OPTIMIZER MEMORY SCHEDULES FOR OUTSCALING THE OVERTRAINING AXIS

Anonymous authors
Paper under double-blind review

ABSTRACT

We show that optimizer memory schedules can produce an increasing token-efficiency advantage over AdamW as Transformer training horizons grow. We compare AdamW, Muon, SOAP, and ADANA across four model sizes from 51M to 458M parameters and overtraining (OT) factors from $1\times$ to $256\times$, sweeping the base learning rate at every setting. We introduce a timescale-based momentum cooldown rule that shortens optimizer memory during terminal learning rate decay, producing large long-horizon gains for ADANA that compound with log-time weight decay. With these schedules, ADANA outscales AdamW with a scaling advantage close to that predicted by DANA theory on power-law random features. Longer horizons generally favor longer fixed memory, but ADANA's scaling advantage persists after tuning AdamW's fixed memory separately at each horizon. Muon's token-efficiency advantage over AdamW is roughly constant across OT factors, while SOAP gains further at high OT. ADANA begins behind both matrix-preconditioned optimizers but surpasses Muon and becomes competitive with SOAP at our highest OT factors. These results establish training horizon as an essential axis for optimizer evaluation and design.

1 INTRODUCTION

Language models exhibit empirical scaling laws relating loss to model size, training data, and compute (Kaplan et al., 2020). In particular, the Chinchilla scaling law finds that model size and training tokens should grow together to minimize loss at fixed pretraining compute, corresponding to approximately 20 training tokens per parameter (Hoffmann et al., 2022). This result motivated subsequent model and optimizer scaling studies to evaluate performance near the Chinchilla-optimal training horizon. However, for widely deployed language models, the recurring cost of inference can dominate the one-time cost of pretraining (Sardana et al., 2024; Bian et al., 2025), and model size can be tightly constrained by memory and latency requirements, particularly for deployment on mobile devices (Liu et al., 2024). Both pressures favor training smaller models on far more tokens, a practice known as *overtraining* (Sardana et al., 2024; Gadre et al., 2024; Bian et al., 2025). Open-weight models can be trained on hundreds or thousands of times the Chinchilla ratio of tokens per parameter; for example, Llama 3 8B and Qwen3-0.6B are trained on about $100\times$ and $3000\times$ this ratio (Grattafiori et al., 2024; Yang et al., 2025). In this regime, high-quality data can also become scarce (Muennighoff et al., 2023; Villalobos et al., 2024). We show that optimizer memory schedules can produce a token-efficiency advantage that grows with the training horizon, which comparisons at a single horizon miss.

It is common for optimizer benchmarks to select hyperparameters and compare methods at a single training horizon or fixed token-to-parameter ratio. Optimizer rankings and optimal hyperparameter choices can change with training duration (Wen et al., 2025; Semenov et al., 2025; Wen et al., 2026), yet prior work covers a limited range of overtraining factors. We therefore study how optimizer performance and optimal hyperparameters scale across the overtraining axis.

The optimizers we study combine different forms of preconditioning with different memory rules. A preconditioner transforms a gradient by rescaling or mixing its coordinates before the parameter update. AdamW uses coordinatewise adaptive preconditioning, which rescales entries separately, whereas Muon and SOAP apply matrix-level preconditioning that can couple coordinates within a weight matrix (Jordan et al., 2024; Vyas et al., 2025); all three use scalar averaging coefficients that

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.

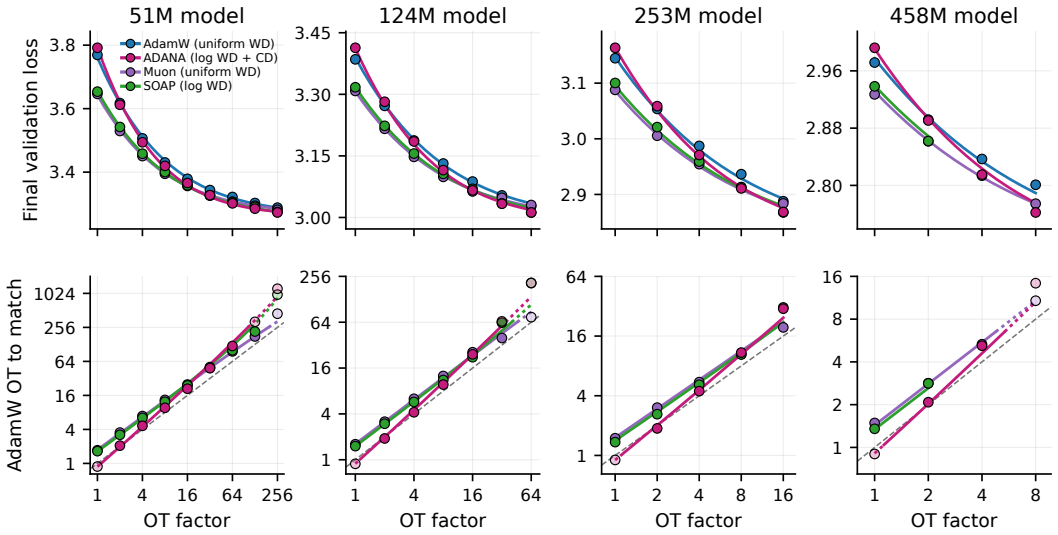

Figure 1: **ADANA outscales AdamW and becomes competitive with matrix-preconditioned optimizers as the training horizon grows.** Columns show four model sizes. Top: final validation loss at the best learning rate. Bottom: the overtraining (OT) factor AdamW requires to match that loss at the same model size. We use the better-performing weight decay schedule for each optimizer: AdamW and Muon use uniform weight decay, while SOAP and ADANA (with momentum cooldown, CD) use log-time weight decay. Curves use the jointly fitted Skaling functional form (Videau et al., 2026), with shared residual floor E and model exponent m; see Appendix H. Bottom-row markers invert measured losses through the fitted AdamW curve; curves compare fitted treatment and AdamW losses. Gray dashed lines mark equal OT. Translucent markers and dotted segments indicate comparisons that require extrapolating beyond the available AdamW measurements.

keep their memory horizons fixed throughout training. DANA instead derives a momentum schedule whose memory grows throughout training, and ADANA incorporates this schedule into an Adam-style adaptive optimizer (Ferbach et al., 2025; 2026). We compare AdamW, ADANA, Muon, and SOAP to ask how fixed memory, matrix preconditioning, and scheduled memory affect optimizer scaling across the overtraining axis.

To study this question, we compare these optimizers on four model sizes from 51M to 458M parameters at overtraining factors $f = T/(20P)$ from $1\times$ up to $256\times$ on the smallest model, where T is the number of training tokens and P is the nominal parameter count (Appendix B); $1\times$ OT corresponds to 20 tokens per parameter. For every optimizer, model size, and OT factor, we independently sweep the base learning rate. We fit each optimizer's final losses jointly across model size and training tokens and read off two slices of the fitted loss surface. The *token multiplier* fixes model size and asks how many more training tokens a baseline optimizer needs to match the indicated optimizer's loss (Qiu et al., 2025; Ferbach et al., 2026; Wen et al., 2026), which measures the savings in pretraining compute and data. The *parameter multiplier* fixes the token budget and asks how much larger the baseline model must be, which translates to savings in the memory and serving costs of the deployed model. Values above one favor the indicated optimizer.

Our contributions include the following. First, we compare optimizers across the overtraining axis rather than at a single horizon, sweeping the learning rate at every model size and OT factor and summarizing the results with joint model–token loss fits and token and parameter multipliers (Section 3). Under this comparison, optimizer rankings can reverse as training extends (Figure 1). Second, optimal hyperparameters also change along this axis: the preferred learning rate schedule switches from linear to cosine decay as the OT factor grows (Appendix D), and the best weight decay coefficient grows approximately as $\sqrt{f}$ for both uniform and log-time weight decay (Section 5). Third, we show that tuning AdamW's fixed memory at every horizon does not remove ADANA's growing advantage (Section 4). Longer horizons favor longer fixed memory, but a fixed window must also serve early training, whereas a memory schedule can keep memory short early and long late. Fourth, we

propose a form of *momentum cooldown* that benefits ADANA in particular, capping each momentum timescale at the learning rate decay timescale so that memory shortens during terminal learning rate decay. It substantially improves ADANA at long horizons while leaving AdamW nearly unchanged, and its gains compound with those of log-time weight decay (Section 6). Fifth, with both schedules, ADANA outscales AdamW at all four model sizes (Section 7), with fitted equivalent-OT slopes slightly above the exponent predicted by DANA theory on power-law random features (Ferbach et al., 2025). Muon's advantage over AdamW stays roughly constant across OT factors, while SOAP gains further at high OT; ADANA begins behind both but surpasses Muon and becomes competitive with SOAP.

2 Related Work

Long and scheduled optimizer memory. Most optimizers use fixed memory timescales throughout training where their momentum and other moving averages have constant coefficients, but several recent methods instead combine or schedule multiple timescales. AdEMAMix combines separate fast and slow momentum buffers, with distinct coefficients controlling their memory timescales (Pagliardini et al., 2024). Schedule-free methods use iterate averaging, which can be viewed as a form of memory in parameter space that preserves information from earlier optimizer updates (Defazio et al., 2024). ScheduleFree+ reports that the benefit of this parameter-averaging memory increases with training duration (Defazio, 2026). Morwani et al. (2025) connect schedule-free optimizers and AdEMAMix to accelerated SGD variants that decouple momentum decay from the weight assigned to the current gradient. Their experiments find larger gains at small batch sizes, consistent with these approaches being particularly effective in noise-dominated optimization. DANA derives a momentum timescale that grows throughout training and improves loss-scaling exponents in the power-law random features model, while ADANA incorporates this schedule into an Adam-style adaptive optimizer (Ferbach et al., 2025; 2026). Optimal hyperparameters also change with training horizon and batch size: Björck et al. (2025) show empirically that the optimal learning rate decreases predictably with token horizon, while Marek et al. (2025) find that Adam's β_2 should scale with batch size to keep the second-moment half-life fixed in tokens rather than optimizer updates.

Decoupled weight decay sets a related memory timescale for the parameters (Wang & Aitchison, 2025). Its optimal timescale depends on the data-to-model ratio (Bergsma et al., 2025a), and ADANA introduces a log-time weight decay schedule whose timescale grows throughout training (Ferbach et al., 2026).

Matrix-preconditioned optimizers. Muon, SOAP, and Shampoo can be viewed as matrix-preconditioned optimizers, in contrast to adaptive optimizers such as Adam, whose coordinatewise normalization acts as a diagonal preconditioner. In a high-dimensional least-squares model, Paquette et al. (2026) show that an idealized version of Muon provides a square-root spectral speedup for eigenmodes large enough to be resolved at the given batch size, whereas unresolved modes behave like SGD. Empirically, Qiu et al. (2025) find that Muon, SOAP, and Shampoo achieve approximately constant $1.4\times$ compute-multiplier gains over AdamW across model sizes from 190M to 1.4B when both the learning rate and weight decay are scaled appropriately.

Optimizer comparisons across training horizons. While many optimizer benchmarks compare methods at a single training horizon or fixed token-to-parameter ratio, recent studies have begun to investigate how relative optimizer performance changes across the overtraining axis. Wen et al. (2025) report that, relative to AdamW, the token multipliers for Muon and SOAP are approximately $1.2\times$–$1.4\times$ on 130M–520M models from $1\times$ to $8\times$ Chinchilla. Muon performs better through $4\times$, while SOAP catches or overtakes it at $8\times$ and in their additional $16\times$ experiments. On the 1.2B model, both token multipliers decrease to approximately $1.1\times$. Semenov et al. (2025) find that AdamW narrows and eventually reverses SOAP's short-horizon advantage as training extends. They also show that the long-horizon ordering between SOAP and AdEMAMix depends on retuning AdEMAMix's slow-memory coefficient: SOAP overtakes AdEMAMix when its shorter-horizon coefficient is reused, whereas increasing β_3 restores AdEMAMix's lead. Wen et al. (2026) report that, relative to AdamW, the token multiplier for standard Muon remains approximately $1.1\times$ across $1\times$–$8\times$ Chinchilla, whereas the token multiplier for Muon Hyperball increases across the same range from approximately $1.2\times$ to $1.3\times$. These results show that optimizer advantages can change

with training horizon and experimental treatment, while leaving their behavior under substantially greater overtraining unresolved.

3 Comparing Optimizers Across the Overtraining Axis

Throughout this paper, we use the term *memory* to refer to how long information from earlier optimizer updates remains influential.

3.1 Momentum and Weight Decay as Memory

For a momentum state with constant coefficient β and $m_0 = 0$,

$$m_t = \beta m_{t-1} + (1-\beta) g_t = (1-\beta) \sum_{s=1}^{t} \beta^{t-s} g_s. \tag{1}$$

A gradient from a updates ago is therefore weighted by β^a, giving an effective memory window $M = 1/(1-\beta)$. Fixed-memory optimizers keep this timescale constant across training steps, whereas scheduled-memory optimizers vary it during training. The same interpretation applies to exponential averages of squared gradients. ADANA, for example, averages its first and second moments with coefficient $\Delta_t = \delta/(\delta + t)$ in place of $1-\beta$, so both memory windows, $1 + t/\delta$, grow linearly with the update count. It also scales its momentum term by $\chi_t \approx t^{1-\kappa}$, so that each new gradient enters through the momentum with an effective step size proportional to $t^{-\kappa}$, where κ reflects a property of the data distribution (Ferbach et al., 2025; 2026) and determines the token-exponent advantage predicted by theory on power-law random features (Section 7.1). Our memory sweeps tune gradient averaging for Muon and second-moment averaging through β_2 for AdamW and SOAP.

Whereas momentum controls how long the optimizer state remembers past gradient information, weight decay controls how quickly the parameters forget it (Wang & Aitchison, 2025). Under decoupled weight decay, the update is $\theta_{t+1} = (1-\alpha_t)\theta_t - \eta_t d_t$, where d_t is the optimizer direction and α_t is the parameter decay fraction (Loshchilov & Hutter, 2019). For constant $\alpha_t = \alpha$, an earlier update's contribution decays as $(1-\alpha)^a$ after a steps, giving an effective memory window of approximately $1/\alpha$ updates. Scheduling weight decay changes how quickly the parameters forget earlier updates.

3.2 Experimental Setup

We compare AdamW, ADANA, Muon, and SOAP on decoder-only Transformers trained on FineWeb (Penedo et al., 2024). Our main comparisons extend to $256\times$, $64\times$, $16\times$, and $8\times$ OT for the 51M, 124M, 253M, and 458M models, respectively; SOAP's coverage on the 458M model extends to $2\times$ OT. We use a sequence length of $2{,}048$ and a global batch size of 256 sequences, so increasing OT proportionally increases the number of optimizer updates. At every setting, we sweep the base learning rate and report the lowest final validation loss. Primary comparisons use linear warmup followed by cosine decay to zero; Appendix D reports the schedule-selection experiments.

Unless otherwise stated, AdamW and SOAP use $\beta_2 = 0.98$, and Muon uses momentum coefficient $\beta = 0.98$, selected at $1\times$ OT on the 253M model. ADANA uses $\delta = 8$ and $\kappa = 0.85$, with $g_3 = 8$ selected at the same setting. DANA theory relates κ to the power-law spectrum of the data (Ferbach et al., 2025); prior ADANA experiments find that $\kappa = 0.85$ performs well across model scales and architectures (Ferbach et al., 2026). Appendices B, C, and F give the complete setup, optimizer definitions, and supporting memory sweeps.

3.3 Token and Parameter Multipliers

We quantify how optimizer advantages change with the training horizon using loss surfaces fitted jointly across model size and training tokens. For each optimizer treatment o, we fit final validation losses at all model sizes P and token budgets T with the Skaling functional form (Videau et al., 2026), $\mathcal{L}_o(P,T) = E + [A_o p^{-m/k_o} + B_o t^{-\beta_o}]^{k_o}$, where p and t are P and T in fixed reference

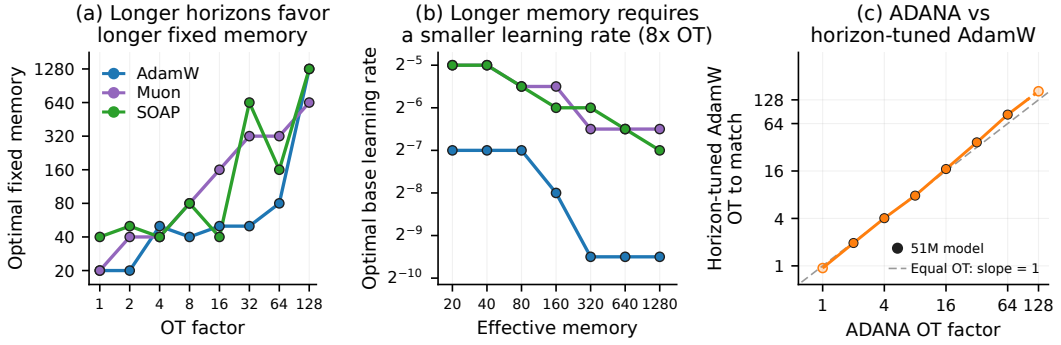

Figure 2: **Longer horizons favor longer fixed memory, but ADANA's scheduled memory still outperforms horizon-tuned AdamW at long horizons.** All panels use the 51M model and uniform weight decay. (a) Optimal effective memory after sweeping the base learning rate at each memory and OT factor. (b) At $8\times$ OT, longer memory requires a smaller learning rate. (c) The OT factor horizon-tuned AdamW requires to match ADANA's loss. ADANA uses no cooldown, and its base learning rate is tuned at each horizon. Equivalent OT is computed by piecewise-linear interpolation of horizon-tuned AdamW loss in log-OT. See Appendix F for additional results.

units. The residual floor E and model exponent m are shared across treatments (Appendix H), and the curves in Figure 1 (top) show these fits.

Two slices of these surfaces compare an optimizer o with a baseline b. At fixed model size, the *equivalent OT factor* f_b is the OT factor at which b matches o's loss at OT factor f_o, and the *token multiplier* is f_b/f_o, similar to prior work (Qiu et al., 2025; Ferbach et al., 2026; Wen et al., 2026). At a fixed token budget, the *parameter multiplier* is the model size b needs to match o's loss, divided by o's model size. For both multipliers, values above one favor o. Figure 1 (bottom) plots equivalent OT factors, Figure 4 token multipliers, and Figure 5c parameter multipliers.

We say that o *outscales* b across the overtraining axis when its token multiplier increases with the OT factor, so that its equivalent-OT curve rises faster than the equal-OT diagonal. At fixed model size, each fitted curve approaches a *high-token loss limit* $E + A_o^{k_o} p^{-m}$, and the excess loss above it decays asymptotically as $T^{-\beta_o}$; we call β_o the *token-decay exponent*. Outscaling can therefore come from a lower limit, a larger exponent, or both (Figure 5b).

