 1$, so we can view μ_t as essentially 1 and we want to damped this term with an α_t which is decaying in t. In Ferbach et al. (2025) (see DANA-DECAYING Alg.), the authors showed that the optimal choice for this dampening factor in the PLRF (see Section B) is $\hat{\alpha}_t = (1+t)^{-\kappa}$ for some κ depending on the power-law covariance structure of the data.

By a simple substitution, we have that

$$\begin{aligned} \Phi_{t+1} = \Phi_t - \gamma(g_{t+1} + \hat{\alpha}_t(\mu_{t-1} m_t + g_{t+1})) &= \Phi_t - \gamma(1+\hat{\alpha}_t) g_{t+1} - \gamma \hat{\alpha}_t \mu_{t-1} m_t \\ &= \Phi_t - \gamma(1+\hat{\alpha}_t) g_{t+1} + \hat{\alpha}_t \mu_{t-1} v_t. \end{aligned} \tag{105}$$

The last equality follows from the relationship between v and m, namely $v_t = -\gamma m_t$.

Generalized Look-ahead Nesterov. Let us show that *Generalized Nesterov Accelerated Method* (104) can be formulated as a *generalized Look-Ahead Nesterov* (see update rule in (99)). For this, we proposed as an Ansatz for the dynamics in (θ, v) space that

$$v_{t+1} = \mu_{t-1} v_t - \gamma g_{t+1} \quad \text{and} \quad \theta_{t+1} = \theta_t + \delta_t v_{t+1}, \quad \text{where} \quad g_{t+1} = \nabla \mathcal{L}(\theta_t + \eta_{t-1} v_t).$$

We will choose δ_t and η_{t-1} so that it matches the dynamics in (105). Note in the original Look Ahead Version $\eta_{t-1} = \mu_{t-1}$ and $\delta_t = 1$. Let us define $\Phi_t \overset{\text{def}}{=} \theta_t + \eta_{t-1} v_t$ since Φ_t was our look ahead parameter. Then we have the following

$$\begin{aligned} \Phi_{t+1} &= \theta_{t+1} + \eta_t v_{t+1} \\ (\theta_{t+1} = \theta_t + \delta_t v_{t+1}) \quad &= \theta_t + (\delta_t + \eta_t) v_{t+1} \\ (\theta_t = \Phi_t - \eta_{t-1} v_t) \quad &= \Phi_t - \eta_{t-1} v_t + (\delta_t + \eta_t) v_{t+1} \\ (v_{t+1} = \mu_{t-1} v_t - \gamma g_{t+1}) \quad &\Phi_t - \eta_{t-1} v_t + (\delta_t + \eta_t)(\mu_{t-1} v_t - \gamma g_{t+1}) \\ &= \Phi_t + \left[(\delta_t + \eta_t)\mu_{t-1} - \eta_{t-1}\right] v_t - \gamma(\delta_t + \eta_t) g_{t+1} \end{aligned}$$

Now we match up coefficients in (105), specifically we see that

$$\begin{aligned} \text{coefficients } g_t\text{:} \quad & 1 + \alpha_t = \delta_t + \eta_t \\ \text{coefficients } v_t\text{:} \quad & \alpha_t \mu_{t-1} = (\delta_t + \eta_t)\mu_{t-1} - \eta_{t-1} = (1 + \hat{\alpha}_t)\mu_{t-1} - \eta_{t-1} \end{aligned}$$

Solving this, we get that $\eta_{t-1} = \mu_{t-1}$ and $\delta_t = 1 + \hat{\alpha}_t - \mu_t$. Putting this altogether, we have the following for the (generalized) Look-ahead Nesterov method

$$\boxed{v_{t+1} = \mu_{t-1} v_t - \gamma g_t \quad \text{and} \quad \theta_{t+1} = \theta_t + \delta_t v_{t+1}, \quad \text{where} \quad g_t = \nabla \mathcal{L}(\theta_t + \mu_{t-1} v_t) \quad \text{and} \quad \delta_t = 1 + \hat{\alpha}_t - \mu_t.}$$

$$\text{Change of variables:} \quad v_t = -\gamma m_t \quad \text{and} \quad \theta_t = \Phi_t + \gamma \mu_{t-1} m_t. \tag{106}$$

This formulation of the algorithm is nice because it has an intepretation as in terms of physical quantities such as momentum and damping. We summarize the results below.

Damping factor $\hat{\alpha}_t$	Learning rate $\delta_t = 1 + \hat{\alpha}_t - \mu_t$	Behavior
$\hat{\alpha}_t = \mu_t$	$\delta_t = 1$	Standard Nesterov
$\boldsymbol{\hat{\alpha}_t < \mu_t}$	$\boldsymbol{\delta_t < 1}$	**Damped steps (more conservative)**
$\hat{\alpha}_t > \mu_t$	$\delta_t > 1$	Amplified steps (more agressive)
$\hat{\alpha}_t = 0$	$\delta_t = 1 - \mu_t$	No momentum contribution/reduces to SGD

Given the issues with stability of standard Nesterov when the gradients are stochastic, we are interested in the regime where $\boldsymbol{\hat{\alpha}_t < \mu_t}$, so that we take more conservative momentum steps. We do not want to make this too small because then we are not doing any momentum. In the work of Ferbach et al. (2025) with DANA-DECAYING, they propose a good choice for $\boldsymbol{\hat{\alpha}_t = (1+t)^{-\kappa}}$ where $\kappa = 1/(2\rho)$ on the PLRF problem (see Section B). The ρ corresponds to the exponent on the power-law covariance of the input data x where $x_i \sim N(0, i^{-2\rho})$.

Generalized Two-sequence Nesterov. We finish by also expressing the generalized form back into the original $(\tilde{x}, y)$ variables in (77). In order to match the value that the gradient g_t is evaluated on, we defined $y_t = \theta_t + \mu_{t-1} v_t$. Thus $g_{t+1} = \nabla \mathcal{L}(y_t)$. From y_t, we have that $v_t = \frac{y_t - \theta_t}{\mu_{t-1}}$. Thus, the v_{t+1}-update and θ_t-parameter update become

$$v_{t+1} = y_t - \theta_t - \gamma g_{t+1} \quad \text{and} \quad \theta_{t+1} = \theta_t + \boldsymbol{\delta_t}\big[y_t - \theta_t - \gamma g_{t+1}\big].$$

We see that $\theta_{t+1} = \theta_t + \boldsymbol{\delta_t} v_{t+1} \Rightarrow v_{t+1} = \frac{\theta_{t+1} - \theta_t}{\boldsymbol{\delta_t}}$. Therefore, the next look ahead point is

$$y_{t+1} = \theta_{t+1} + \mu_t v_{t+1} = \theta_{t+1} + \frac{\mu_t}{\boldsymbol{\delta_t}}\big(\theta_{t+1} - \theta_t\big).$$

Putting everything together, we have the following

$$\boxed{\theta_{t+1} = (1 - \boldsymbol{\delta_t})\theta_t + \delta_t y_t - \gamma \boldsymbol{\delta_t} g_{t+1} \quad \text{and} \quad y_{t+1} = \theta_{t+1} + \frac{\mu_t}{\boldsymbol{\delta_t}}(\theta_{t+1} - \theta_t), \quad \text{where} \quad g_{t+1} = \nabla \mathcal{L}(y_t).} \tag{107}$$

$$\text{Change of variables:} \quad y_t = \theta_t + \mu_{t-1} v_t \quad \text{and} \quad \theta_t = \theta_t.$$

In the original formulation, $\delta_t = 1$.

Generalized EMA Nesterov. Since we used no properties that $g_{t+1} = \nabla \mathcal{L}(\Phi_t)$, we can redefine g_t in the generalized Nesterov formulation (104) so that

$$g_{t+1} = (1 - \mu_{t-1})\tilde{g}_{t+1}.$$

Using this we get that the momentum update m_{t+1} becomes an exponential moving average which is similar to β_1 in ADAMW. Let us define

$$p_{t+1} = \mu_{t-1} p_t + (1 - \mu_{t-1}) g_{t+1}.$$

Thus we can think of $p_{t+1} \approx (1 - \mu_{t-1}) m_{t+1}$ and Φ_t remains the same as in (104). While this is not a direct equivalence, it is close and it better aligns with the implementation of β_1 in ADAMW. We call this formulation, *generalized exponential moving average (EMA) Nesterov*, and its updates are

$$\boxed{p_{t+1} = \mu_{t-1} p_t + (1 - \mu_{t-1}) g_{t+1} \quad \text{and} \quad \Phi_{t+1} = \Phi_t - \gamma \left(g_{t+1} + \frac{\hat{\boldsymbol{\alpha}}(\boldsymbol{t})}{1 - \mu_{t-1}} \cdot p_{t+1} \right)} \tag{108}$$

$$\text{where} \quad m_{t+1} \approx (1 - \mu_{t-1}) p_{t+1}.$$

O.3 CORRESPONDENCE BETWEEN EMA DANA AND THE ORIGINAL DANA FORMULATION

In this section, we discuss how to express the original formulation of DANA (Ferbach et al. (2025)) in terms of an exponential moving average (EMA) formulations. In (Ferbach et al., 2025) the authors write the DANA algorithm as

$$\begin{aligned} &\textit{(mom.)} \quad m_{t+1} = \beta_1(t) m_t + g_{t+1} \\ &\textit{(param.)} \quad \theta_{t+1} = \theta_t - \gamma(t)(g_{t+1} + \alpha(t) m_{t+1}). \end{aligned} \tag{Original DANA}$$