Markers invert measured losses through the fitted baseline curve. Dotted curves and translucent markers indicate comparisons that require extrapolating beyond the available baseline measurements at the same model size.

4 Scheduled Memory Versus Horizon-Tuned Fixed Memory

This section asks whether tuning a fixed memory window separately at each training horizon can match a memory schedule. On the 51M model, we sweep AdamW's second-moment coefficient β_2, Muon's momentum coefficient β, and SOAP's second-moment coefficient β_2 jointly with the base learning rate at every OT factor, and report each through its effective memory window M (Section 3.1; Appendix F). Optimal memory also depends on batch size (Marek et al., 2025); Appendix J examines this dependence.

Longer horizons favor longer fixed memory (Figure 2a). From $1\times$ to $128\times$ OT, the optimal window grows from $M = 20$ to $M = 640$ for Muon and from $M = 40$ to $M = 1280$ for SOAP, while AdamW's stays at $M \leq 80$ through $64\times$ OT before reaching $M = 1280$ at $128\times$. Per-horizon tuning therefore helps: at $128\times$ OT, it improves final validation loss relative to $M = 50$ by 3.3, 3.6, and 1.3 millinats (mnats) for AdamW, Muon, and SOAP (Appendix Figure 18). This trend is an empirical counterpart to the motivation for memory schedules: spectral analyses suggest that later training resolves progressively weaker modes, whose gradient signal may benefit from longer averaging (Saxe et al., 2014; Bordelon et al., 2020).

A fixed window cannot grow with the horizon without also lengthening memory at the start of training, when gradients may change quickly. Longer memory also requires a smaller learning rate (Figure 2b), suggesting a stability cost. A fixed coefficient may therefore trade long averaging late in training against stability early in training, whereas a schedule can keep memory short early and long late. ADANA's memory window, which grows linearly with the update count (Appendix C), is shorter than the best fixed windows early in training and much longer by the end.

Horizon-tuned fixed memory does not close ADANA's advantage (Figure 2c). With uniform weight decay and no cooldown, ADANA continues to gain on AdamW with memory and learning rate tuned at every horizon, so its advantage is not explained by holding AdamW's β_2 fixed across horizons. Horizon-tuned Muon and SOAP retain roughly constant advantages over horizon-tuned AdamW (Appendix Figures 19b–c).

Muon and SOAP tolerate long fixed memory better than AdamW. Muon's optimal window roughly doubles with each doubling of OT from $4\times$ to $128\times$, and SOAP's reaches $M = 1280$ at $128\times$ OT, even though these windows also apply at the start of training. Their optimal learning rates also fall less steeply with memory than AdamW's (Figure 2b).

5 Weight Decay Across Training Horizons

Uniform weight decay uses a coefficient that stays constant across training updates, $\lambda_t = c_{\text{uniform}}(f)/S$, where S is the total update count. Log-time weight decay instead decreases this coefficient as $\lambda_t = c_{\log}(f)/(\tau + t)$ (Ferbach et al., 2026), where τ is an initial offset. In both cases, the parameter decay fraction is $\alpha_t = s_t\lambda_t$, where $s_t = \eta_t/\eta$ is the learning rate schedule normalized by the base learning rate η.

The best weight decay coefficient grows approximately as $\sqrt{f}$ for both schedules. For AdamW with uniform weight decay, Bergsma et al. (2025a) find that the optimal AdamW timescale scales as $(T/P)^{-0.527}$, which corresponds to $c_{\text{uniform}}(f) \propto f^{0.527}$ (Appendix E). Jointly sweeping the base learning rate and weight decay coefficient for every optimizer on the 51M model, we fit an exponent of 0.527 for uniform weight decay, matching their AdamW exponent, and 0.473 for log-time weight decay, which they did not study. We therefore use $c_{\text{uniform}}(f) = 8\sqrt{f}$ and $c_{\log}(f) = 2\sqrt{f}$ rather than tuning the coefficient at every horizon. Muon is the exception for log-time weight decay: its performance under this rule is nonmonotonic across OT factors, suggesting a poorly matched coefficient.

Under these rules, log-time weight decay increasingly benefits ADANA as overtraining grows and also benefits SOAP, particularly at high OT, while it has little consistent effect on AdamW. These comparisons use independently tuned learning rates without momentum cooldown (Appendix E). Accordingly, Figure 1 and the outscaling comparisons in Section 7 use uniform weight decay for AdamW and Muon, and log-time weight decay for SOAP and ADANA.

6 Momentum Cooldown for Growing Memory

ADANA's memory is longest near the end of training, when the learning rate approaches zero and the optimizer may benefit from responding more quickly to recent gradients. We introduce momentum cooldown to cap the averaging window during this terminal decay while preserving its growth earlier in training. A similar idea of reducing momentum at the end of training has been used for Muon in nanochat (Karpathy, 2025).

We define the current learning rate decay timescale as $\tau_{\eta,t} = \eta_t/|\eta_{t+1} - \eta_t|$, taking $\tau_{\eta,t} = \infty$ when the learning rate is constant or increasing. For any momentum state with coefficient β_t, we define its original memory timescale as $\tau_{\text{mom},t} = 1/(1-\beta_t)$. In momentum cooldown, we replace this timescale with

$$\tau_{\text{CD},t} = \max(1, \min\{\tau_{\text{mom},t}, \tau_{\eta,t}\}), \qquad \beta_{\text{CD},t} = 1 - \frac{1}{\tau_{\text{CD},t}}. \tag{2}$$

See Appendix G for the complete definitions and ablations.

Figure 3a shows how cooldown changes ADANA's late-training loss trajectory. At $32\times$ OT on the 51M model with log-time weight decay, the loss nearly plateaus without cooldown, while cooldown

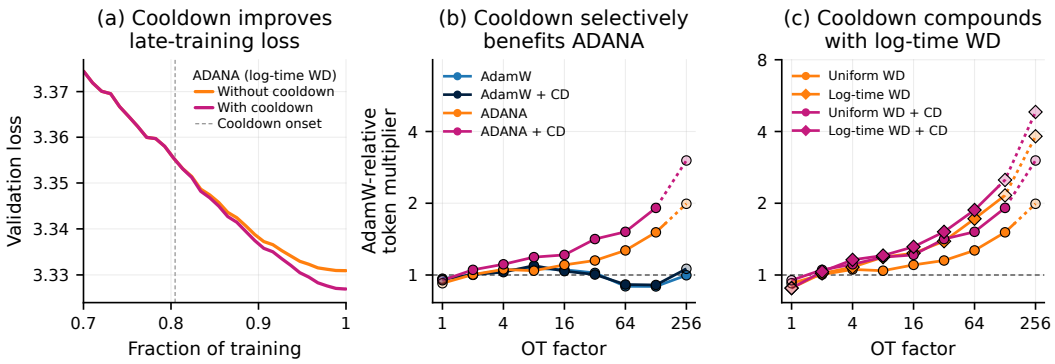

Figure 3: **Momentum cooldown selectively benefits ADANA and compounds with log-time weight decay.** All panels use the 51M model. (a) Without cooldown, ADANA's loss nearly plateaus during terminal learning rate decay; momentum cooldown causes a late-training improvement. We show the final 30% of training at $32\times$ OT with log-time weight decay, using the learning rate that is best without cooldown. The dashed vertical line marks cooldown onset. (b) Under uniform weight decay, cooldown has little consistent effect on AdamW but improves ADANA at long horizons. (c) Log-time weight decay and cooldown provide complementary gains, with their combination yielding the largest long-horizon improvement. Panels (b)–(c) show token multipliers relative to AdamW with uniform weight decay, with learning rates independently tuned for each treatment and horizon. See Appendix G for additional results.

causes a late-training improvement. Both runs use the learning rate selected without cooldown, and their validation losses agree until cooldown activates.

With learning rates independently tuned for each treatment, Figure 3b shows that cooldown has little consistent effect on fixed-memory AdamW but substantially improves ADANA at long horizons. Figure 3c shows that these gains compound with those from log-time weight decay. Across the 51M, 124M, and 253M models, the combined treatment outperforms ADANA with uniform weight decay and no cooldown (Appendix Figure 28c). Its growing token-efficiency advantage over AdamW motivates the outscaling analysis in Section 7. We therefore use ADANA with log-time weight decay and momentum cooldown in the subsequent strongest-treatment comparisons. We leave additional analysis of the mechanism behind this benefit and exploration of additional momentum cooldown rules to future work.

7 Outscaling Across the Overtraining Axis

This section compares the optimizers under the treatments selected in Sections 5 and 6 (Figures 1, 4, and 5); Appendix I repeats these comparisons with uniform weight decay and no cooldown.

7.1 ADANA outscales AdamW

The DANA momentum schedule was originally derived to improve loss-scaling exponents relative to SGD in the power-law random features (PLRF) model (Ferbach et al., 2025). In this analysis, DANA accelerates an effective optimization clock, which reparameterizes training time according to the rate at which spectral modes are learned. When the step size for the momentum channel scales as $t^{-\kappa}$, the momentum contribution to this clock grows as

$$\left(\int_0^t s^{-\kappa/2}\, ds\right)^2 \propto t^{2-\kappa},$$

whereas the SGD clock grows linearly with t. Consequently, matching DANA after t updates requires SGD to train for a horizon proportional to $t^{2-\kappa}$. In terms of our equivalent-OT metric, this would give DANA an equivalent-OT exponent of $2-\kappa$ relative to SGD.

We find that this theoretical prediction transfers remarkably well to our Transformer experiments: ADANA outscales AdamW with roughly the $2-\kappa$ exponent (Figure 5a). Against

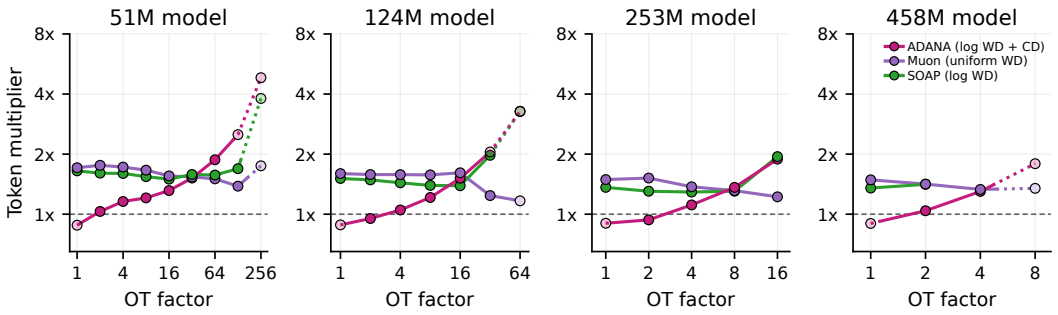

Figure 4: **ADANA's token-efficiency advantage over AdamW grows with overtraining; Muon's is roughly constant, while SOAP gains further at high OT.** Token multipliers for the treatments and losses in Figure 1, relative to AdamW with uniform weight decay. A token multiplier of two means that AdamW requires twice as many tokens to reach the same validation loss at the same model size. Markers invert measured treatment losses through the fitted AdamW curves.

AdamW with uniform weight decay, the fitted equivalent-OT curves have log–log slopes of 1.18–1.24 over their plotted ranges at all four model sizes, slightly above $2 - \kappa = 1.15$ for our choice $\kappa = 0.85$. This comparison uses log-time weight decay and momentum cooldown for ADANA and holds similarly whether AdamW uses uniform or log-time weight decay (Appendix Figure 40). At 51M, Figure 5b shows that ADANA's fitted token-decay exponent advantage remains positive across the high-token loss limits considered.

7.2 Comparison with matrix-preconditioned optimizers

Figure 4 compares Muon and SOAP with AdamW. Muon provides token multipliers of approximately $1.2\times$–$1.8\times$, with a roughly constant advantage across OT factors. SOAP provides approximately $1.3\times$–$1.7\times$ across most horizons but gains further at high OT. Its token multiplier reaches approximately $3.8\times$ at $256\times$ OT on the 51M model and $3.3\times$ at $64\times$ OT on the 124M model; these two comparisons require extrapolating the AdamW baseline.

ADANA starts substantially behind: at $2\times$ OT, its token multiplier is $0.59\times$–$0.73\times$ relative to Muon and $0.64\times$–$0.79\times$ relative to SOAP across the four model sizes (Appendix Figure 38). As the training horizon increases, ADANA closes these gaps, surpassing Muon and becoming competitive with SOAP at the highest OT factors we test. ADANA appears to outscale Muon, while SOAP's own high-OT gains make their relative scaling less clear.

8 Discussion and Limitations

Optimizer evaluation should treat training horizon as part of the experimental design. An optimizer selected near the compute-optimal horizon may not remain strongest when the same model is trained substantially longer, and its preferred learning rate schedule, weight decay treatment, and memory can also change across the overtraining axis. Long-horizon comparisons therefore require scaling rules for hyperparameters that are often selected using experiments at only one horizon.

Large token multipliers can correspond to much smaller parameter multipliers (Figure 5c). For example, at $64\times$ OT on the 51M model, AdamW needs $1.87\times$ as many training tokens to match ADANA at the same model size, but only a $1.05\times$ larger model at the same token budget. Near a fitted high-token loss limit, a small loss improvement requires many more tokens but only slightly more parameters. However, in scenarios where model deployment costs dominate or parameter count is tightly constrained by memory or latency requirements, a modest change in the parameter count can have substantial benefits.

Our results suggest complementary roles for preconditioning and scheduled memory. In our experiments, Muon and SOAP provide large advantages over AdamW across most horizons, while ADANA's relative efficiency increases with training horizon.

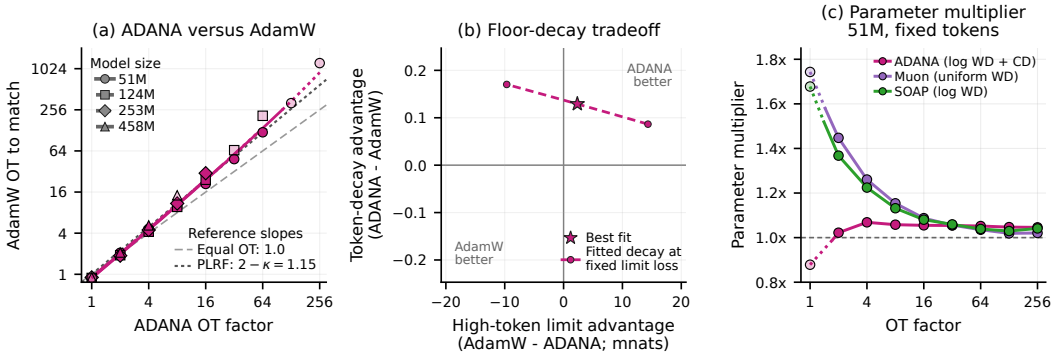

Figure 5: **ADANA's fitted token-decay exponent advantage persists across the high-token loss limits considered.** All panels use AdamW with uniform weight decay as the baseline. (a) Equivalent AdamW OT for ADANA with log-time weight decay and momentum cooldown across four model sizes, compared with the $2 - \kappa = 1.15$ reference from DANA theory (Ferbach et al., 2025). (b) At 51M, fixing the high-token loss-limit advantage and refitting the remaining parameters traces its tradeoff with the token-decay exponent advantage. Positive values on either axis favor ADANA; the star marks the best fit. (c) Large long-horizon token multipliers correspond to much smaller parameter multipliers at fixed training tokens. The parameter multiplier is the AdamW model size required to match a 51M treatment's loss at the same token budget, divided by 51M. Treatments match Figure 1.

Our main optimizer comparisons use a global sequence batch of 256. Appendix J reports a fixed-token experiment at $8\times$ OT in which ADANA begins losing efficiency at smaller batch sizes than AdamW, despite jointly optimizing the learning rate with β_2 for AdamW and g_3 for ADANA. Increasing batch size simultaneously reduces update count and gradient noise, while the other optimizer hyperparameters remain fixed, so this experiment cannot separate these effects.

Our experiments cover one model family and dataset, with model sizes from 51M to 458M parameters and one seed per configuration. The maximum tested OT factor decreases with model size, and some high-OT token multipliers require baseline extrapolation. The fixed-memory and batch-size studies are limited to the 51M model. Establishing how these findings transfer across larger models, architectures, and datasets remains important future work.

9 Conclusion

Optimizer rankings and optimal hyperparameter settings change as models are trained beyond the Chinchilla reference horizon. ADANA's scaling advantage over AdamW persists after tuning AdamW's fixed memory separately at each training horizon. Log-time weight decay and momentum cooldown further amplify this advantage, with fitted equivalent-OT slopes slightly above the $2 - \kappa$ reference from DANA theory on power-law random features. Muon provides roughly constant gains over AdamW, while SOAP gains further at high OT. ADANA begins behind both but surpasses Muon and becomes competitive with SOAP as training increases. Treating training horizon as an essential experimental axis reveals optimizer behavior hidden by single-horizon comparisons and suggests that scheduling optimizer memory is a promising route to methods whose relative efficiency improves with additional training.

AI USE STATEMENT

In this work, we used generative AI tools to help implement and run experiments, write and edit code, analyze results, and create figures. We also used them to search for and summarize related literature, format and check references, and draft and edit text. Generating synthetic datasets, translation, and qualitative or thematic data analysis are not applicable to this work. We reviewed all code, revised and approved the main text sentence by sentence, and reviewed the figures and analyses. We read the cited papers ourselves and checked the citations against them. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*, 2020. URL https://arxiv.org/abs/2002.09018.

Jacob Austin, Sholto Douglas, Roy Frostig, Anselm Levskaya, Charlie Chen, Sharad Vikram, Federico Lebron, Peter Choy, Vinay Ramasesh, Albert Webson, and Reiner Pope. How to scale your model: A systems view of LLMs on TPUs. https://jax-ml.github.io/scaling-book/, 2025. Online book.

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. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. doi: 10.52202/085713-4171. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b5f78a17a94da3e34c935515d1b6adae-Abstract-Conference.html.

Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Straight to zero: Why linearly decaying the learning rate to zero works best for LLMs. In *International Conference on Learning Representations*, 2025b. URL https://arxiv.org/abs/2502.15938.

Shane Bergsma, Nolan Dey, and Joel Hestness. Predicting training re-evaluation curves enables effective data curriculums for LLMs. In *International Conference on Learning Representations*, 2026a. URL https://arxiv.org/abs/2509.25380.

Shane Bergsma, Bin Claire Zhang, Nolan Dey, Shaheer Muhammad, Gurpreet Gosal, and Joel Hestness. Scaling with collapse: Efficient and predictable training of LLM families. In *International Conference on Learning Representations*, 2026b. URL https://arxiv.org/abs/2509.25087.

Song Bian, Minghao Yan, and Shivaram Venkataraman. Scaling inference-efficient language models. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2501.18107.