Generalized Version	Update Rules	Relationship	Standard Nesterov
Extra-gradient Nesterov (104) (Φ_t, m_t)	$g_t = \nabla\mathcal{L}(\Phi_t)$ $m_{t+1} = \mu_{t-1}m_t + g_t$ $\Phi_{t+1} = \Phi_t - \gamma(g_t + \hat{\boldsymbol{\alpha}}_t \cdot m_{t+1})$		$\hat{\boldsymbol{\alpha}}_t = \boldsymbol{\mu}_t$
Look-ahead Nesterov (106) (θ_t, v_t)	$v_{t+1} = \mu_{t-1}v_t - \gamma\nabla\mathcal{L}(\theta_t + \mu_{t-1}v_t)$ $\theta_{t+1} = \theta_t + \boldsymbol{\delta}_t \cdot v_{t+1}$	$v_t = -\gamma m_t$ $\theta_t = \Phi_t + \mu_{t-1}\cdot\gamma\cdot m_t$ $\boldsymbol{\delta}_t = \mathbf{1} + \hat{\boldsymbol{\alpha}}_t - \boldsymbol{\mu}_t$	$\boldsymbol{\delta}_t = \mathbf{1}$
2-sequence Nesterov (107) (θ_t, y_t)	$\theta_{t+1} = (\mathbf{1} - \boldsymbol{\delta}_t)\cdot\theta_t + \boldsymbol{\delta}_t \cdot y_t - \gamma\cdot\boldsymbol{\delta}_t\nabla\mathcal{L}(y_t)$ $y_{t+1} = \theta_{t+1} + \frac{\mu_t}{\delta_t}(\theta_{t+1} - \theta_t)$	$y_t = \Phi_t$ $\theta_t = \Phi_t + \mu_{t-1}\cdot\gamma\cdot m_t$ $\boldsymbol{\delta}_t = \mathbf{1} + \hat{\boldsymbol{\alpha}}_t - \boldsymbol{\mu}_t$	$\boldsymbol{\delta}_t = \mathbf{1}$
EMA Nesterov (108) (Φ_t, p_t)	$g_t = \nabla\mathcal{L}(\Phi_t)$ $p_{t+1} = \mu_{t-1}p_t + (1-\mu_{t-1})g_t$ $\Phi_{t+1} = \Phi_t - \gamma(g_t + \frac{\hat{\alpha}_t}{1-\mu_{t-1}}\cdot p_{t+1})$	Not an exact equivalence $p_{t+1} \approx \frac{m_{t+1}}{1-\mu_{t-1}}$	$\hat{\boldsymbol{\alpha}}_t = \boldsymbol{\mu}_t$

Table 22: We summarize the different formulations of standard Nesterov accelerated method, Section O.2. The relationship column shows how to go between the different parameters. Here $\gamma > 0$ is a learning rate and $\mu_t = \frac{t}{t+3}$.

where $\alpha(t) \asymp (1+t)^{-\kappa}$ for DANA-DECAYING and $\alpha(t) \asymp \frac{1}{d}$ on the PLRF for DANA-constant. They are respectively the largest monomial and constant schedules that avoid divergence of (Original DANA). To better understand this in the format of ADAM, we want to write the momentum as

$$m_{t+1} = \beta_1(t)m_t + (1 - \beta_1(t))g_{t+1}. \tag{109}$$

A precise correspondence can be made between both writings. Indeed, from (109) we have

$$m_t = \sum_{s=1}^{t-1}(1-\beta_1(s))g_s \prod_{r=s+1}^{t-1}\beta_1(r) \approx \sum_{s=1}^{t-1}\frac{\delta}{\delta+s}g_s\left(\frac{s+\delta}{t+\delta}\right)^{\delta}.$$

Hence using the EMA representation for m_{t+1}, we get for the parameter update

$$\theta_{t+1} \approx \theta_t - \gamma(t)\left(g_{t+1} + \hat{\alpha}(t)\sum_{s=1}^{t-1}\frac{\delta}{\delta+s}g_s\left(\frac{s+\delta}{t+\delta}\right)^{\delta}\right). \tag{110}$$

Similarly for (Original DANA), we can write the momentum as

$$m_t = \sum_{s=1}^{t-1} g_s \prod_{r=s+1}^{t-1}\beta_1(r) \approx \sum_{s=1}^{t-1} g_s\left(\frac{s+\delta}{t+\delta}\right)^{\delta}$$

and the parameter updates as

$$\theta_{t+1} \approx \theta_t - \gamma(t)\left(g_{t+1} + \hat{\alpha}(t)\sum_{s=1}^{t-1} g_s\left(\frac{s+\delta}{t+\delta}\right)^{\delta}\right). \tag{111}$$

We see that setting $\alpha(t) \overset{\text{def}}{=} \hat{\alpha}(t)\frac{\delta+t}{\delta}$, the two parameter updates are very similar:

$$\theta_{t+1} \approx \theta_t - \gamma(t)(g_{t+1} + \hat{\alpha}(t)\sum_{s=1}^{t-1} g_s\left(\frac{s+\delta}{t+\delta}\right)^{\delta}), \qquad \text{(Original Dana)}$$

as compared with

$$\theta_{t+1} \approx \theta_t - \gamma(t)(g_{t+1} + \hat{\alpha}(t)\sum_{s=1}^{t-1} g_s\left(\frac{s+\delta}{t+\delta}\right)^{\delta-1}). \qquad \text{(General Dana)}$$

Hence setting $\alpha(t) \stackrel{\text{def}}{=} \hat{\alpha}(t)\frac{\delta+t}{\delta}$ make a direct correspondence between both algorithms with the only change being from δ to $\delta - 1$. Note that this change is minor since Ferbach et al. (2025) noticed that the exact constant δ does not matter as long as it is large enough.

P PROOFS UNDER SPARSE GRADIENT ORACLE

To prove the results in Section L.2, we require bounds on the tightness of certain stochastic sequences arising from the Adam update rule. We formalize the simplified per-parameter stochastic process that models the transformation ADAM performs on gradients.

Let the sequence of random variables be defined as

$$Y_t = \sum_{r=1}^{t} \frac{A_t(r) g_r (1-\beta_1(r))}{\sqrt{\sum_{s=1}^{t} B_t(s)(g_s)^2(1-\beta_2(s))} + \epsilon} =: \frac{m_t}{\sqrt{v_t}+\epsilon}$$

where $A_t(r) = \prod_{j=r+1}^{t} \beta_1(j)$, $B_t(s) = \prod_{j=s+1}^{t} \beta_2(j)$, and $g_r = X_r B_r$ with X_r independent random variables with second moment 1 and $B_r \sim \text{Bernoulli}(p)$. This represents a simplified per-parameter stochastic process which models the transformation that ADAM performs to a gradient.

At the very least, this transformation should not greatly expand the order of magnitude of the updates—in the simple setup proposed, we should have that Y_t remain stochastically bounded (which is the same order as the original input sequence). However, due to sparsity, there is another effect. In the time between nonzero updates, the optimizer continues to apply updates, even though no additional gradient information has been provided.

Hence if we let T_ℓ be the times at which nonzero gradients occur (i.e. the times at which consecutive nonzero B_r occur), then the cumulative sum of updates between those times should remain bounded:

$$Z_\ell \stackrel{\text{def}}{=} \sum_{t=T_\ell}^{T_{\ell+1}-1} Y_t, \qquad \ell \geq 1.$$

We would like that these remain bounded, which will allow us to distinguish bounded rules from unbounded rules.

We begin by showing that the usual application of ADAM remains bounded in this sense. Throughout this section, we apply inequalities and actions entry-wise.

Theorem P.1 (Boundedness of Standard ADAM, Theorem L.3)**.** *If $\beta_1 \in (0,1)$ and $\beta_2 \in (0,1)$ are fixed constants (not depending on k) and $\beta_1^2 < \beta_2$, the family of random variables $\{Z_\ell : \ell \geq 1, p \in (0,1), \epsilon > 0\}$ satisfies the uniform first-moment bound*

$$\mathbb{E}\,|Z_\ell| \leq \frac{1-\beta_1}{(\sqrt{1-\beta_2})(1-\frac{\beta_1}{\sqrt{\beta_2}})^2}.$$

Proof of Theorem P.1. We start by observing that the coefficient sequences $A_t(r) = \beta_1^{t-r}$ and $B_t(s) = \beta_2^{t-s}$. Fix $t \geq T_\ell$ with $\ell \geq 1$. For any $1 \leq j \leq \ell$, we have that $T_j \leq T_\ell$ and

$$v_t = \sum_{s=1}^{t} B_t(s)(g_s)^2(1-\beta_2(s)) \geq B_t(T_j) g_{T_j}^2 (1-\beta_2) = \beta_2^{t-T_j} X_{T_j}^2 (1-\beta_2).$$

Here we used that $g_{T_j} = X_{T_j}$ since $B_{T_j} = 1$ as T_j is the jth nonzero gradient seen.

As for the first moment, we have for any $t \in [T_\ell, T_{\ell+1} - 1]$,

$$m_t = \sum_{r=1}^{t} A_t(r) g_r (1-\beta_1(r)) = \sum_{j=1}^{\ell} A_t(T_j) X_{T_j}(1-\beta_1(T_j)) = \sum_{j=1}^{\ell} \beta_1^{t-T_j} X_{T_j}(1-\beta_1).$$

Here again we used that only the nonzero gradients are counted in the summation for m_t. Putting this together, we have the following bound

$$\frac{|m_t|}{\sqrt{v_t}+\epsilon} \leq \sum_{j=1}^{\ell} \frac{\beta_1^{t-T_j} X_{T_j}(1-\beta_1)}{\sqrt{\beta_2^{t-T_j} X_{T_j}^2 (1-\beta_2)}} \leq \frac{1-\beta_1}{\sqrt{1-\beta_2}} \sum_{j=1}^{\ell} \left(\frac{\beta_1}{\sqrt{\beta_2}}\right)^{t-T_j}.$$