Johan Björck, Alon Benhaim, Vishrav Chaudhary, Furu Wei, and Xia Song. Scaling optimal LR across token horizons. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=WYL4eFLcxG.

Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. Spectrum dependent learning curves in kernel regression and wide neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1024–1034. PMLR, 2020. URL https://proceedings.mlr.press/v119/bordelon20a.html.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs. https://github.com/jax-ml/jax, 2018. Version 0.6.2.

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. URL https://arxiv.org/abs/2503.09799.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. URL https://arxiv.org/abs/1604.06174.

Aaron Defazio. Schedulefree+: Scaling learning-rate-free and schedule-free learning to large language models. *arXiv preprint arXiv:2605.19095*, 2026. URL https://arxiv.org/abs/2605.19095.

Aaron Defazio, Xingyu Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The road less scheduled. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15682.

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. URL https://arxiv.org/abs/2505.01618.

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, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12666–12700. PMLR, 2024. URL https://proceedings.mlr.press/v235/everett24a.html.

Damien Ferbach, Katie Everett, Gauthier Gidel, Elliot Paquette, and Courtney Paquette. Dimension-adapted momentum outscales SGD. *arXiv preprint arXiv:2505.16098*, 2025. URL https://arxiv.org/abs/2505.16098.

Damien Ferbach, Courtney Paquette, Gauthier Gidel, Katie Everett, and Elliot Paquette. Logarithmic-time schedules for scaling language models with momentum. *arXiv preprint arXiv:2602.05298*, 2026. URL https://arxiv.org/abs/2602.05298.

Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, Rui Xin, Marianna Nezhurina, Igor Vasiljevic, Jenia Jitsev, Luca Soldaini, Alexandros G. Dimakis, Gabriel Ilharco, Pang Wei Koh, Shuran Song, Thomas Kollar, Yair Carmon, Achal Dave, Reinhard Heckel, Niklas Muennighoff, and Ludwig Schmidt. Language models scale reliably with over-training and on downstream tasks. *arXiv preprint arXiv:2403.08540*, 2024. URL https://arxiv.org/abs/2403.08540.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. URL https://proceedings.mlr.press/v80/gupta18a.html.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. *arXiv preprint arXiv:2405.18392*, 2024. URL https://arxiv.org/abs/2405.18392.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX. https://github.com/google/flax, 2025. Version 0.10.6.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén 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*, volume 35, pp. 30016–30030, 2022. URL https://papers.neurips.cc/paper_files/paper/2022/hash/c1e2faff6f588870935f114ebe04a3e5-Abstract-Conference.html.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zheng Leng Thai, Kaihuo Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. MiniCPM: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024. URL https://arxiv.org/abs/2404.06395.

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/.

Norman P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, 2017. doi: 10.1145/3079856.3080246.

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

Andrej Karpathy. nanochat: The best ChatGPT that $100 can buy. https://github.com/karpathy/nanochat, 2025. Software repository.

Mikail Khona, Aditya Vavre, Boxiang Wang, Deyu Fu, Hao Wu, Mike Chrzanowski, Bryan Catanzaro, Dheevatsa Mudigere, Jeff Pool, Michael Lightstone, Mohammad Shoeybi, Mostofa Patwary, Nima Tajbakhsh, and Tijmen Blankevoort. SOAP, Muon, and beyond: Pushing LLM pretraining scales. *arXiv preprint arXiv:2607.20548*, 2026. URL https://arxiv.org/abs/2607.20548.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025. URL https://arxiv.org/abs/2502.16982.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32431–32454, 2024. URL https://proceedings.mlr.press/v235/liu24ce.html.

Ziyin Liu, Zhikang T. Wang, and Masahito Ueda. LaProp: Separating momentum and adaptivity in Adam. *arXiv preprint arXiv:2002.04839*, 2020. doi: 10.48550/arXiv.2002.04839. URL https://arxiv.org/abs/2002.04839.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

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, 2025. URL https://arxiv.org/abs/2507.07101.

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. URL https://arxiv.org/abs/2512.22382.

Depen Morwani, Itai Shapira, Nikhil Vyas, Eran Malach, Sham Kakade, and Lucas Janson. A new perspective on Shampoo's preconditioner. *arXiv preprint arXiv:2406.17748*, 2024. URL https://arxiv.org/abs/2406.17748.

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. URL https://arxiv.org/abs/2502.02431.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models. *arXiv preprint arXiv:2305.16264*, 2023. URL https://arxiv.org/abs/2305.16264.

Matteo Pagliardini, Pierre Ablin, and David Grangier. The AdEMAMix optimizer: Better, faster, older. *arXiv preprint arXiv:2409.03137*, 2024. URL https://arxiv.org/abs/2409.03137.

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*, volume 37, pp. 16459–16537, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/1dccfc3ee01871d05e33457c61037d59-Abstract-Conference.html.

Elliot Paquette, Noah Marshall, Lucas Benigni, Guangyuan Wang, Atish Agarwala, and Courtney Paquette. Phases of Muon: When Muon eclipses SignSGD. *arXiv preprint arXiv:2605.09552*, 2026. URL https://arxiv.org/abs/2605.09552.

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 *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2406.17557.

Project Apertus et al. Apertus: Democratizing open and compliant LLMs for global language environments, 2025. URL https://arxiv.org/abs/2509.14233.

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*, 2025. URL https://arxiv.org/abs/2512.05620.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language-models.pdf.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020. doi: 10.1109/SC41405.2020.00024.

Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2401.00448.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations*, 2014. URL https://arxiv.org/abs/1312.6120.

Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. Benchmarking optimizers for large language model pretraining. *arXiv preprint arXiv:2509.01440*, 2025. URL https://arxiv.org/abs/2509.01440.

Ishaan Shah, Anthony M. Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J. Shah, Khoi Nguyen, Kurt Smith, Michael Callahan, Michael Pust, Mohit Parmar, Peter Rushton, Platon Mazarakis, Ritvik Kapila, Saurabh Srivastava, Somanshu Singla, Tim Romanski, Yash Vanjani, and Ashish Vaswani. Practical efficiency of Muon for pretraining. *arXiv preprint arXiv:2505.02222*, 2025. URL https://arxiv.org/abs/2505.02222.

Hao-Jun Michael Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat. A distributed data-parallel PyTorch implementation of the distributed Shampoo optimizer for training neural networks at-scale. *arXiv preprint arXiv:2309.06497*, 2023. URL https://arxiv.org/abs/2309.06497.

Egor Shulgin, Dimitri von Rütte, Tianyue H. Zhang, Niccolò Ajroldi, Bernhard Schölkopf, and Antonio Orvieto. Deriving hyperparameter scaling laws via modern optimization theory. *arXiv preprint arXiv:2603.15958*, 2026. URL https://arxiv.org/abs/2603.15958.

The JAX Authors. Splash attention: Sparse flash attention for TPU. https://github.com/jax-ml/jax, 2023. JAX Pallas Splash Attention implementation.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. URL https://arxiv.org/abs/2307.09288.

Mathurin Videau, Badr Youbi-Idrissi, David Lopez-Paz, and Kartik Ahuja. Skaling: Chinchilla's exponents meet kaplan's coupling. *arXiv preprint arXiv:2608.07222*, 2026. doi: 10.48550/arXiv.2608.07222. URL https://arxiv.org/abs/2608.07222.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Position: Will we run out of data? Limits of LLM scaling based on human-generated data. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 49523–49544, 2024. URL https://proceedings.mlr.press/v235/villalobos24a.html.

Alexandra Volkova, Mher Safaryan, Christoph H. Lampert, and Dan Alistarh. Towards robust scaling laws for optimizers. *arXiv preprint arXiv:2602.07712*, 2026. URL https://arxiv.org/abs/2602.07712.

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. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=IDxZhXrpNf.

Xi Wang and Laurence Aitchison. How to set AdamW's weight decay as you scale model and dataset size. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2405.13698.

Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them. *arXiv preprint arXiv:2509.02046*, 2025. URL https://arxiv.org/abs/2509.02046.

Kaiyue Wen, Xingyu Dang, Kaifeng Lyu, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them II: Hyperball optimization. *arXiv preprint arXiv:2606.16899*, 2026. URL https://arxiv.org/abs/2606.16899.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

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*, volume 34, 2021. URL https://arxiv.org/abs/2203.03466.

A Extended Related Work

Training beyond compute-optimal horizons. Data-constrained scaling studies characterize how repeated data changes the benefit of continued training when the available unique data are limited (Muennighoff et al., 2023). Warmup-Stable-Decay schedules separate an extensible stable-training phase from terminal learning rate decay (Hu et al., 2024), while reusable training trajectories allow scaling behavior to be evaluated across multiple training durations without restarting each run from initialization (Hägele et al., 2024). Together, these works treat training horizon as an experimental axis distinct from model size.

Scaling-law functional forms. For scaling analyses that vary both model size and training horizon, Videau et al. (2026) propose a functional form that couples model size and training data through an interaction exponent. We use this functional form with a model exponent shared across optimizers, similar to Volkova et al. (2026), who show that sharing scaling-law parameters across optimizers improves identifiability.

Hyperparameter transfer across scale. The maximal-update parameterization and μTransfer enable zero-shot hyperparameter transfer across model width by prescribing how initialization and learning rates should scale (Yang et al., 2021). Subsequent work derives optimizer-dependent width exponents, develops depth-aware parameterizations, and extends hyperparameter transfer across modules, width, depth, batch size, and training duration (Everett et al., 2024; Dey et al., 2025; Mlodozeniec et al., 2025). For matrix-preconditioned optimizers, preserving gains across model scale requires optimizer-appropriate learning rate and weight decay transfer (Qiu et al., 2025). Complementary theoretical work derives coupled scaling rules for learning rate, momentum, and batch size from the iteration or token budget (Shulgin et al., 2026).

Optimizer memory and weight decay timescales. Recent work uses normalized optimizer timescales to align training curves across model scales and characterize how information from different portions of training remains influential (Bergsma et al., 2026b;a). In particular, Bergsma et al. (2025a) fit a normalized AdamW weight decay timescale proportional to $(D/N)^{-0.527}$, where D is the token horizon and N is model size. At fixed model size, this implies that the corresponding memory window measured in tokens grows approximately as $D^{0.473}$. This provides additional context for studying momentum and weight decay as horizon-dependent memory timescales.

Preconditioner order and scalable matrix optimizers. Adam accumulates momentum before applying coordinatewise adaptive normalization, whereas LaProp applies normalization before momentum accumulation (Kingma & Ba, 2015; Liu et al., 2020). Shampoo introduced structured matrix-valued preconditioning, with subsequent work developing scalable and distributed implementations (Gupta et al., 2018; Anil et al., 2020; Shi et al., 2023; Morwani et al., 2024). SOAP applies Adam-like updates in Shampoo's evolving eigenbasis, while Muon approximately orthogonalizes matrix-valued updates (Vyas et al., 2025; Jordan et al., 2024). Recent studies further examine the practical implementation and large-scale behavior of these optimizers (Liu et al., 2025; Shah et al., 2025; Khona et al., 2026).

B EXPERIMENTAL DETAILS

Unless explicitly stated otherwise, all experiments use the model family, initialization, data, training, and evaluation protocols defined in this section.

B.1 NOTATION, ARCHITECTURE, AND MODEL SIZES

Table 1 defines the notation used for the model family and for nominal compute accounting.

Table 1: Notation used for the architecture, training budget, and nominal compute.

Symbol	Definition
H	Number of attention heads
D	Model dimension
D_{head}	Dimension of each attention head
N	Number of Transformer blocks
F	Feed-forward intermediate dimension
L	Sequence length in tokens
V	Padded vocabulary size
B	Global batch size in sequences
S	Number of optimizer updates
$T = SBL$	Number of training tokens
P	Nominal parameter count used for scaling and compute accounting
P_{nonemb}	Number of non-embedding parameters
P_{train}	Exact trainable parameter count, including both vocabulary matrices and learned normalization scales
$C = 6PT$	Nominal training compute in FLOPs
$\text{TPP} = T/P$	Training tokens per nominal parameter
$f = T/(20P) = \text{TPP}/20$	OT factor: training-token horizon in multiples of 20 tokens per nominal parameter

We use decoder-only causal Transformers. The architecture, initialization, and width, depth, and attention co-scaling setup broadly follow the Enoki ladder in Ferbach et al. (2026). Relative to that setup, we replace GeLU with SwiGLU, LayerNorm with RMSNorm, and QK-LayerNorm with QK-RMSNorm, and use a constant token embedding initialization. The resulting Transformer blocks are architecturally similar to the Llama 2 style architecture (Touvron et al., 2023) used in Qiu et al. (2025).

The co-scaling rule, introduced in Charles et al. (2025) and used in Ferbach et al. (2026), is

$$D_{\text{head}} = 64, \qquad D = 64H, \qquad N = \lfloor 3H/4 \rfloor, \qquad F = 4D,$$

Table 2: Model architecture.

Component	Choice
Block normalization	Pre-RMSNorm; learned scale, no bias; $\epsilon = 10^{-6}$
Attention	Multi-head attention with $D_{\text{head}} = 64$
Q/K normalization	QK-RMSNorm before RoPE; learned scale, no bias; $\epsilon = 10^{-6}$
Positional encoding	RoPE with base 10,000
Activation	SwiGLU
Tokenizer	GPT-2 with vocabulary padded to $V = 50{,}304$
Embedding / readout tying	Untied
Biases	None
Dropout	None
Nominal parameter count	$P = 16ND^2 + DV$

The nominal count includes the readout matrix but omits the input embedding matrix, although the two are untied. Kaplan et al. (2020) found that excluding embedding parameters produced cleaner scaling trends across model depths. We nevertheless include the readout because it is applied as a dense vocabulary projection, whereas the input embedding is accessed by lookup.

Table 3: Model sizes used in our experiments. Model names use rounded nominal parameter counts $P = 16ND^2 + DV$. Non-embedding and exact trainable parameter counts include learned normalization scales.

Model	D	N	F	P_{nonemb}	P	P_{train}	$S_{1\times}$
51M	512	6	2,048	25,173,248	50,921,472	76,684,544	1,942
124M	768	9	3,072	84,950,400	123,568,128	162,217,344	4,713
253M	1,024	12	4,096	201,353,728	252,837,888	304,376,320	9,645
458M	1,280	15	5,120	393,257,600	457,605,120	522,035,840	17,456

Throughout the paper, model size labels such as 51M and 253M refer to rounded nominal parameter counts P, not exact trainable parameter counts.

We use overtraining (OT) factor as a concise convention for reporting training horizons: $1\times$ OT denotes 20 training tokens per nominal parameter, $2\times$ denotes 40, and so forth. The 20-tokens-per-parameter heuristic is motivated by Hoffmann et al. (2022), but we do not assume that this horizon is compute-optimal for every optimizer, model family or dataset. For each model, the $1\times$ horizon uses $S_{1\times} = \lfloor 20P/(BL) \rfloor$ optimizer updates. We report the realized token count after this integer rounding.

Following Hoffmann et al. (2022) and Austin et al. (2025), we approximate training compute as

$$C = 6PT,$$

where P is the nominal parameter count and T is the number of training tokens. This first-order estimate counts approximately $2PT$ FLOPs for the forward pass and $4PT$ FLOPs for the backward pass through the parameterized matrix multiplications.

B.2 Initialization

Following the ScaledGPT initialization used in Ferbach et al. (2026), ordinary projections use fan-in scaling, while attention and feed-forward residual-output projections receive an additional $1/\sqrt{2N}$ depth factor. This depth correction follows the initialization used in GPT-2 (Radford et al., 2019). Because residual-output projections are initialized independently with zero-mean weights, distinct residual increments have zero cross-covariance in expectation over initialization. Scaling each of the $2N$ residual-output projections by $1/\sqrt{2N}$ therefore keeps the expected aggregate variance contributed by the residual branches of order one as depth changes. We initialize token embeddings separately with the width-independent constant standard deviation 0.02. Residual branches are added to the residual stream with coefficient one; no additional residual multiplier is applied.

All matrix entries are drawn independently from untruncated, zero-mean Gaussian distributions. Table 4 reports their standard deviations; learned normalization scales are initialized to one.

Table 4: Initialization standard deviations.

Parameter	Standard deviation
Token embedding	0.02
Query, key, and value projections	$D^{-1/2}$
Attention output projection	$(2ND)^{-1/2}$
Feed-forward input projection	$D^{-1/2}$
SwiGLU gate projection	$D^{-1/2}$
Feed-forward output projection	$(2NF)^{-1/2}$
Readout matrix	$D^{-1/2}$
Learned normalization scales	0 (constant value 1)

B.3 Common training controls

Table 5 lists the training controls shared across the primary experiments unless explicitly stated otherwise. For an experiment with S optimizer updates and tuned peak learning rate η, every candidate learning rate schedule uses the same linear warmup.

Table 5: Common training controls.

Control	Default
Sequence length	$L = 2{,}048$ tokens
Global batch size	$B = 256$ sequences
Gradient clipping	Global ℓ_2 norm of 1.0, applied after gradient accumulation
Learning rate warmup	Linear from 0.01η to η over $W = \max\{100, \min\{\lfloor S/50 \rfloor, 5{,}000\}\}$ updates

The learning rate schedule after warmup and the weight decay treatment are selected experimentally in Appendices D and E, respectively.

B.4 Dataset, training order, seeds, and evaluation

We use the `sample/350BT` subset of FineWeb (Penedo et al., 2024) at revision `9bb295ddab0e05d785b879661af7260fed5140fc`. We encode each document with the GPT-2 tokenizer (Radford et al., 2019) and append one GPT-2 end-of-text token. We use source files 0–507 for training and reserve source files 508–509 for validation. The resulting training set contains 360.5 billion tokens. Training and validation are drawn from disjoint FineWeb source files; we do not perform additional deduplication across the two partitions.

Training follows a fixed sequential order through the tokenized corpus. End-of-text tokens separate documents, and training sequences may span document boundaries. We do not shuffle the training data. All experiments begin at the start of the corpus and use this same ordering. Consequently, experiments with the same global batch size and training horizon process the same tokens in the same optimizer updates.

Unless explicitly stated otherwise, every hyperparameter configuration starts from a fresh initialization with random seed 42. The training order is fixed independently of the seed, so configurations at the same model and training coordinate share both their initialization and their ordered training data. We use one seed per configuration for the hyperparameter sweeps in this paper. These matched comparisons reduce nuisance variation between nearby configurations, but they do not estimate variation across random initializations.

From the two held-out source files, we construct one fixed validation set containing 1,920 sequences and 3,932,160 scored next-token targets. Every model is evaluated on the same targets in the same order. We report mean per-token next-token cross-entropy over the complete validation set, in nats

per token. Loss differences and fit errors are reported in millinats (mnats), with $1\,\mathrm{mnat} = 10^{-3}$ nats. This full-set loss is used for both hyperparameter selection and reported loss comparisons.