Summing over $t \in [T_\ell, T_{\ell+1} - 1]$, we have that

$$\begin{aligned}
|Z_\ell| &= \sum_{t=T_\ell}^{T_{\ell+1}-1} \frac{|m_t|}{\sqrt{v_t} + \epsilon} \leq \frac{1-\beta_1}{\sqrt{1-\beta_2}} \sum_{t=T_\ell}^{T_{\ell+1}-1} \sum_{j=1}^{\ell} \left(\frac{\beta_1}{\sqrt{\beta_2}}\right)^{t-T_j} \\
&= \frac{1-\beta_1}{\sqrt{1-\beta_2}} \sum_{j=1}^{\ell} \sum_{t=T_\ell}^{T_{\ell+1}-1} \left(\frac{\beta_1}{\sqrt{\beta_2}}\right)^{t-T_j} \\
&\leq \frac{1-\beta_1}{\sqrt{1-\beta_2}(1-\frac{\beta_1}{\sqrt{\beta_2}})} \sum_{j=1}^{\ell} \left(\frac{\beta_1}{\sqrt{\beta_2}}\right)^{T_\ell - T_j} \\
&\leq \frac{1-\beta_1}{\sqrt{1-\beta_2}(1-\frac{\beta_1}{\sqrt{\beta_2}})^2}.
\end{aligned}$$

This proves the result. □

We show that this rule remains salvageable for long first moment buffers, if we scale the updates by $\sqrt{p}$.

Theorem P.2 (Boundedness of log-time schedules for β_1 and β_2 if multiplied by $\sqrt{p}$, Theorem L.4)**.** *If $\beta_1(t) = \beta_2(t) = 1 - \frac{c}{t}$ for $c > 2$ the family of random variables $\{Z_\ell : \ell \geq 1, p \in (0,1), \epsilon > 0\}$ satisfies the first-moment bound*

$$\mathbb{E}\,|Z_\ell| \leq \frac{C}{\sqrt{p}}.$$

Proof of Theorem P.2. We have that the moment sequences satisfy the estimates

$$A_t(r) \asymp \left(\frac{r}{t}\right)^c, \quad B_t(s) \asymp \left(\frac{s}{t}\right)^c,$$

with the constants depending on c. Moreover, we have that by Riemann sum approximation,

$$\sum_{j=1}^{k} A_k(j)(1-\beta_1(j)) \to \int_0^1 ct^{c-1}\,\mathrm{d}t = 1.$$

For a given X^{T_r}, we observe the following estimates on the conditional moment of v_k:

$$\frac{1}{\sqrt{\mathbb{E}(v_k \mid X^{T_r})} + \epsilon} \leq \mathbb{E}\left(\frac{1}{\sqrt{v_k} + \epsilon} \mid X^{T_r}\right) \lesssim \left(\frac{T_r}{k}\right)^{-c/2} \frac{\sqrt{T_r}}{|X^{T_r}|}\mathfrak{p}_k + \sqrt{\frac{1}{p\,\mathbb{E}\,|X^{T_r}|^2}}, \tag{112}$$

with the left hand side following from Jensen's inequality, the $\mathfrak{p}_k$ in the right hand side giving the probability

$$\mathfrak{p}_k \coloneqq \Pr\left(M_k \leq \frac{1}{2}\,\mathbb{E}\,M_k\right) \quad \text{where} \quad M_k \coloneqq \sum_{j=2k/3}^{k} \frac{B_k(j)|g^j|^2}{j}. \tag{113}$$

(The bound then follows by selecting a group of $k/3$ terms not including T_r and bounding v_k below on the event described in the bound or on its complement.) From moment-based bounds, we have that for any $C > 0$, $\mathfrak{p}_k \lesssim_C (\max\{1, pk\})^{-C}$.

As for the first moment buffer, we have that for $k \in [T_\ell, T_{\ell+1} - 1]$:

$$m_k = \sum_{r=1}^{\ell} B_k(T_r) X^{T_r}(1-\beta_1(T_r)).$$

Thus summing over k in the range, we have that

$$\mathbb{E}\,(|Z_{\ell+1}|) \leq \sum_{r=1}^{\ell} \mathbb{E}\left(\sum_{k=T_\ell}^{T_{\ell+1}-1} \frac{B_k(T_r)|X^{T_r}|(1-\beta_1(T_r))}{\sqrt{v_k} + \epsilon}\right).$$

We then take a conditional expectation, using (112) and (113), and we have that

$$\mathbb{E}\left(\sum_{k=T_\ell}^{T_{\ell+1}-1}\frac{B_k(T_r)}{\sqrt{v_k}+\epsilon}\,\middle|\,X^{T_r},T_r\right)\lesssim \left.\begin{aligned}&\mathbb{E}\left(\sum_{k=T_\ell}^{T_{\ell+1}-1}\left(\frac{T_r}{k}\right)^{c/2}\frac{\sqrt{T_r}}{|X^{T_r}|}\mathfrak{p}_k\,\middle|\,X^{T_r},T_r\right)\\ &+\mathbb{E}\left(\sum_{k=T_\ell}^{T_{\ell+1}-1}\left(\frac{T_r}{k}\right)^{c}\sqrt{\frac{1}{p\,\mathbb{E}\,|X^{T_\ell}|^2}}\,\middle|\,X^{T_r},T_r\right)\end{aligned}\right\}\begin{aligned}&\text{Term-(i)}\\ &\text{Term-(ii)}.\end{aligned}$$

For Term-(ii) we bound the sum over k^{-c} by $(T_{\ell+1}-T_\ell)(T_\ell)^{-c}$, combining it with the factor $|X^{T_r}|(1-\beta_1(T_r))$ to give

$$\sum_{r=1}^{\ell}\mathbb{E}\left(|X^{T_r}|(1-\beta_1(T_r))\text{Term-(ii)}\right)\lesssim\sum_{r=1}^{\ell}c\,\mathbb{E}\left(\left(\frac{T_r}{T_\ell}\right)^{c-1}\frac{T_{\ell+1}-T_\ell}{T_\ell}\right)\frac{\mathbb{E}\,|X|}{\sqrt{p\,\mathbb{E}\,|X^2|}}\to\frac{\mathbb{E}\,|X|}{\sqrt{p\,\mathbb{E}\,|X^2|}},$$

as $\ell\to\infty$. This is because the sum is a stochastic Riemann sum approximation to the integral $\int_0^1 ct^{c-1}\,\mathrm{d}t=1$; the ratio $(T_{\ell+1}-T_\ell)\ell/T_\ell$ converges in moments to 1, while the ratio T_r/T_ℓ converges to r/ℓ.

As for Term-(i),

$$\sum_{r=1}^{\ell}\mathbb{E}\left(|X^{T_r}|(1-\beta_1(T_r))\text{Term-(i)}\right)\lesssim\sum_{r=1}^{\ell}\mathbb{E}\left(\left(\frac{T_r}{T_\ell}\right)^{c/2-1}\frac{(T_{\ell+1}-T_\ell)\sqrt{T_r}}{T_\ell}\max\{1,pT_\ell\}^{-1/2}\right).$$

The expression $\sqrt{T_r}\max\{1,pT_\ell\}^{-1/2}$ is bounded deterministically by $\sqrt{1/p}$, and hence we arrive at both Term-(i) and Term-(ii) giving

$$\mathbb{E}(|Z_{\ell+1}|)\le\frac{C}{\sqrt{p}}.$$

□

Conversely, we show that with $\beta_1(\ell)=1-\frac{c}{\ell}$ it is not possible to choose a $\beta_2(\ell)$ which is monotone increasing and produces a bounded sequence.

Theorem P.3 (Unboundedness of LOG-ADAMW, Theorem L.1)**.** *Suppose $\beta_1(\ell)=1-c/\ell$ for $c>1$. For any monotone sequence $\beta_2(\ell)\in(0,1)$, the family of random variables $\{Z_\ell:\ell\ge1,p\in(0,1),\epsilon>0\}$ is unbounded, i.e.*

$$\limsup_{p\to0}\sup_{\ell\ge1,\epsilon>0}\mathbb{E}(|Z_\ell|)=\infty.$$

In fact it follows that along a subsequence of $p\to0$,

$$|Z_2|\xrightarrow{\Pr}\infty.$$

Proof. We consider the time of the first non-zero gradient. Let T_ℓ be the ℓ-th time j such that $B^j=1$. Then $T_\ell-T_{\ell-1}$ is a geometric random variable with parameter p. The probability that $T_1=k$ is $p(1-p)^{k-1}$. Then on sending $p\to0$ we have that

$$(pT_1,p(T_2-T_1))\xrightarrow{\text{law}}(\text{Exp}(1),\text{Exp}(1)),$$

in distribution as $p\to0$ with the convergence in probability, with the limiting random variables independent. In particular we have convergence

$$\Pr\left(T_1\in\frac{1}{p}[1,2],T_2\in\frac{1}{p}[4,5]\right)\to q\in(0,1).$$

Let $\mathcal{E}$ be this event. Then on this event we have that for $k\in[T_1,T_2-1]$

$$Y_k=\frac{X^{T_1}(1-\beta_1(T_1))(A_k(T_1))}{|X^{T_1}|\sqrt{B_k(T_1)(1-\beta_2(T_1))}+\epsilon}.$$

We have the representation

$$|Z_2| = \sum_{k=T_1}^{T_2-1} \frac{|X^{T_1}|(1-\beta_1(T_1))(A_k(T_1))}{|X^{T_1}|\sqrt{B_k(T_1)(1-\beta_2(k-T_1))}+\epsilon} \xrightarrow{\text{a.s.}} \sum_{k=T_1}^{T_2-1} \frac{(1-\beta_1(T_1))(A_k(T_1))}{\sqrt{B_k(T_1)(1-\beta_2(k-T_1))}},$$

as $\epsilon \to 0$. Hence for this to remain bounded over all p, we need that this expression is bounded uniformly above on the event $\mathcal{E}$.