B.5 Hyperparameter sweep interiority

Throughout the paper, we assess hyperparameter sweeps using the same boundary-interiority criterion. We regard a one-dimensional sweep as interior only when its minimum and the complete set of points within one mnat of that minimum lie away from both tested boundaries. For nested sweeps, we assess the inner learning rate sweep independently at every value of the outer hyperparameter before interpreting regret or selecting a minimum along the outer axis. We then apply the same 1-mnat interiority criterion to the outer axis.

B.6 Hardware, numerical precision, and parallel execution

We implement the models in JAX (Bradbury et al., 2018) and Flax (Heek et al., 2025) and train on TPU v4-8 and v6e-4 accelerators (Jouppi et al., 2017) using causal Splash attention with 512-token kernel tiles and a fused backward pass.

We use mixed-precision training. Model parameters and floating-point optimizer states are stored in `float32`; parameter gradients and RMSNorm statistics are computed in `float32`; and model activations and the operands of dense and attention matrix multiplications generally use `bfloat16`. The JAX Splash implementation uses `float32` for selected attention accumulations (The JAX Authors, 2023). The training loss materializes the vocabulary logits and computes full-vocabulary cross-entropy. For validation, logits are promoted from `bfloat16` to `float32` before computing cross-entropy and device-local loss sums; the ordered batch sums are then combined on the host using `float64` summation.

For gradient accumulation, we divide the global batch contiguously into equal microbatches. We average the resulting gradients, apply global-norm clipping to the accumulated batch-mean gradient, and then perform one optimizer update. Gradient accumulation therefore preserves the global batch size, data ordering, and number of optimizer updates. It produces the same batch-mean gradient in exact arithmetic, although the different reduction order can introduce small finite-precision differences.

When activation rematerialization is enabled, we apply it independently to each Transformer block (Chen et al., 2016). Intermediate activations within the block are discarded after the forward pass and recomputed during the backward pass, reducing activation memory at the cost of additional computation.

For configurations requiring parameter sharding, we use ZeRO Stage 3 fully sharded data parallelism over a one-dimensional TPU mesh (Rajbhandari et al., 2020). Model parameters, gradients, and optimizer states are partitioned across devices rather than replicated.

C Optimizer definitions and implementation details

C.1 Common update wrapper

Let g_t denote the gradient averaged over the global batch after combining any accumulated microbatch gradients. The optimizer receives the globally clipped gradient

$$\widetilde{g}_t = \frac{g_t}{\max(1, \|g_t\|_2/G)}, \qquad G = 1. \tag{3}$$

We write the scheduled base learning rate as $\eta_t = \eta s_t$, where η is the tuned peak learning rate and $s_t \in [0, 1]$ is a dimensionless schedule multiplier that includes warmup and decay. We tune a single base learning rate without additional layerwise learning rate tuning. For parameter group i, we write

$$\eta_i = \eta r_i, \qquad \eta_{t,i} = s_t \eta_i, \tag{4}$$

where r_i is a fixed parameter-group multiplier. We use $r_i = 1$ throughout, except for Muon's Adam-updated parameters, which use $r_i = 1.6$.

Following decoupled weight decay (Loshchilov & Hutter, 2019), we implement weight decay as a separate parameter update term rather than as an L_2 penalty in the training objective. Given an optimizer-specific direction $d_{t,i}$, the common parameter update is

$$\theta_{t+1,i} = \theta_{t,i} - s_t \eta_i d_{t,i} - s_t \lambda_{t,i} \theta_{t,i}. \tag{5}$$

The data and decay terms are computed from $\theta_{t,i}$ and applied in the same update. Decay is therefore not added to the gradient, accumulated in the optimizer moments, or transformed by an optimizer preconditioner. It follows the normalized schedule s_t, but is independent of the peak learning rate and the parameter-group multiplier.

For the uniform WD schedule,

$$\lambda_{t,i} = m_i^{\mathrm{WD}} \frac{c}{S}, \tag{6}$$

where S is the number of optimizer updates, c is the weight decay coefficient, and $m_i^{\mathrm{WD}} \in \{0, 1\}$ is the decay mask. In the absence of a data update, a decayed parameter is therefore multiplied at update t by $1 - s_t c/S$.

C.2 AdamW

We use AdamW (Loshchilov & Hutter, 2019; Kingma & Ba, 2015). Adam maintains exponential moving averages of the first and second elementwise moments of the clipped gradient,

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \widetilde{g}_t, \tag{7}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \widetilde{g}_t^{\odot 2}, \tag{8}$$

with $m_0 = v_0 = 0$. We apply the standard bias corrections

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}, \qquad \widehat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{9}$$

and define the Adam direction elementwise as

$$d_t = \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}. \tag{10}$$

We use $\epsilon = 10^{-15}$. Substituting this direction into Equation 5 gives

$$\theta_{t+1,i} = \theta_{t,i} - s_t \eta_i \frac{\widehat{m}_{t,i}}{\sqrt{\widehat{v}_{t,i}} + \epsilon} - s_t \lambda_{t,i} \theta_{t,i}. \tag{11}$$

The groupwise learning rate therefore affects the Adam data update, while weight decay is independent of the peak learning rate and the parameter-group multiplier.

C.3 Muon

We apply Muon (Jordan et al., 2024) to the hidden-layer attention and feed-forward matrices. Muon applies Nesterov momentum followed by Newton–Schulz matrix orthogonalization. Following Qiu et al. (2025), we use Adam for token embeddings, the readout matrix and learned normalization scales. The Adam route has native learning rate multiplier $r_i = 1.6$, while the Muon route has $r_i = 1$.

For an eligible hidden weight matrix, let $G_{t,i}$ denote the effective matrix obtained from its clipped gradient. Rank-three attention kernels are reshaped to matrices before the Muon transformation and restored to their original shapes afterward. Muon updates its momentum buffer and forms the Nesterov lookahead as

$$B_{t,i} = \beta B_{t-1,i} + (1 - \beta) G_{t,i}, \tag{12}$$

$$H_{t,i} = \beta B_{t,i} + (1 - \beta) G_{t,i}, \tag{13}$$

with $B_{0,i} = 0$. Starting from

$$X_0 = \frac{H_{t,i}}{\|H_{t,i}\|_F + \epsilon_{\mathrm{NS}}}, \qquad \epsilon_{\mathrm{NS}} = 10^{-8}, \tag{14}$$

we transpose X_0 when necessary so that its shorter dimension comes first, then apply five Newton–Schulz iterations,

$$A_j = X_j X_j^\top, \tag{15}$$

$$X_{j+1} = c_0 X_j + (c_1 A_j + c_2 A_j^2) X_j, \tag{16}$$

where

$$(c_0, c_1, c_2) = (3.4445, -4.7750, 2.0315). \tag{17}$$

After restoring the original orientation and tensor shape, we apply no additional shape-dependent multiplier, and the Muon direction is

$$d_{t,i} = X_5. \tag{18}$$

Parameters assigned to the Adam route use the Adam direction from the preceding subsection.

C.4 SOAP

We use SOAP (Vyas et al., 2025), which applies Adam in the eigenbasis of a Shampoo preconditioner. Following Wen et al. (2025), we apply SOAP to every learned matrix, including the token embedding, attention and feed-forward kernels, and readout matrix. Rank-three attention kernels are reshaped to their effective matrix representations and restored afterward. Learned normalization scales use the Adam direction defined above. Both routes use $r_i = 1$.

For each matrix, we partition every eligible axis into blocks of at most 512 coordinates, zero-padding and subsequently cropping ragged boundary blocks. An axis longer than 10,000 coordinates is left unpreconditioned. Consequently, the embedding and readout receive one-sided SOAP, with an identity basis on the vocabulary axis. For gradient block $G_{t,b}$, SOAP updates the left and right Gram matrices as

$$L_{t,b} = \beta_{\mathrm{Sh}} L_{t-1,b} + (1 - \beta_{\mathrm{Sh}}) G_{t,b} G_{t,b}^\top, \tag{19}$$

$$R_{t,b} = \beta_{\mathrm{Sh}} R_{t-1,b} + (1 - \beta_{\mathrm{Sh}}) G_{t,b}^\top G_{t,b}. \tag{20}$$

Let $Q_{t,b}^L$ and $Q_{t,b}^R$ denote the bases used for the current update. After an initialization-only optimizer call that constructs these bases by eigendecomposition, index $t = 1, 2, \ldots$ counts applied SOAP updates. SOAP maintains its first moment in the original coordinates and its second moment in the rotated coordinates:

$$M_{t,b} = \beta_1 M_{t-1,b} + (1 - \beta_1) G_{t,b}, \tag{21}$$

$$\overline{G}_{t,b} = (Q_{t,b}^L)^\top G_{t,b} Q_{t,b}^R, \tag{22}$$

$$\overline{M}_{t,b} = (Q_{t,b}^L)^\top M_{t,b} Q_{t,b}^R, \tag{23}$$

$$V_{t,b} = \beta_2 V_{t-1,b} + (1 - \beta_2) \overline{G}_{t,b}^{\odot 2}. \tag{24}$$

The blockwise SOAP direction is

$$\overline{D}_{t,b} = \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \frac{\overline{M}_{t,b}}{\sqrt{V_{t,b}} + \epsilon}, \tag{25}$$

$$D_{t,b} = Q_{t,b}^L \overline{D}_{t,b} (Q_{t,b}^R)^\top. \tag{26}$$

We use $\epsilon = 10^{-15}$ in the denominator of the blockwise SOAP direction in Equation 25 and add no diagonal regularization to the Gram matrices ($\epsilon_{\mathrm{matrix}} = 0$).

We refresh the left and right bases after every $K = 10$ applied updates. Before each refresh, we compute the Rayleigh quotient of every current basis vector. For example, for a left-basis column q_j,

$$\widehat{\lambda}_{t,b,j}^L = q_j^\top L_{t,b} q_j, \tag{27}$$

which estimates the gradient variance along that direction. We sort the basis vectors in decreasing order of these estimates and apply the same permutation to the corresponding rows or columns of

$V_{t,b}$. We then obtain the refreshed basis from a QR decomposition of the Gram matrix applied to the sorted basis,

$$Q^L_{t+1,b} = \mathrm{qr}\Big(L_{t,b} Q^{L,\mathrm{sorted}}_{t,b}\Big), \tag{28}$$

with the analogous operation on the right. Thus the second-moment slots are reordered by approximate spectral rank but are not otherwise transported through the change of basis.

Finally, we merge and crop the blocks, restore the original tensor shape, and set $d_{t,i} = D_{t,i}$. We apply no additional spectral normalization or shape-dependent multiplier.

C.5 ADANA

We implement ADANA in JAX using the update rules from the released PyTorch implementation for the language-model experiments in Ferbach et al. (2025; 2026). Let $t = 1, 2, \ldots$ count applied optimizer updates; gradient-accumulation microbatches do not advance this clock. ADANA uses the same log-time weight for its first and second moments,

$$\Delta_t = \frac{\delta}{\delta + t}, \tag{29}$$

and updates

$$m_t = (1 - \Delta_t) m_{t-1} + \Delta_t \widetilde{g}_t, \tag{30}$$

$$v_t = (1 - \Delta_t) v_{t-1} + \Delta_t \widetilde{g}_t^{\odot 2}, \tag{31}$$

with $m_0 = v_0 = 0$. We apply no additional bias correction.

Define the time-dependent momentum amplification

$$\chi_t = (t+1)^{1-\kappa} + 1. \tag{32}$$

The ADANA direction is

$$d_t = \frac{g_2 \widetilde{g}_t + g_3 \chi_t m_t}{\sqrt{v_t} + \epsilon}, \tag{33}$$

where all operations involving v_t are elementwise. We use $\epsilon = 10^{-15}$. We fix $g_2 = 1$ and $\delta = 8$, while g_3 and κ control the strength and growth rate of the momentum contribution and are treated as momentum hyperparameters. The additional $+1$ in χ_t and the first-update clock follow the released implementation rather than the ADANA pseudocode in Ferbach et al. (2026).

All reported ADANA experiments disable the optional coordinatewise update cap.

C.6 Parameter routing

Table 6 summarizes the optimizer direction and weight decay mask for each parameter class.

Table 6: Parameter routing for the four primary optimizers. The weight decay column gives the common independent weight decay mask.

Parameter class	AdamW	Muon	SOAP	ADANA	Weight decay
Token embedding	Adam	Adam	SOAP	ADANA	Yes
Attention and feed-forward matrices	Adam	Muon	SOAP	ADANA	Yes
Readout matrix	Adam	Adam	SOAP	ADANA	Yes
Learned normalization scales	Adam	Adam	Adam	ADANA	No

All native route multipliers are $r_i = 1$, except for Muon's Adam route, which uses $r_i = 1.6$. Models in the primary architecture contain no bias parameters.

D LEARNING RATE SCHEDULE SELECTION

We compare linear and cosine learning rate decay across training horizons, jointly tuning the base learning rate and weight decay coefficient for each schedule and OT factor.

Both schedules begin with a linear warmup from 0.01η to the peak learning rate η over $W = \max\{100, \min\{\lfloor S/50 \rfloor, 5{,}000\}\}$ optimizer updates, where S is the total number of optimizer updates. For the remaining $S - W$ updates, let $u_t \in [0, 1]$ denote the fraction of post-warmup training completed, and write the learning rate as $\eta_t = \eta s_t$. We compare linear decay to zero with cosine decay to zero (Loshchilov & Hutter, 2017),

$$s_t^{\text{linear}} = 1 - u_t, \qquad s_t^{\text{cosine}} = \frac{1 + \cos(\pi u_t)}{2}. \tag{34}$$

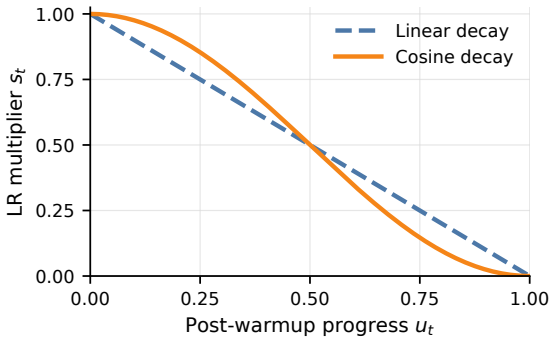

Figure 6: **Linear and cosine learning rate decay schedules after warmup.** Both schedules use the same linear warmup before decaying from the peak learning rate to zero.

We compare linear and cosine decay on the 51M model at $1\times$, $8\times$, and $32\times$ OT for AdamW, ADANA, Muon, and SOAP. We hold the optimizer-specific momentum hyperparameters fixed throughout this comparison: $\beta_1 = 0.9$ and $\beta_2 = 0.95$ for AdamW; $g_3 = 8$ and $\kappa = 0.85$ for ADANA; $\beta = 0.95$ for Muon; and $\beta_1 = 0.95$, $\beta_2 = 0.99$, and $\beta_{\text{Sh}} = 0.95$ for SOAP.

To account for possible interactions between the learning rate schedule and weight decay coefficient, we sweep the base learning rate η and weight decay coefficient c separately for each optimizer, OT factor, and schedule. We use the uniform WD schedule, defined here by a per-update coefficient c/S before multiplication by the learning rate schedule s_t. In particular, s_t multiplies both the optimizer's data-dependent update and the independent weight decay update, making an interaction between the learning rate schedule shape and the weight decay coefficient plausible. For each schedule, we extend the learning rate sweep at every weight decay coefficient, OT factor, and optimizer setting until the minimum is attained in the interior of the sweep. We likewise extend the weight decay range whenever the best pair lies on its boundary. We then report the lowest final validation loss over the resulting (learning rate, weight decay) grid for each optimizer and OT factor.

Let L_s^* denote the lowest final validation loss over the resulting (learning rate, weight decay) grid for schedule $s \in \{\text{linear}, \text{cosine}\}$. We summarize the schedule advantage as

$$10^3 \left(L_{\text{linear}}^* - L_{\text{cosine}}^*\right)$$

in mnats, so positive values indicate that cosine decay achieves the lower final validation loss. Appendix E separately selects the weight decay schedule and its WD coefficient scaling rule. The exact fixed memory settings, tested WD coefficients, and complete learning rate sweeps appear in Appendix K.

Figure 8 summarizes the horizon-dependent crossover. After selecting the best learning rate and weight decay coefficient pair for each schedule, linear decay achieves the lower loss at $1\times$ OT for all four optimizers, whereas cosine decay achieves the lower loss at $8\times$ and $32\times$ OT for all four. Thus, a schedule selected at $1\times$ OT does not transfer unchanged to the more overtrained regimes studied here.

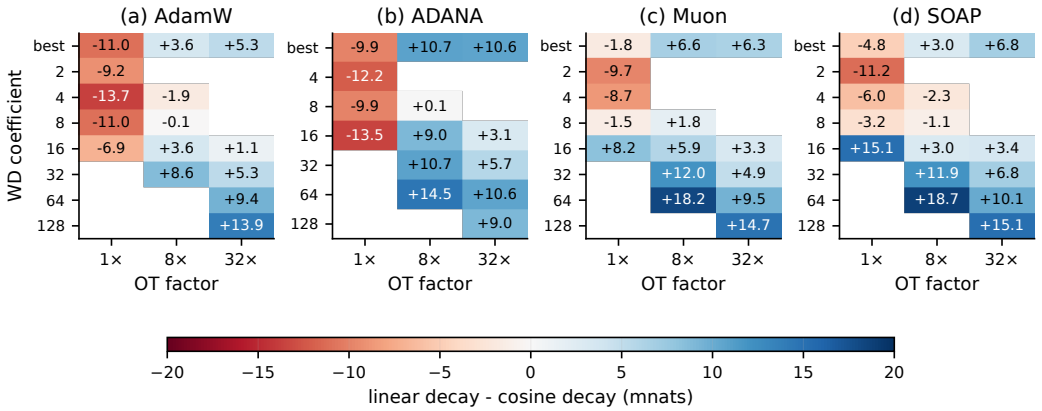

Figure 7: **Cosine decay vs linear decay learning rate schedule advantage across OT factor and coefficient of the uniform WD schedule on the 51M parameter model.** Each cell reports $10^3(L_{\text{linear}} - L_{\text{cosine}})$ after selecting the best base learning rate separately for each schedule at the indicated weight decay coefficient; positive values favor cosine decay to zero. The row labeled "best" selects the best weight decay coefficient for each OT factor separately for each schedule. Momentum hyperparameters are fixed to AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$), ADANA ($g_3 = 8, \kappa = 0.85$), Muon ($\beta = 0.95$), and SOAP ($\beta_1 = 0.95, \beta_2 = 0.99, \beta_{\text{Sh}} = 0.95$).