On this event, since $A_k(j) \asymp (j/k)^c$ we have that it is bounded below by a constant, and using monotonicity of β_2 and $B_k(j)$ we conclude that for $\mathbb{E}\,|Z_2|$ to remain bounded independent of p, there is a constant $C > 0$

$$C \le (B_{4k}(2k))(1-\beta_2(k)) \le B_{4k}(2k).$$

Putting everything together, we have that for all k sufficiently large

$$-C \le \log B_{4k}(2k) \le \sum_{\ell=2k}^{4k} \log(1-(1-\beta_2(\ell))) \le -2k(1-\beta_2(4k)).$$

Rearranging and using monotonicity, we conclude that there is a $C > 0$ for all k sufficiently large

$$(1-\beta_2(k)) \le \frac{C\log k}{k}. \tag{114}$$

However this implies that

$$\sum_{k=T_1}^{T_2-1} \frac{(1-\beta_1(T_1))(A_k(T_1))}{\sqrt{B_k(T_1)(1-\beta_2(k-T_1))}} \gtrsim \sum_{k=T_1}^{T_2-1} \frac{1}{\sqrt{k\log k}} \gtrsim \frac{1}{\sqrt{p\log(1/p)}},$$

which is a contradiction. □

As a corollary, we conclude the following strong form of instability:

Theorem P.4 (Strong Blowup, Corollary L.2)**.** *Suppose that $F : \mathbb{R}^d \to \mathbb{R}$ is a coercive, Lipschitz loss, and suppose we are given access to a noisy unbiased gradient oracle $\mathcal{O}$, which for any given parameter $\theta \in \mathbb{R}^d$ provides a distributional, $L^2(\Pr)$ approximation of ∇F. That is, for any $\theta \in \mathbb{R}^d$ a sample $\mathcal{G} \sim \mathcal{O}(\theta)$ satisfies for some constant $M > 0$,*

$$\mathbb{E}\,\|\mathcal{G}\|^2 \le M \quad \textit{and} \quad \mathbb{E}(\mathcal{G}) = \nabla F(\theta).$$

For $p \in [0,1]$, let the sparse oracle $\mathcal{O}_p$ be defined by the sampling rule that $G' \sim \mathcal{O}_p(\theta)$ is given by $G' = BG$ where $G \sim \mathcal{O}(\theta)$ and where $B \sim \text{Bernoulli}(p)$ is independent of G. Suppose we run ADAM *with $\beta_1(\ell) = 1 - c/\ell$ for $c > 1$ and all $\ell > 1$ and suppose $\beta_2(\ell) \in (0,1)$ is any monotone sequence. Let $\{\theta^k\}$ be the iterates of* ADAM *generated from the noisy oracle $\mathcal{O}_p$. Let T_2 be the time at which the second nonzero gradient occurs. Then there is a sequence of $p \to 0$ so that*

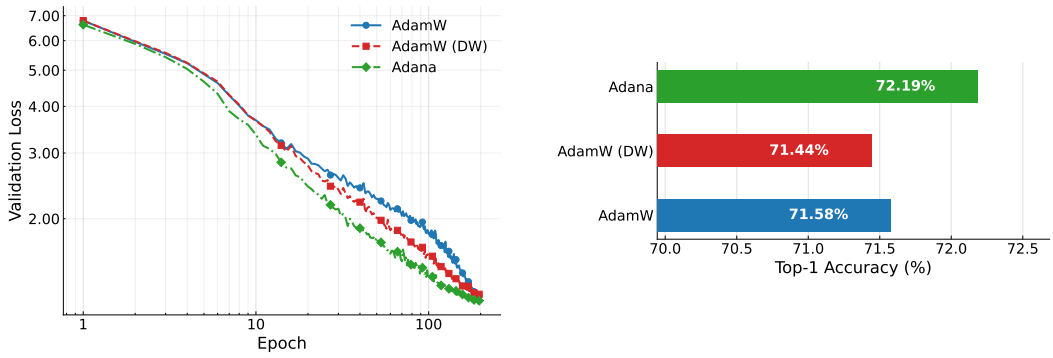
$$\theta^{T_2} \xrightarrow{\Pr} \infty.$$

Hence ADAM with $1-\beta_1(\ell) = c/\ell$ is essentially uniformly bad—on *every* problem with a sufficiently sparse oracle, it will cause arbitrarily bad iterates.

Q Additional Experiments

Q.1 Vision Experiment

In Figure 42 we present an empirical comparison of ADANA, ADAMW and ADAMW with decaying weight-decay on a vision training experiment. More precisely, we trained a Vision Transformer using the Timm library Wightman (2019) on the Imagenet dataset Deng et al. (2009) for 200 epochs and swept the learning rate. Importantly, ADANA shows an improvement over the baseline ADAMW while decaying weight-decay does not seem to improve.



(a) Validation loss during training.

(b) Final top-1 accuracy.