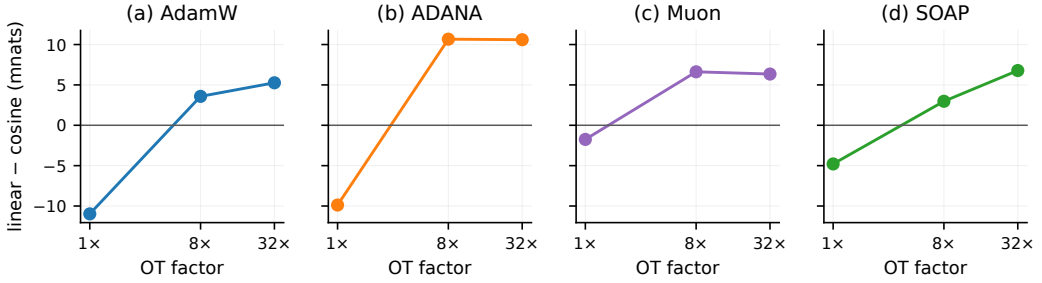

Figure 8: **High overtraining factors favor cosine decay over linear decay on the 51M parameter model for all optimizers.** Each point reports $10^3(L_{\text{linear}} - L_{\text{cosine}})$ after selecting the best base learning rate and weight decay coefficient pair for each schedule; positive values favor cosine decay to zero. Momentum hyperparameters are fixed as in Figure 7.

Figure 7 shows that the crossover is not produced by a single favorable weight decay coefficient: the high-OT advantage of cosine decay appears across multiple coefficients. The same crossover remains in the row labeled "best," where the weight decay coefficient is selected separately for each schedule and OT factor.

Because the remaining experiments in this paper focus primarily on high OT factors, we use cosine decay to zero after warmup as the primary learning rate schedule.

This comparison is restricted to the 51M model, the two schedule families considered here, fixed optimizer-specific momentum settings, and the uniform WD schedule. Appendix E next studies weight decay schedules and coefficient scaling, while Appendix K reports the complete learning rate sweeps underlying this comparison.

E WEIGHT DECAY SCHEDULES AND COEFFICIENT SCALING

The weight decay (WD) schedule specifies how the coefficient varies within a run, while the coefficient scaling rule specifies how its amplitude changes across training horizons. We compare uniform and log-time WD under cosine learning rate decay to zero, then fit coefficient rules for settings where per-horizon tuning is impractical. Appendix K reports the underlying learning rate-by-coefficient grids.

Following decoupled weight decay (Loshchilov & Hutter, 2019), we implement weight decay as a separate parameter update term rather than as an L_2 penalty in the training objective. We apply weight decay to all learned parameters except learned normalization scales, including both the token embedding and readout matrices. Our models do not contain biases. For parameters included in the weight decay mask, the independent decay contribution at update t is $-s_t\lambda_t\theta_t$, where $s_t \in [0, 1]$ denotes the normalized learning rate schedule.

We use uniform WD to denote the standard schedule in which the per-update coefficient, before multiplication by s_t, is constant across optimizer updates. Following Wang & Aitchison (2025), we divide the WD coefficient by the total number of optimizer updates S:

$$\lambda_t^{\text{uniform}} = \frac{c_{\text{uniform}}}{S}. \tag{35}$$

For a fixed learning rate schedule shape, this approximately prevents cumulative decay from increasing with the number of updates.

We use log-time WD to denote the schedule proposed in Ferbach et al. (2026), in which the per-update coefficient decreases inversely with the shifted update index:

$$\lambda_t^{\log} = \frac{c_{\log}}{\tau + t}. \tag{36}$$

We use c for the WD coefficient throughout and add a schedule subscript only when multiple WD schedules appear in the same expression. The coefficient values are not directly comparable across schedules because they parameterize different decay shapes.

E.1 PROTOCOL CHOICE FOR THE LOG-TIME WD OFFSET

Applying log-time WD across training horizons requires specifying how its offset changes when the number of optimizer updates changes. Let S denote the total number of optimizer updates in the current experiment and $S_{1\times}$ the number at $1\times$ OT. The schedule in Ferbach et al. (2026) uses an offset proportional to the current training horizon. We instead anchor the offset to the $1\times$ OT horizon:

$$\underbrace{\lambda_{t,\text{horizon}}^{\log} = \frac{c}{0.1S + t}}_{\text{horizon-normalized}} \qquad \underbrace{\lambda_{t,1\times}^{\log} = \frac{c}{0.1S_{1\times} + t}}_{1\times\text{-anchored, used here}}. \tag{37}$$

The two definitions coincide at $1\times$ OT. At longer horizons, the horizon-normalized definition stretches the initial phase of the WD schedule with the full training horizon, whereas the $1\times$-anchored definition preserves the initial timescale used at $1\times$ OT. We adopt $\tau = 0.1S_{1\times}$ as our protocol convention for extending log-time WD across the OT axis. Unless otherwise stated, all log-time WD experiments in this paper use this convention.

E.2 INTERACTION WITH THE LEARNING RATE ENDPOINT

The language model experiments in Ferbach et al. (2026) use cosine decay to 10% of the peak learning rate. Comparing log-time WD with uniform WD for AdamW and AdEMAMix, they report approximately 10% compute-efficiency gains that appear to increase with model scale. The literature generally finds that decaying the learning rate to zero outperforms schedules with higher endpoints. For example, Bergsma et al. (2025b) find that linear decay to zero consistently outperforms cosine decay to 10% across a range of language model pretraining settings. We therefore test whether the magnitude of the log-time WD advantage depends on the learning rate endpoint.

To isolate this interaction, we hold the base learning rate fixed at $\log_2 \eta = -8$ for AdamW and -12.5 for ADANA and match the total parameter shrinkage produced by the two WD schedules

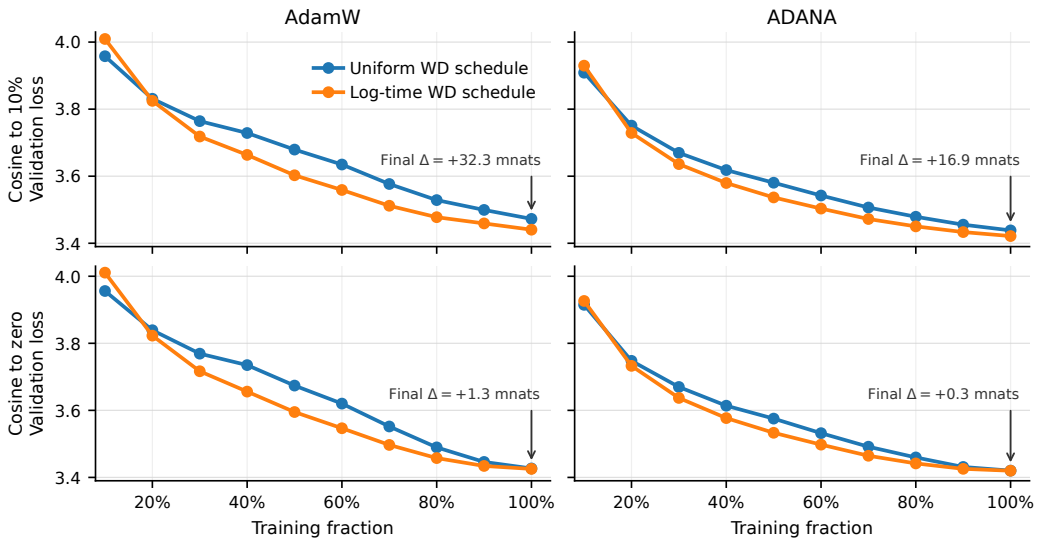

Figure 9: **The apparent advantage of the log-time WD schedule is substantially diminished when the learning rate decays to zero rather than to 10% of its peak.** We compare the uniform and log-time WD schedules with the same base learning rate and matched total parameter shrinkage from weight decay alone on the 51M parameter model at $8\times$ OT. Markers show validation evaluations at selected training steps, and lines linearly interpolate between them. With a 10% endpoint, the log-time WD schedule reduces final validation loss by 32.3 mnats for AdamW and 16.9 mnats for ADANA; with a zero endpoint, the corresponding differences are 1.3 and 0.3 mnats.

in the absence of gradient updates. Within each optimizer and learning rate endpoint pair, all other training coordinates are identical. Under weight decay alone, a parameter evolves according to

$$\theta_{t+1} = (1 - s_t\lambda_t)\theta_t. \tag{38}$$

The fraction of its initial value retained after S updates is therefore

$$R(\lambda) = \prod_{t=0}^{S-1}(1 - s_t\lambda_t), \qquad \theta_S = R(\lambda)\theta_0. \tag{39}$$

For a log-time WD coefficient $c_{\log} = 2$, we choose the uniform WD coefficient c_{uniform} by solving

$$\prod_{t=0}^{S-1}\left(1 - s_t\frac{c_{\text{uniform}}}{S}\right) = \prod_{t=0}^{S-1}\left(1 - s_t\frac{c_{\log}}{\tau + t}\right). \tag{40}$$

We solve Equation 40 numerically in log space using the exact discrete learning rate schedule, including warmup. This gives $c_{\text{uniform}} = 11.500447$ for cosine decay to 10% and $c_{\text{uniform}} = 12.319252$ for cosine decay to zero. The matching depends only on the learning rate schedule and training horizon, so we use the same coefficients for AdamW and ADANA. This comparison therefore isolates the effect of when weight decay is applied while holding its cumulative parameter shrinkage fixed. These coefficients match retention rather than minimize validation loss.

Under both learning rate endpoints, log-time WD develops an advantage during training; with decay to zero, however, most of this advantage disappears during the final learning rate cooldown. The observed log-time-WD advantage therefore depends strongly on the learning rate endpoint in this comparison with the protocol of Ferbach et al. (2026).

E.3 Comparing uniform and log-time WD schedules across OT factors

Because our primary learning rate schedule decays to zero, we next compare uniform and log-time WD under this endpoint. For each optimizer, WD schedule, and OT factor, we evaluate a two-dimensional grid over base learning rate and WD coefficient. At each WD coefficient, we sweep the

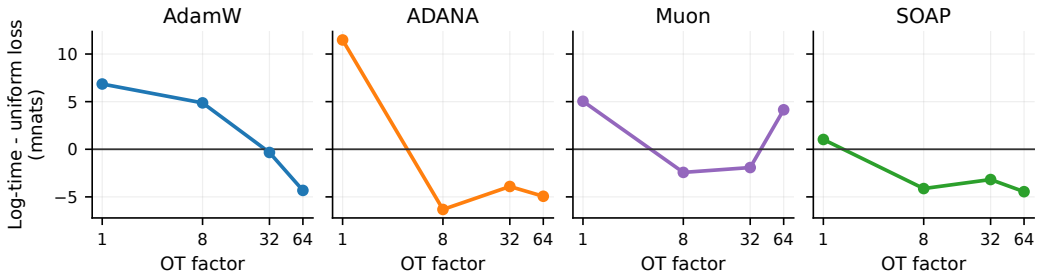

Figure 10: **The relative performance of the uniform and log-time WD schedules depends on both the optimizer and OT factor.** On the 51M parameter model with cosine decay to zero and fixed optimizer-specific momentum, each point shows $10^3(L_{\log} - L_{\text{uniform}})$ after selecting the best (base learning rate, WD coefficient) pair for each WD schedule. Positive values favor the uniform WD schedule, while negative values favor the log-time WD schedule.

base learning rate independently and then compare the resulting minimum losses across coefficients, following the sweep-interiority criteria in Section B.5. The summary figures report quantities minimized over base learning rate, while Section K reports the coefficient grids in Table 18 and shows every evaluated point in Figures 46 and 47. This comparison therefore makes no assumption about how the WD coefficient should scale across training horizons.

Uniform WD performs better at $1\times$ OT for AdamW, ADANA, and Muon. At higher OT factors, log-time WD becomes competitive or better for some optimizers, including ADANA and SOAP at the longest measured horizons, but it is not uniformly superior across optimizers or horizons.

E.4 WD COEFFICIENT SCALING RULES ACROSS OT FACTORS

When computationally feasible, the best way to set weight decay is to jointly tune the base learning rate and WD coefficient at every training horizon. However, this joint sweep is often computationally impractical. We therefore investigate how the WD coefficient should scale with training horizon to determine a scaling rule that can be used in place of per-horizon WD tuning.

An equivalent view of AdamW treats the parameter iterate itself as an exponential moving average of preconditioned updates: in conventional notation with constant learning rate η and weight decay λ, the averaging coefficient is $\eta\lambda$ and the resulting memory timescale is $1/(\eta\lambda)$ updates (Wang & Aitchison, 2025). This view has recently been used to explain other horizon-dependent phenomena. Bergsma et al. (2026b) show that the normalized AdamW timescale controls training-loss-curve shape and must be matched, together with tokens per parameter, for curves to collapse across model scales. Bergsma et al. (2026a) predict when training data are retained from AdamW's implicit EMA coefficients and use the resulting re-evaluation curves to design data curricula. We therefore interpret WD coefficient scaling as controlling a normalized optimizer memory, rather than only the strength of regularization.

Prior work in Bergsma et al. (2025a) asks the same question for AdamW under a uniform WD schedule: how should the WD coefficient scale with model size, training horizon, and batch size? In the following imported notation, B counts tokens per update, D is the total training-token budget, and N is model size; these differ from the architecture symbols in Table 1. The normalized timescale τ below is also distinct from our log-time-WD offset. They define the normalized AdamW timescale

$$\tau = \frac{B}{\eta\lambda D} = \frac{1}{\eta\lambda S}, \tag{41}$$

and find that the optimal timescale scales approximately as

$$\tau_{\text{opt}} \propto \left(\frac{D}{N}\right)^{-0.527}. \tag{42}$$

They also find that, at fixed model size and training horizon, the optimal raw AdamW coefficient λ scales approximately linearly with batch size below the critical batch size. In our uniform WD

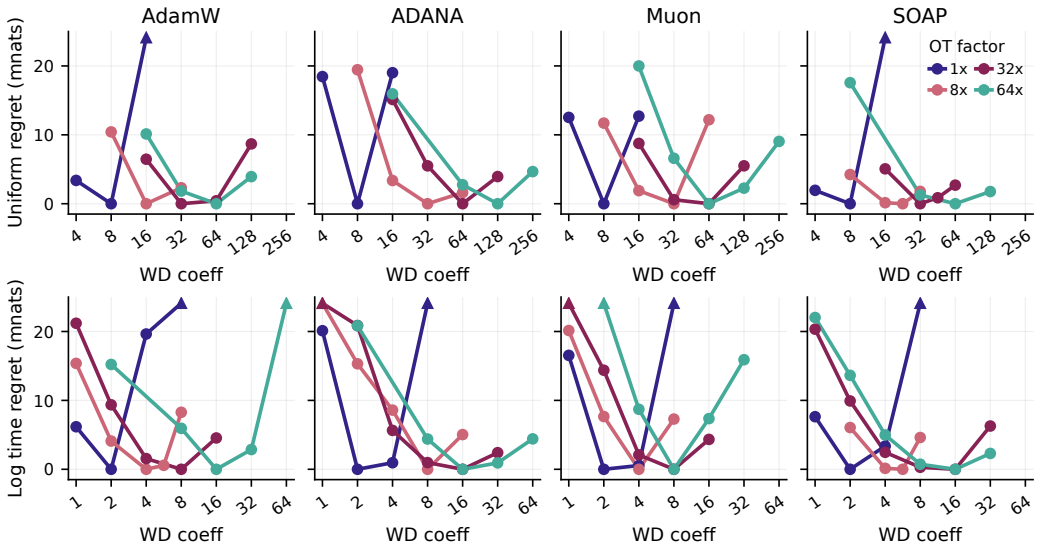

Figure 11: **The best WD coefficient changes with OT factor.** Each curve shows coefficient regret at one OT factor for a fixed optimizer and WD schedule on the 51M parameter model. The regret of coefficient c is $10^3[L(c) - \min_{c'} L(c')]$, where $L(c)$ is the best final validation loss over the base learning rate sweep at coefficient c. The movement of the minima across curves shows that a coefficient selected at one training horizon need not transfer to another. The learning rate schedule uses cosine decay to zero and optimizer-specific momentum is held fixed. Upward triangles at the upper boundary represent regrets greater than 25 mnats.

parameterization, c/S corresponds to the peak effective decay coefficient $\eta\lambda$, and hence $c = 1/\tau$. Thus, at fixed token horizon, their batch-scaling prescription corresponds to holding c fixed as batch size varies: the factor of B is already absorbed through $S = D/B$. Their horizon-scaling result then implies

$$\frac{c_{\text{uniform}}(f)}{c_{\text{uniform}}(1)} = f^{0.527} \approx \sqrt{f}. \tag{43}$$

Power Lines studies uniform WD with AdamW; we extend this analysis to log-time WD and the broader set of optimizers considered here.

Using the per-horizon joint sweeps described above, Figure 11 shows the resulting regret for each WD coefficient after minimizing over the base learning rate,

$$\mathcal{R}(c; f) = 10^3 \left[L_f(c) - \min_{c'} L_f(c') \right], \tag{44}$$

where $L_f(c)$ is the best final validation loss over the learning rate sweep at coefficient c.

The best WD coefficient generally increases with OT factor for both WD schedules, although its precise value and the width of the low-regret region differ across optimizers. Consequently, fixing the coefficient selected at $1\times$ OT can incur increasing regret at longer training horizons.

To quantify this trend, let $c^*_{o,s}(f)$ denote the best coefficient for optimizer o, WD schedule s, and OT factor f after minimizing over the base learning rate. We anchor the uniform and log-time WD schedules at their $1\times$ OT coefficients, $c_{\text{uniform}}(1) = 8$ and $c_{\log}(1) = 2$, and fit a separate power-law exponent for each WD schedule across optimizers by least squares in log space:

$$\log_2 c^*_{o,s}(f) = \log_2 c_s(1) + \alpha_s \log_2 f + \epsilon_{o,s,f}. \tag{45}$$

The independently fitted exponents are 0.527 for uniform WD and 0.473 for log-time WD.

Because both fitted exponents are close to $1/2$, we use the square-root exponent for both schedules, with schedule-specific $1\times$ OT coefficients determined by our sweeps:

$$c_{\text{uniform}}(f) = 8\sqrt{f}, \qquad c_{\log}(f) = 2\sqrt{f}. \tag{46}$$

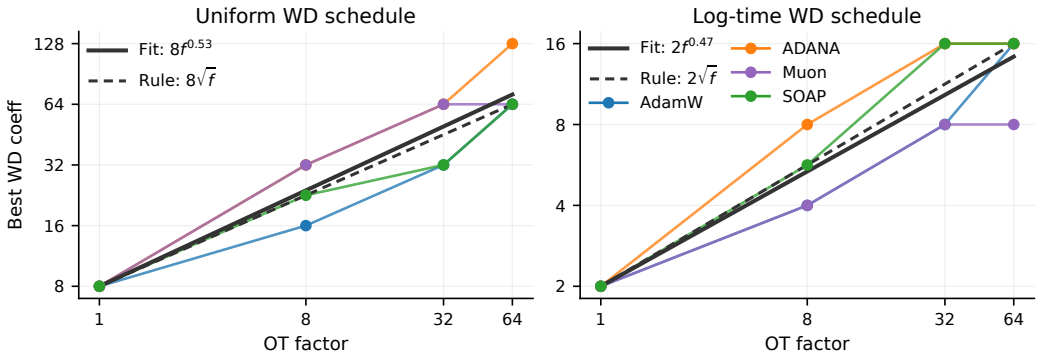

Figure 12: **The best WD coefficients are consistent with square-root scaling across OT factor.** For each optimizer, WD schedule, and OT factor on the 51M parameter model, points show the WD coefficient from the best (base learning rate, WD coefficient) pair in the two-dimensional grid. The solid black curves show the separate power-law fits in Equation 45, while the dashed curves show the adopted square-root rules. The learning rate schedule uses cosine decay to zero and optimizer-specific momentum is held fixed.

We use these prescriptions when repeating the complete learning rate and WD coefficient grid is computationally impractical. When direct joint tuning is feasible, we instead use the best measured (base learning rate, WD coefficient) pair.

Figures 13–15 compare uniform and log-time weight decay under these coefficient rules, with the base learning rate tuned independently at each setting. Figure 16 compares both schedules against a common uniform-WD AdamW baseline.

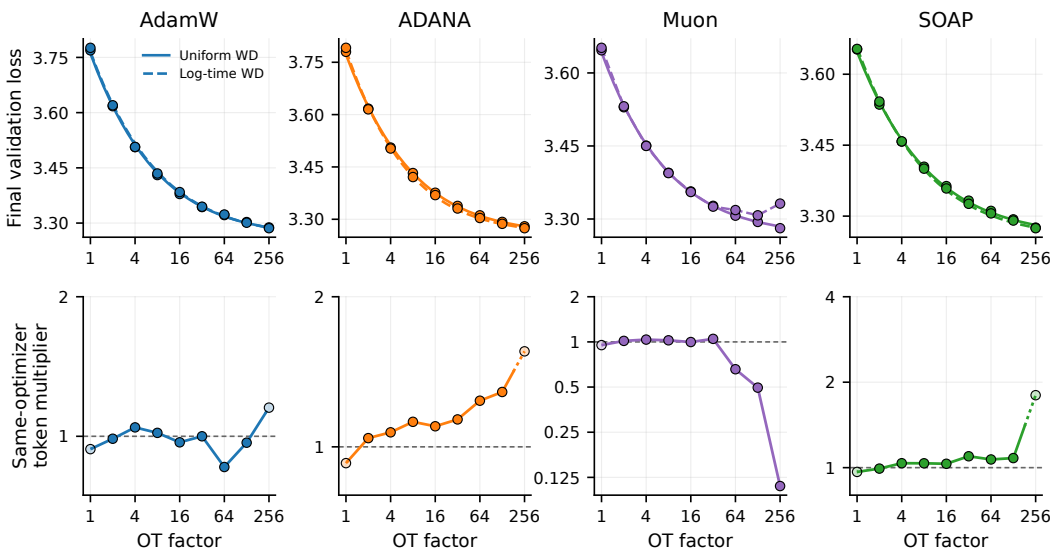

Figure 13: **The effect of log-time weight decay depends on optimizer and training horizon.** We compare the adopted WD coefficient rules on the 51M model, independently tuning the base learning rate; ADANA has no cooldown. Top: final validation losses for uniform (solid) and log-time (dashed) WD. Curves show fitted losses, except for log-time-WD Muon, where they connect observations. Bottom: log-time-WD token multipliers relative to the same optimizer with uniform WD, obtained by inverting observed losses through fitted baseline curves. Values above one favor log-time WD. Lines connect adjacent comparisons; translucent markers and dotted segments indicate extrapolation.

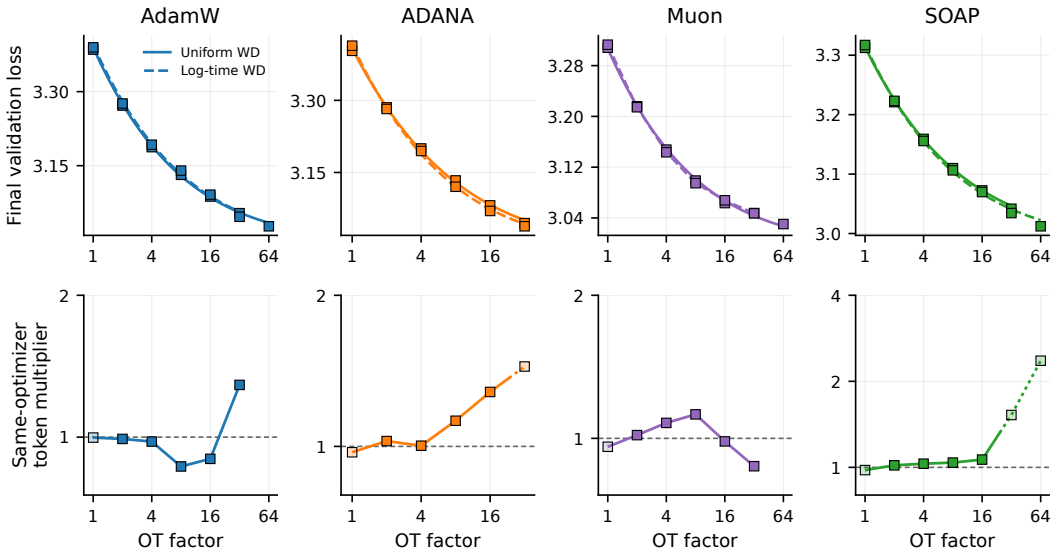

Figure 14: Same as Figure 13, for the 124M model.

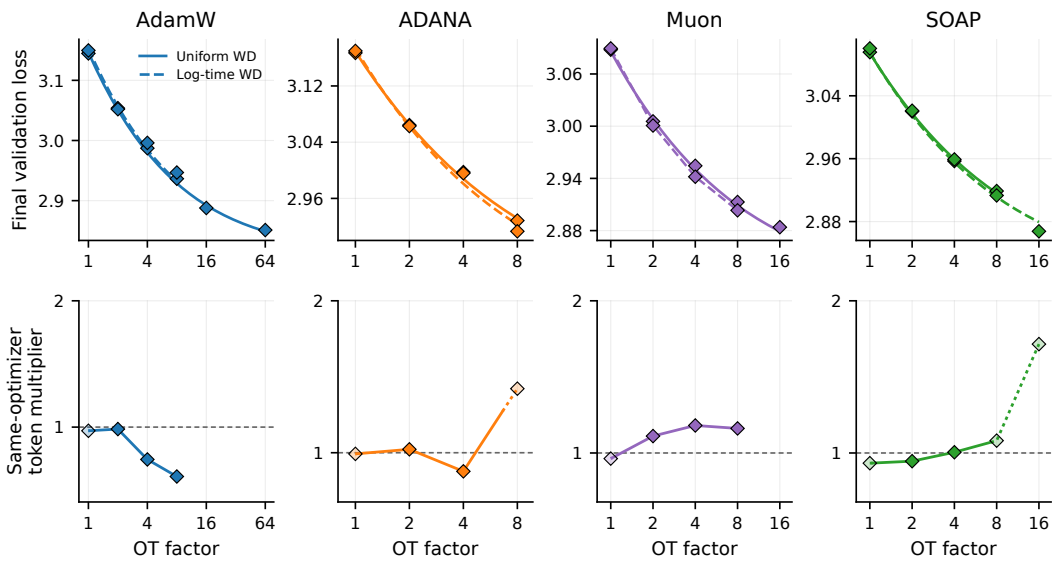

Figure 15: Same as Figure 13, for the 253M model.

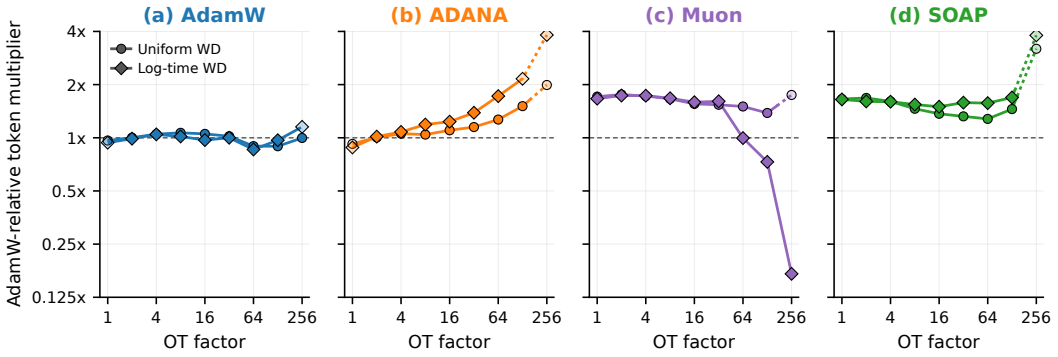

Figure 16: **Both weight decay schedules relative to a common AdamW baseline.** Using the 51M treatments in Figure 13, we compare uniform WD (circles) and log-time WD (diamonds) against uniform-WD AdamW. Markers invert observed losses through the fitted AdamW curve; lines connect adjacent comparisons. Values above one favor the indicated treatment. Translucent markers and dotted segments indicate extrapolation.

F FIXED-MEMORY TUNING ACROSS TRAINING HORIZONS

Spectral analyses of deep linear and kernel learning show that higher-eigenvalue modes are learned earlier, whereas lower-eigenvalue modes require longer training (Saxe et al., 2014; Bordelon et al., 2020). This progression suggests that averaging gradients over longer windows may become increasingly useful at extended training horizons: longer windows can accumulate weak but persistent directional signal needed to resolve harder modes. This motivation is consistent with the long-memory benefits observed for AdEMAMix, the growing-memory construction of DANA, and the use of AdEMAMix in the long-duration Apertus pretraining runs (Pagliardini et al., 2024; Ferbach et al., 2025; Project Apertus et al., 2025). However, longer memory also responds more slowly to changes in the gradient distribution and can retain stale or unusually large gradients for longer, creating a competing stability consideration. For fixed-memory optimizers—which use scalar coefficients such as β to maintain a constant effective memory throughout a run—we therefore ask how the memory coefficient should vary across training horizons.

F.1 PER-HORIZON MEMORY TUNING

We conduct this analysis on the 51M-parameter model, where it is computationally feasible to jointly sweep one optimizer-specific memory coefficient and the base learning rate at every overtraining factor. For AdamW, we vary the second-moment coefficient β_2 and fix $\beta_1 = 0.9$. For Muon, we vary its native momentum coefficient β and fix its non-matrix AdamW path at $(\beta_1, \beta_2) = (0.9, 0.95)$. For SOAP, we vary the rotated-basis second-moment coefficient β_2 and fix $\beta_1 = \beta_{\mathrm{Sh}} = 0.95$ with preconditioner refresh interval $K = 10$. Our choice to tune AdamW's β_2 is supported by Marek et al. (2025), who find that preserving the second-moment half-life transfers better across batch sizes while $\beta_1 = 0.9$ remains effective.

We report each averaging coefficient using the effective memory window M defined in Section 3.1. For constant β, the half-life is $H_{1/2} = \log(1/2)/\log\beta$, with $H_{1/2} \approx (\log 2)M$ near $\beta = 1$. The $M = 50$ setting used in our uniform-weight-decay comparison corresponds to $\beta = 0.98$ and a half-life of approximately 34 optimizer updates. At fixed batch size, holding β constant as the overtraining factor increases makes this window a progressively smaller fraction of training.

For every optimizer, overtraining factor, and value of M, we independently sweep the base learning rate. For all experiments, we sweep over the memory grid $M \in \{20, 40, 80, 160, 320, 640, 1280\}$. We additionally test $M = 10$ at six short-horizon settings whose minima initially reached the lower boundary, and extend AdamW and SOAP to $M = 2560$ at $128\times$ OT. We evaluate these sweeps using the interiority criteria in Appendix B.5. Within each $(\text{optimizer}, M, \text{OT factor})$ setting, we first require the optimal learning rate to be interior in the learning rate sweep. After minimizing over learning rate, we require the optimal memory to be interior in the memory sweep at every $(\text{optimizer}, \text{OT factor})$ setting. At each horizon, the per-horizon treatment selects the $(M, \text{base learning rate})$ pair with the lowest final validation loss.

F.2 HORIZON-TUNED MEMORY RESULTS

Longer horizons favor longer memory. Figure 2a shows that longer training horizons generally favor longer fixed memory. From $1\times$ to $128\times$ OT, the optimal memory increases from $M = 20$ to $M = 640$ for Muon, from $M = 20$ to $M = 1280$ for AdamW, and from $M = 40$ to $M = 1280$ for SOAP, although the trajectories are not always monotonic.

Many horizons have broad low-regret regions, so small loss differences can change the minimizing memory setting (Figure 20). We use the best measured setting at each horizon for the fixed-memory comparison.

Memory and learning rate must be tuned jointly. Figure 21 shows that the loss-minimizing base learning rate generally decreases as effective memory grows, although the relationship depends on optimizer and horizon. This coupling creates a tradeoff: longer memory becomes useful at long horizons but is typically paired with a smaller learning rate, which can slow learning throughout the run. A fixed-memory optimizer must accept one compromise between memory and learning rate throughout training, whereas scheduled memory can use shorter memory early and lengthen it

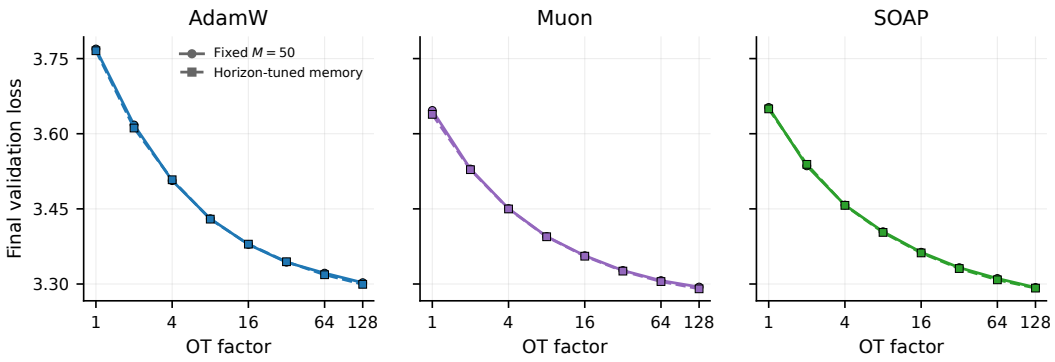

Figure 17: **Horizon-tuned memory improves on a single fixed-memory setting.** Final validation losses for the 51M model under uniform weight decay, comparing fixed $M = 50$ with memory selected separately at each horizon. We independently tune the base learning rate for every setting.

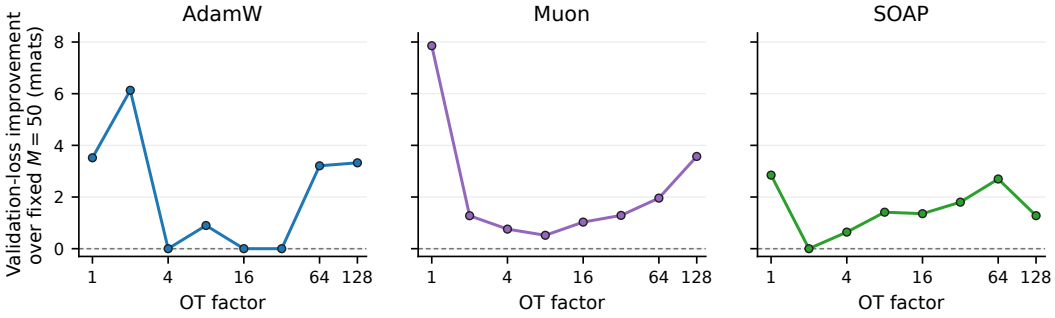

Figure 18: **Per-horizon memory tuning improves performance at both short and long horizons.** Validation-loss reductions relative to each optimizer's $M = 50$ setting in Figure 17. Positive values favor horizon-tuned memory.

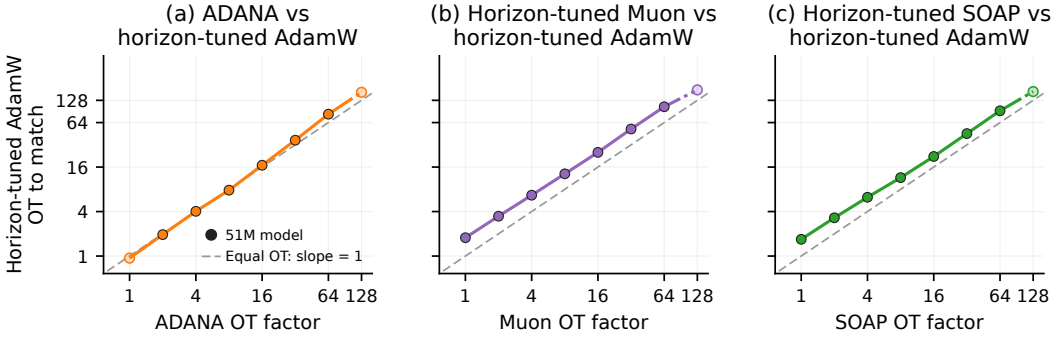

Figure 19: **ADANA's advantage over AdamW persists after horizon-specific fixed-memory tuning.** Comparisons use the 51M model with uniform weight decay. AdamW, Muon, and SOAP jointly tune memory and learning rate at each horizon; ADANA uses $(\kappa, \delta, g_3) = (0.85, 8, 8)$, without cooldown, and tunes only the learning rate. We compute equivalent OT by inverting a piecewise-linear curve through horizon-tuned AdamW losses in log-OT. Translucent markers indicate extrapolation.