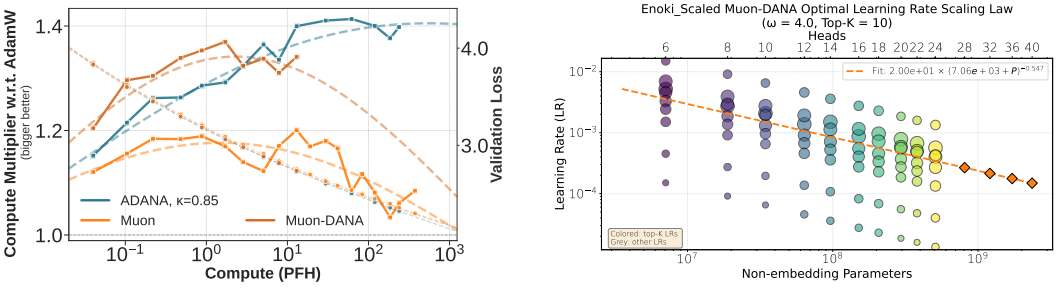
Figure 42: **Vision transformers (ViT) on Imagenet for 200 epochs** Comparison of ADANA, ADAMW and ADAMW with decaying weight-decay (DW). ADANA outperforms ADAMW throughout training and final validation loss. Decaying weight-decay does not seem to improve final validation loss over vanilla ADAMW but smooths out the learning curve. Showing the best learning rate after tuning.

Q.2 COMBINING MUON AND ADANA SCHEDULES

As we previously mentioned, ADANA's logarithmic-time schedules could potentially be combined with MUON's spectral orthogonalization. There are different ways to combine Muon and DANA, for example, putting the Newton-Schultz operator before or after the momentum. In Figure 43 and Figure 44 we compare ADAMW, MUON, ADANA and a potential candidate Muon-DANA which is designed as follows:

- for 2D matrices, the Newton Schultz operator is applied to $g_t + \alpha(t)m_t$ where g_t is the gradient, m_t the long-momentum state and $\alpha(t)$ the damping schedule,
- for 1D matrices AdamW is replaced by ADANA.

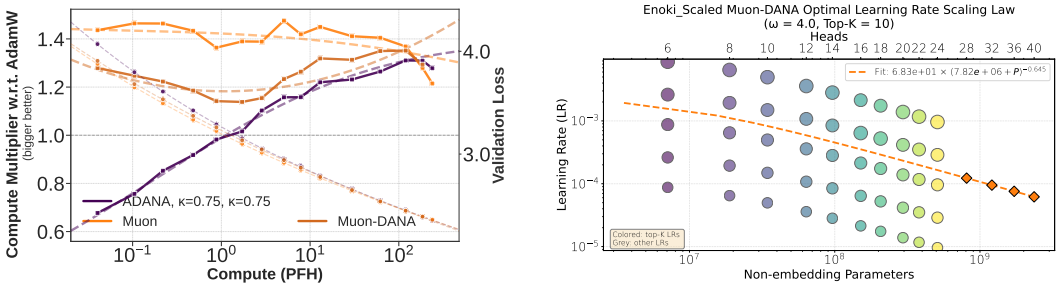
In the small batch setting Figure 43, Muon-DANA improves a bit over ADANA but rapidly degrades with scale. This is consistent with the fact that MUON's performance degrades at large scale against ADAMW. In the large batch setting Figure 44 where MUON outperforms ADANA at small scales, Muon-DANA improves over ADANA. However, at larger scales where MUON's performance degrades, Muon-DANA's advantage over ADANA degrades. Overall, it remains unclear whether MUON and ADANA gains can be fully added on top of each other. However, while we tested with one potential candidate by applying Newton-Schultz normalization after the momentum, studying other ways of combining Muon and Dana would be an interesting direction for future work.



(a) Scaling comparison ($\kappa = 0.85$).

(b) Learning rate extrapolation.

Figure 43: **Batch 32, $\kappa = 0.85$.** (a) Scaling comparison of MUON-DANA, ADANA and MUON against AdamW, and (b) learning rate extrapolation.



(a) Scaling comparison ($\kappa = 0.75$, large batch).

(b) Learning rate extrapolation (large batch).

Figure 44: **Batch 256, $\kappa = 0.75$.** (a) Scaling comparison of MUON-DANA, ADANA and MUON against AdamW, and (b) learning rate extrapolation.

R DETAILS ON THE ESTIMATION OF THE COMPUTE USED FOR THE EXPERIMENTS

In this section, we provide details on how we estimated the compute used for this project. Note first that this estimation concern the total compute used for this project(including the experiments that did not end up being in the paper)

We use the metric H100 GPU year which correspond to running a H100 for a year because most of our runs were using H100. Given that the Max Thermal Design Power of a H100 is between 350 and 700W, and between 300 and 400W for an A100. We used the following conversion system: 1hour with an A100 is equivalent to .5 hours with an H100.

Table 23: Compute Usage Report

GPU	Raw GPU days	H100 Factor	H100 Equiv Hours
A100	1801	0.5	900.5
H100	7138	1.0	55710
		Total H100 Equivalent Days:	**8038.5**
		Total H100 Equivalent Years:	**22**