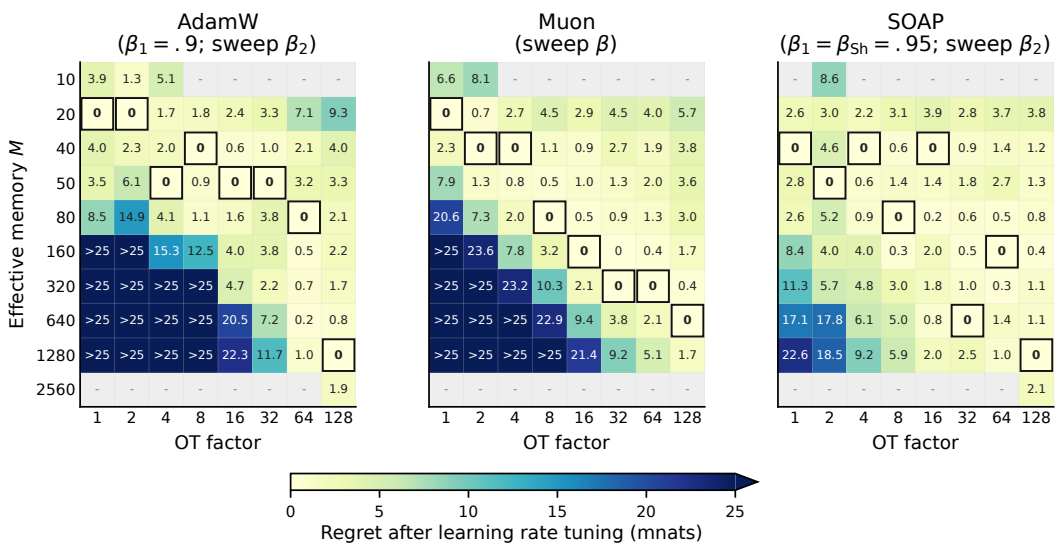

Figure 20: **Longer horizons generally favor longer memory, with broad low-regret ranges.** Each cell shows the validation-loss increase over the best memory at that horizon after independent learning rate tuning, for the 51M model under uniform weight decay. Black outlines mark the minimizing memory. Colors saturate at 25 mnats.

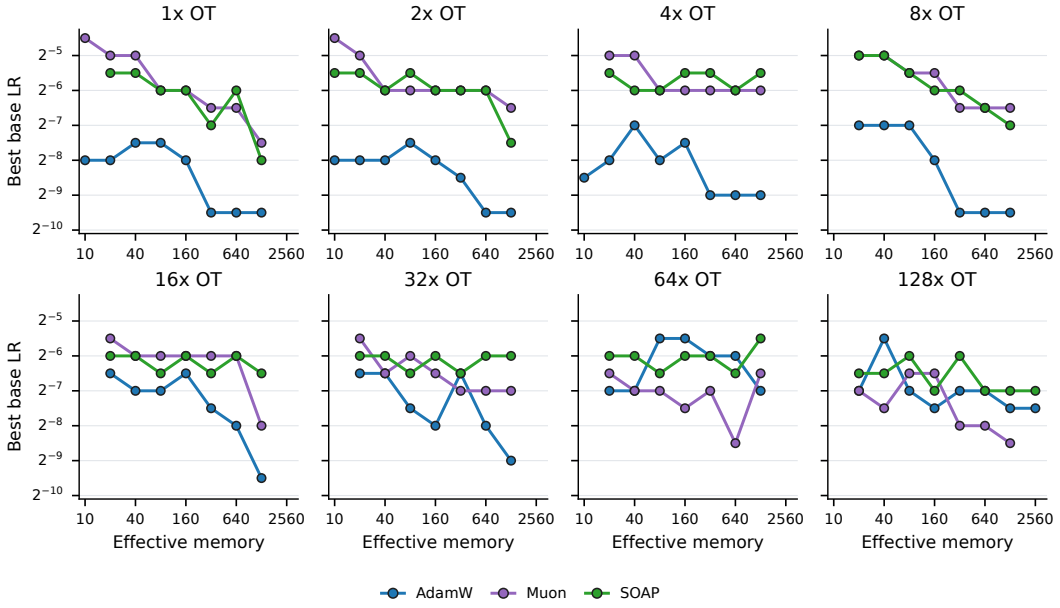

Figure 21: **The best learning rate generally decreases as memory grows, with optimizer- and horizon-dependent deviations.** Each panel fixes the OT factor and shows the base learning rate attaining the lowest final validation loss at each memory setting, for the 51M model under uniform weight decay.

as training proceeds. This provides one plausible explanation for why per-horizon tuned constant memory does not fully match scheduled memory.

Per-horizon tuning improves fixed-memory optimizers. The fixed-memory comparison uses $\beta_2 = 0.98$ for AdamW and SOAP and $\beta = 0.98$ for Muon, corresponding to $M = 50$. The per-horizon sweeps show that $M = 50$ is near-optimal at moderate horizons, but tends to provide too

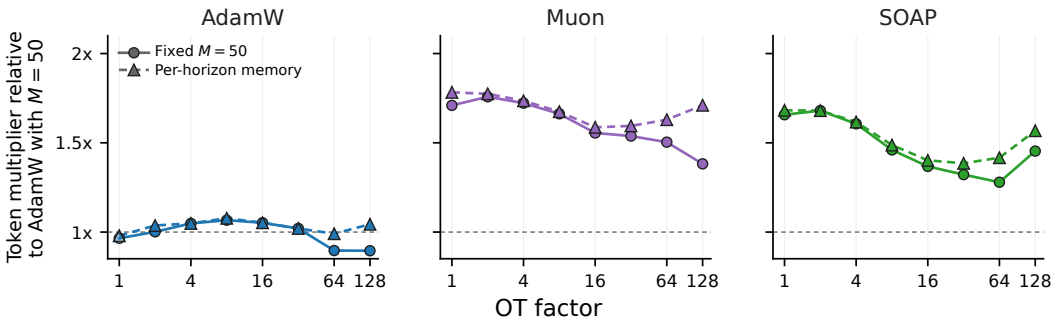

Figure 22: **Horizon-specific memory tuning improves token efficiency relative to fixed-memory AdamW.** The losses in Figure 17 are converted to token multipliers by inverting the fitted $M = 50$ AdamW baseline. Circles show fixed $M = 50$; triangles show horizon-tuned memory. Values above one favor the indicated optimizer.

much memory at the shortest horizons and too little at the longest. The gains therefore appear at both ends of our overtraining range. At $128\times$ OT, per-horizon tuning improves validation loss by 3.3 mnats for AdamW, 3.6 mnats for Muon, and 1.3 mnats for SOAP. Figure 22 expresses the resulting performance as token multipliers relative to fixed-$M = 50$ AdamW.

Interpretation and limitations. At each horizon, this comparison selects the best measured combination of fixed memory and learning rate on the 51M model, varying one memory coefficient per optimizer.

Complete learning rate sweeps for this section are included in Figures 50 and 51 of Appendix K.

G MOMENTUM COOLDOWN FOR LOG-TIME MEMORY

G.1 MOTIVATION: LONG MEMORY CAN BECOME STALE DURING TERMINAL DECAY

Long memory improves gradient averaging, but reduces responsiveness to changes in the optimization trajectory. Under ADANA's log-time rule, the momentum timescale grows approximately linearly with the number of optimizer updates. While the training dynamics change slowly, the optimizer can average over an increasingly long history without fixing one memory length for the entire run.

The same persistence may become undesirable during terminal learning rate decay. Although the learning rate is not stored in the momentum state, the buffer retains gradients evaluated at earlier parameter values reached using larger step sizes. When the learning rate changes faster than the momentum state can respond, substantial kernel mass remains on directions from an earlier optimization regime. We refer to this mismatch as stale memory: the stored gradients are not intrinsically invalid, but are old relative to the timescale on which the current update rule is changing.

This suggests a schedule-aware compromise: retain ADANA's growing memory while the learning rate changes slowly, but shorten it once terminal decay becomes faster than the momentum response. Momentum cooldown implements this principle directly.

G.2 DEFINITION OF MOMENTUM COOLDOWN

Let

$$\tau_{\mathrm{mom},t} = \frac{1}{\Delta_t} = \frac{\delta + t}{\delta} \tag{47}$$

denote the momentum timescale of ADANA's original log-time update. We define the local learning rate decay timescale as

$$\tau_{\eta,t} = \frac{\eta_t}{\eta_t - \eta_{t+1}} \tag{48}$$

whenever $\eta_{t+1} < \eta_t$, and set $\tau_{\eta,t} = \infty$ during constant or increasing portions of the schedule. Momentum cooldown replaces the original timescale with

$$\tau_{\mathrm{CD},t} = \max(1, \min\{\tau_{\mathrm{mom},t}, \tau_{\eta,t}\}), \qquad \Delta_t^{\mathrm{CD}} = \frac{1}{\tau_{\mathrm{CD},t}}. \tag{49}$$

The cooled memory state therefore follows

$$m_t^{\mathrm{CD}} = \left(1 - \Delta_t^{\mathrm{CD}}\right) m_{t-1}^{\mathrm{CD}} + \Delta_t^{\mathrm{CD}} \widetilde{g}_t. \tag{50}$$

Before the two timescales cross, $\tau_{\eta,t} \geq \tau_{\mathrm{mom},t}$, so cooldown leaves the original log-time memory rule unchanged. During sufficiently rapid learning rate decay, it increases the weight placed on the current gradient and concentrates the momentum kernel on more recent updates. The construction depends only on relative changes in the learning rate and is therefore invariant to its overall scalar scale.

Because the standard ADANA implementation uses the same log-time coefficient for its first- and second-moment states, our principal ADANA treatment applies Δ_t^{CD} to both states. We separately examine which states should be cooled in the treatment-selection experiments below. Figure 23 shows how this rule changes the momentum timescale and the age distribution of the resulting kernel.

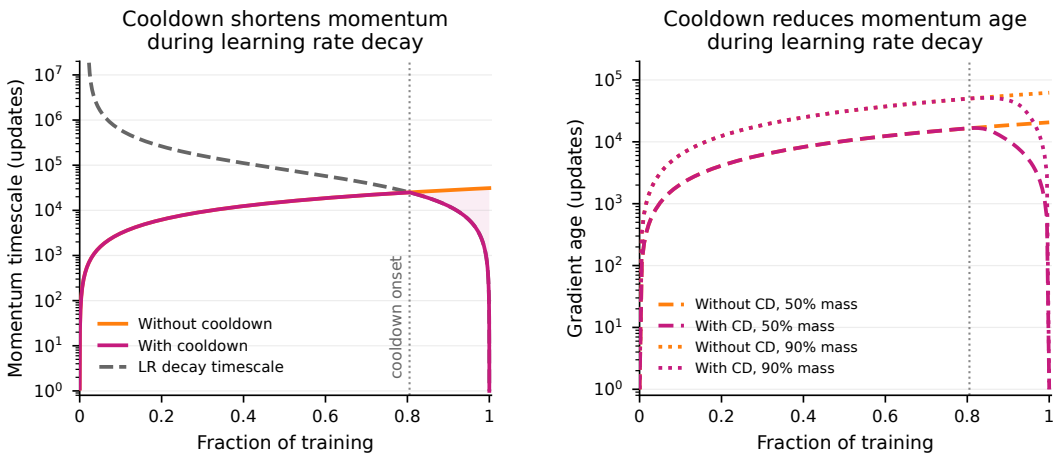

Figure 23: **Momentum cooldown shortens memory during terminal learning rate decay.** We illustrate the rule for the training schedule of a 51M ADANA model at 128× OT. Left: the cooled momentum timescale is the smaller of the original momentum and learning rate decay timescales, bounded below by one update. Right: the shortest recent histories containing 50% and 90% of the momentum kernel's mass. Cooldown leaves the kernel unchanged before activation and concentrates it on more recent gradients afterward.

G.3 Momentum Cooldown Treatment Selection

We distinguish two interventions on ADANA's long-memory channel. *Memory-window cooldown* shortens the kernel used to update the long-memory state. *Long-memory contribution cooldown* leaves that kernel unchanged and instead multiplies the long-memory contribution $g_3\chi_t m_t$ by $\min(1, \tau_\eta/\tau_{\text{mom}})$ during learning rate decay. The combined treatment applies both changes. For AdamW, we analogously apply the memory-window rule to the first moment, the second moment, or both moments.

Table 7: **Momentum cooldown treatment selection at 8× OT on the 51M model with uniform weight decay.** Each reported minimum comes from an independent base learning rate sweep, and gains are measured relative to the same optimizer without cooldown.

Optimizer	Cooldown target	Best base LR	Validation loss	Gain vs. standard (mnats)
AdamW	None	$2^{-7.0}$	3.430405	0.00
AdamW	First moment	$2^{-7.0}$	3.428106	+2.30
AdamW	Second moment	–	**Unstable**	–
AdamW	First and second moments	$2^{-7.0}$	3.428112	+2.29
ADANA	None	$2^{-12.5}$	3.432476	0.00
ADANA	Memory window	$2^{-12.5}$	3.421004	+11.47
ADANA	Long-memory contribution	$2^{-12.5}$	3.431558	+0.92
ADANA	Window and contribution	$2^{-12.5}$	3.426349	+6.13

Cooling AdamW's first moment improves validation loss by 2.30 mnats at 8× OT. Cooling both moments gives essentially the same improvement. By contrast, cooling only the second moment is unstable at every tested learning rate, producing validation losses from 15.2 to 42.9. A response-time mismatch may explain this behavior: the second-moment denominator becomes rapidly responsive during learning rate decay while the first-moment numerator continues to retain older gradients. Cooling both moments removes this mismatch and restores stable training, but provides no additional gain beyond cooling the first moment alone.

For ADANA, memory-window cooldown improves validation loss by 11.47 mnats. Tapering only the long-memory contribution improves loss by 0.92 mnats, while combining that taper with memory-window cooldown improves loss by 6.13 mnats. All three selected minima come from learning rate sweeps satisfying the interiority criterion. Thus shortening the memory window is the most effective tested intervention; tapering the long-memory contribution alone helps only modestly, and combining the two interventions does not improve on memory-window cooldown alone.

G.4 Cooldown results

We first examine how cooldown changes late-training validation loss at a shared learning rate. Figures 24 and 25 show matched ADANA runs under log-time and uniform weight decay. Their validation losses agree before cooldown activates, after which cooldown improves final loss at every tested horizon. These comparisons hold the learning rate fixed within each pair; Figures 26–28 instead compare treatments after independently tuning their learning rates.

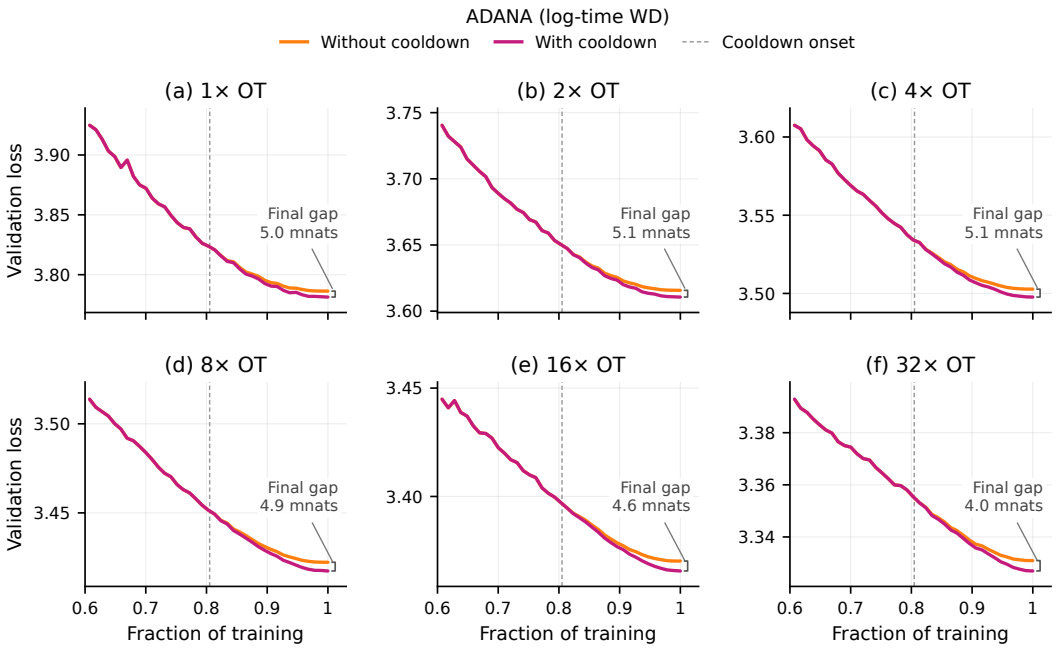

Figure 24: **Momentum cooldown improves ADANA's final validation loss across the tested horizons.** We show the final 40% of training for the 51M model with log-time weight decay at 1×–32× OT. Each pair shares the learning rate selected without cooldown, initialization, and data order, using one seed. Dashed vertical lines mark cooldown onset; annotations report final-loss reductions.

We show in Figure 28 that momentum cooldown has little consistent effect on AdamW but provides a substantial improvement to ADANA, consistent with our motivation that growing log-time memory can become stale during terminal learning rate decay. The ADANA benefit is consistent across all three model sizes.

Moreover, in our experiments on the 51M parameter model, momentum cooldown and log-time weight decay provide complementary gains. Each intervention improves long-horizon ADANA, and combining them produces the lowest validation losses at the longest measured horizons. Figure 27 reports these final-loss comparisons; Figures 52 and 53 provide the underlying learning rate sweeps.

We have not tested cooldown with Muon or SOAP, learning rate schedules other than cosine decay to zero, or crossover thresholds other than a unit timescale ratio.

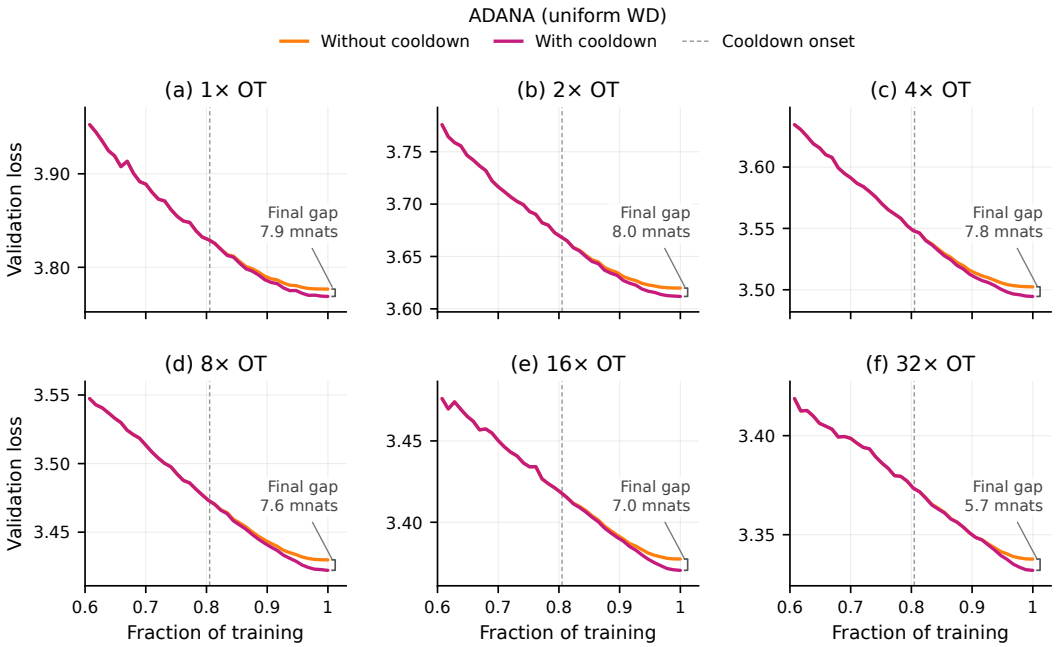

Figure 25: Same as Figure 24, with uniform weight decay.

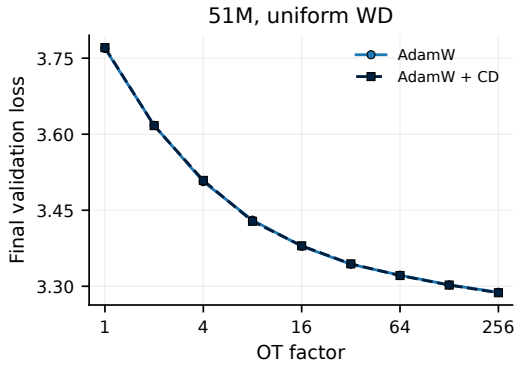

Figure 26: **Momentum cooldown has little consistent effect on AdamW.** Final validation losses for the 51M model under uniform weight decay, with and without cooldown over $1\times$–$256\times$ OT. We independently tune the base learning rate for each treatment and horizon.

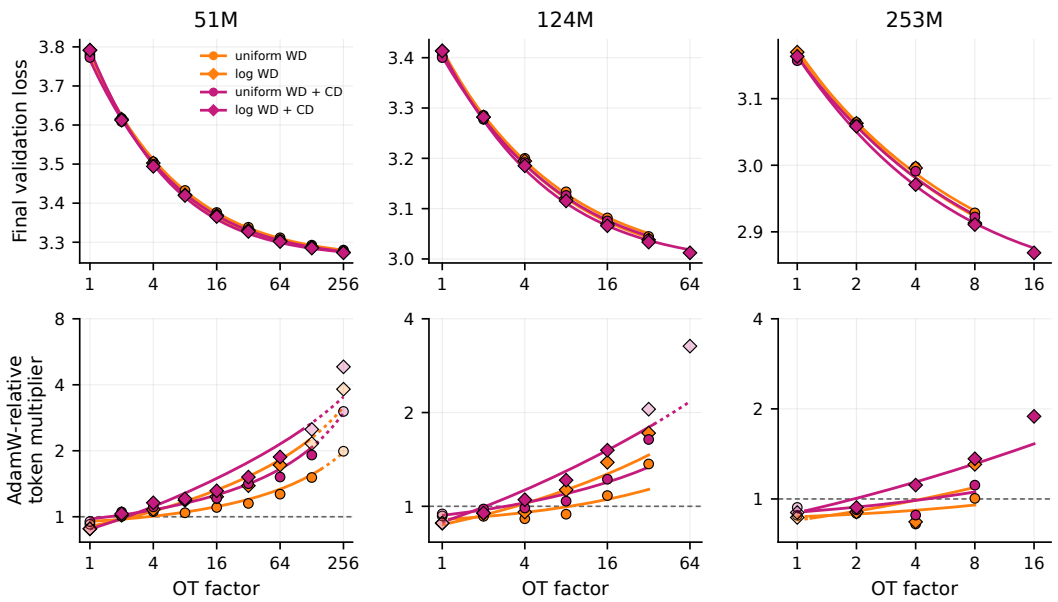

Figure 27: **Log-time weight decay and momentum cooldown provide complementary gains for ADANA.** We compare both weight decay schedules with and without cooldown across three model sizes, independently tuning the learning rate for each setting. Top: final validation losses and fitted curves. Bottom: token multipliers relative to uniform-WD AdamW at the same model size. Markers invert observed losses through the fitted baseline; curves compare fitted losses. Translucent markers and dotted segments indicate extrapolation.

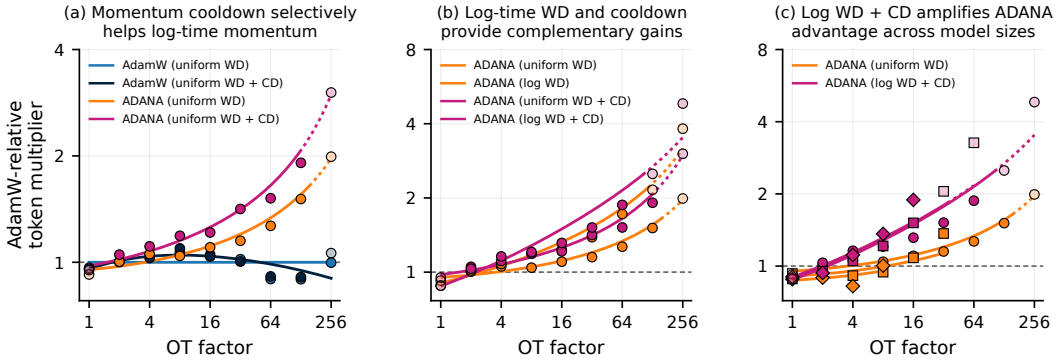

Figure 28: **Momentum cooldown benefits ADANA more than AdamW and combines with log-time weight decay.** All panels use uniform-WD AdamW as the baseline. (a) Cooldown comparisons for 51M AdamW and ADANA under uniform weight decay. (b) The four ADANA treatments at 51M. (c) ADANA with uniform WD and no cooldown versus log-time WD with cooldown across 51M, 124M, and 253M models, shown by circles, squares, and diamonds, respectively. Fit and extrapolation conventions follow Figure 27.

H SCALING FITS AND TOKEN MULTIPLIERS

This section describes how we fit loss across model sizes and training horizons and use these fits to compute token and parameter multipliers.

H.1 PRIMARY COMPARISON AND MEASURED SUPPORT

We jointly fit 156 final validation losses from seven optimizer treatments across 51M, 124M, 253M, and 458M models. These treatments include uniform and log-time weight decay for AdamW, SOAP, and ADANA with momentum cooldown, plus uniform weight decay for Muon. We exclude Muon with log-time weight decay from this joint fit and fit it separately: its nonmonotonic performance across OT suggests that log-time weight decay may be insufficiently tuned for Muon or may not work well with this optimizer. At each treatment, model size, and OT factor, we independently sweep the base learning rate and select the lowest final validation loss. Horizon coverage varies across treatments and model sizes.

Comparisons involving ADANA without cooldown or AdamW with cooldown use auxiliary fits with E and m fixed to the values obtained from this joint fit.

Table 8: Optimizer-specific settings used in the primary comparison. The settings are held fixed across model sizes and OT factors; ADANA's memory changes within training according to its prescribed log-time rule. The base learning rate is swept independently at every coordinate.

Optimizer	Prescribed optimizer-specific settings
AdamW	$\beta_1 = 0.9,\ \beta_2 = 0.98$
Muon	$\beta = 0.98$; Adam route $\beta_1 = 0.9,\ \beta_2 = 0.95$
SOAP	$\beta_1 = 0.95,\ \beta_2 = 0.98,\ \beta_{\mathrm{Sh}} = 0.95,\ K = 10$
ADANA	$g_2 = 1,\ g_3 = 8,\ \kappa = 0.85,\ \delta = 8$

We use these fits to estimate scaling exponents and high-token loss limits and to calculate token and parameter multipliers. These quantities therefore depend on the chosen functional form.

H.2 JOINT MODEL–TOKEN FUNCTIONAL FORM

Let P denote nominal model size and T denote the number of training tokens, and define the dimensionless coordinates $p = P/P_0$ and $t = T/T_0$, with $P_0 = 10^8$ parameters and $T_0 = 2 \times 10^9$ tokens. Throughout the main text, we model the loss values with the joint model–token functional form proposed by Videau et al. (2026), with the residual loss floor E and high-token model exponent m shared across treatments:

$$\mathcal{L}_o(P, T) = E + \left[A_o p^{-m/k_o} + B_o t^{-\beta_o}\right]^{k_o},$$

Here o indexes an optimizer treatment. The residual loss floor E and high-token model exponent m are shared, while A_o, B_o, β_o, and k_o are treatment specific, giving 30 fitted parameters. We refer to this specification as the shared-m Skaling fit.

At fixed model size, the high-token limit is

$$\mathcal{L}_{o,\infty}(P) = E + A_o^{k_o} p^{-m},$$

and the excess above this limit decays asymptotically as $T^{-\beta_o}$. Accordingly, we refer to β_o as the fixed-model token-decay exponent under this functional form. In the infinite-model limit, the token exponent is $k_o\beta_o$. At fixed model size, however, high-token excess loss decays with exponent β_o.

We share m to regularize fits across model sizes, whose horizon coverage is uneven, and evaluate treatment-specific model exponents as a sensitivity analysis.

H.3 Alternative functional forms and model selection

Before selecting the shared-m Skaling fit, we considered six alternative functional forms.

Our first alternative is the Chinchilla functional form, which uses separate additive terms for model size and training tokens (Hoffmann et al., 2022),

$$\mathcal{L}_o(P,T) = E + A_o p^{-\alpha_o} + B_o t^{-\beta_o}.$$

We use a shared residual floor E and treatment-specific values of A_o, α_o, B_o, and β_o.

Our second alternative is a positively constrained parameter–token interaction model motivated by the loss decompositions on power-law random features models in Paquette et al. (2024),

$$\mathcal{L}_o(P,T) = E + A_o p^{-\alpha_o} + C_o p^{-\gamma_o} t^{-\delta_o},$$

where the interaction amplitude C_o and exponents γ_o and δ_o are positive. This form allows the marginal effect of training tokens to depend on model size while preserving monotonic improvement in either coordinate.

We next compare two additive forms using the overtraining factor f,

$$\mathcal{L}_o(P,f) = E + A_o p^{-\alpha_o} + C_o p^{-u_o} f^{-\delta_o},$$

with one version sharing α across treatments and one using treatment-specific α_o.

Since $f = t/p$, the interaction term satisfies

$$p^{-u_o} f^{-\delta_o} = p^{-(u_o-\delta_o)} t^{-\delta_o}.$$

The two specifications therefore differ through their parameter constraints. The parameter–token interaction fit constrains $\gamma_o, \delta_o \in [0.01, 2]$, whereas the additive-OT fits constrain $u_o, \delta_o \in [0.01, 2]$ independently, allowing the effective interaction exponent $\gamma_o = u_o - \delta_o$ to be negative. Both additive-OT fits select negative effective interaction exponents for every treatment; the positively constrained interaction fit reaches its lower bound for every γ_o.

Finally, we compare two additional variants within the Skaling family: a version with treatment-specific high-token model exponents $m_o = k_o\alpha_o$, and a version that shares the inner exponent α while retaining treatment-specific k_o, so that $m_o = k_o\alpha$ remains treatment specific.

We compare the models using root mean squared error (RMSE), small-sample corrected Akaike information criterion (AICc), Bayesian information criterion (BIC), residual structure, and structured holdout tests. Lower AICc and BIC indicate a better tradeoff between fit error and model complexity.

In each structured holdout test, we remove a scientifically meaningful group of loss values, refit the model, and evaluate its predictions on that omitted group. We consider five tests: holding out an entire OT factor; holding out one complete model size; holding out each model size's longest horizon; holding out each model size's two longest horizons; and holding out the longest horizon from all four model sizes simultaneously. For the longest-horizon tests, we select the largest one or two distinct OT factors available at each model size, rather than each treatment's individual endpoint. The joint-frontier test removes the largest available OT factor at all four model sizes simultaneously. These tests evaluate prediction across horizon or model-size boundaries. Ordinary leave-one-loss-value-out prediction is less informative because neighboring optimizers and horizons remain in the fit.

The Chinchilla and positively constrained parameter–token interaction fits have RMSEs of 16.951 and 17.506 mnats, respectively, compared with 4.156–4.886 mnats for the additive-OT and Skaling candidates. Figure 29 also shows pronounced residual structure in the first two fits. Below, we report the structured-holdout results for the five better-fitting candidates.

For the five better-fitting models, Table 10 reports pooled RMSE in mnats for the structured holdout tests. We additionally report every fold's training count, held-out count, and RMSE in Table 11.

Table 9: In-sample fit and information criteria for the candidate scaling forms on 156 losses from seven treatments across four model sizes. The parameter-count column excludes the residual variance included in AICc and BIC.

Functional form	Parameters	RMSE (mnats)	AICc	BIC
Chinchilla	29	16.951	-1197.28	-1120.67
Positively constrained parameter–token interaction	36	17.506	-1164.28	-1075.26
Additive OT, shared model exponent	30	4.816	-1586.80	-1508.25
Additive OT, treatment-specific model exponents	36	4.244	-1606.39	-1517.37
Skaling, treatment-specific model exponents	36	4.156	-1612.89	-1523.88
Shared-m Skaling	30	4.708	-1593.88	-1515.33
Skaling, shared inner exponent	30	4.886	-1582.29	-1503.74

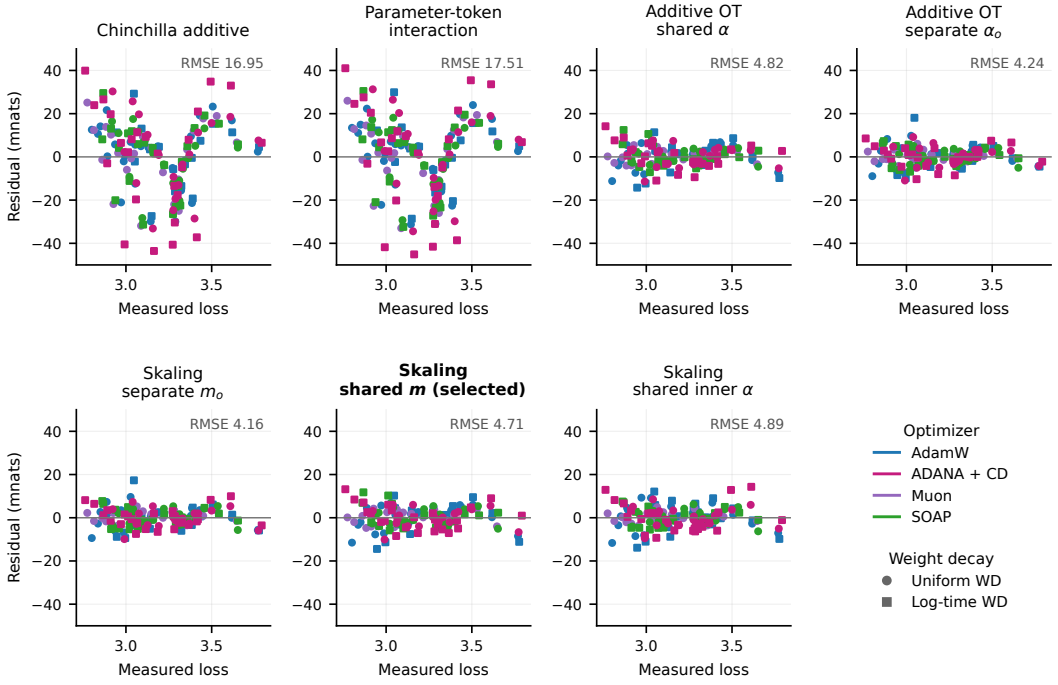

Figure 29: **The separable Chinchilla model and the positively constrained parameter–token interaction model leave systematic residuals.** Residuals are fitted minus observed validation loss in mnats. The Chinchilla-additive and positively constrained parameter–token interaction models retain a pronounced U-shaped structure, whereas the five additive-OT and Skaling candidates remain substantially closer to zero across the observed loss range.

Table 10: Pooled RMSE in mnats for the structured holdout tests for the five better-fitting models.

Functional form	Hold out OT	Hold out longest	Hold out two longest	Joint frontier	Hold out model
Additive OT, shared model exponent	7.26	7.24	8.27	7.12	16.99
Additive OT, treatment-specific model exponents	6.78	6.06	8.15	5.77	18.00
Skaling, treatment-specific model exponents	6.91	6.18	7.52	6.36	17.44
Shared-m Skaling	7.23	7.21	7.43	7.52	16.11
Skaling, shared inner exponent	7.61	7.73	7.84	7.86	14.71

Table 11: Per-fold support and RMSE in mnats for each structured holdout test for the five better-fitting models.

				Additive OT		Skaling		
Test	Held-out group	Train n	Held-out n	Shared α	Treatment-specific α_o	Treatment-specific m_o	**Shared** m	Shared α
Hold out OT	$1\times$	131	25	12.55	11.10	11.72	12.72	13.18
Hold out OT	$2\times$	131	25	5.09	4.94	5.06	5.26	5.95
Hold out OT	$4\times$	132	24	6.48	6.04	5.83	6.22	6.31
Hold out OT	$8\times$	132	24	6.71	5.71	5.84	6.76	7.34
Hold out OT	$16\times$	138	18	5.15	4.06	3.94	4.95	4.18
Hold out OT	$32\times$	142	14	6.06	8.11	7.78	5.43	6.05
Hold out OT	$64\times$	144	12	5.36	5.63	5.80	5.46	5.19
Hold out OT	$128\times$	149	7	3.96	3.91	2.87	2.80	3.93
Hold out OT	$256\times$	149	7	4.03	4.02	4.63	4.42	6.66
Hold out model	51M	93	63	24.88	26.50	25.76	23.65	21.31
Hold out model	124M	110	46	7.33	7.92	7.27	6.49	6.49
Hold out model	253M	123	33	7.71	7.30	6.70	7.14	7.04
Hold out model	458M	142	14	10.62	10.94	11.03	10.81	10.66
Hold out longest	51M	149	7	4.03	4.02	4.63	4.42	6.66
Hold out longest	124M	152	4	8.53	7.03	6.78	8.36	6.78
Hold out longest	253M	155	1	2.01	4.00	3.94	2.38	0.96
Hold out longest	458M	153	3	11.23	8.65	8.63	10.93	11.56
Hold out two longest	51M	142	14	6.19	6.19	4.91	4.52	6.68
Hold out two longest	124M	145	11	9.60	10.61	10.02	8.82	8.38
Hold out two longest	253M	151	5	8.45	7.45	6.74	7.64	6.15
Hold out two longest	458M	150	6	9.64	7.55	7.80	9.59	10.23
Joint frontier	All longest horizons	141	15	7.12	5.77	6.36	7.52	7.86

Allowing treatment-specific model exponents reduces Skaling's in-sample RMSE from 4.708 to 4.156 mnats, adds six parameters, and improves both AICc and BIC. Predictive results are mixed: separate exponents improve OT-factor, longest-horizon, and joint-frontier holdouts, whereas shared m improves whole-model and two-longest-horizon holdouts. We use shared-m Skaling as a regularized reference and report sensitivity to treatment-specific model exponents and alternative functional forms.

Predictions for a completely held-out model size are substantially less accurate than predictions across the OT axis for every candidate. Sharing m reduces the number of parameters fitted along this sparsely sampled axis. In Appendix I.1, we check whether the qualitative conclusions persist across both the Skaling and additive OT families rather than relying on a single parameterization.

The remaining residual structure, particularly for ADANA with log-time weight decay and cooldown, indicates that the selected functional form does not fully capture the measured loss surface.

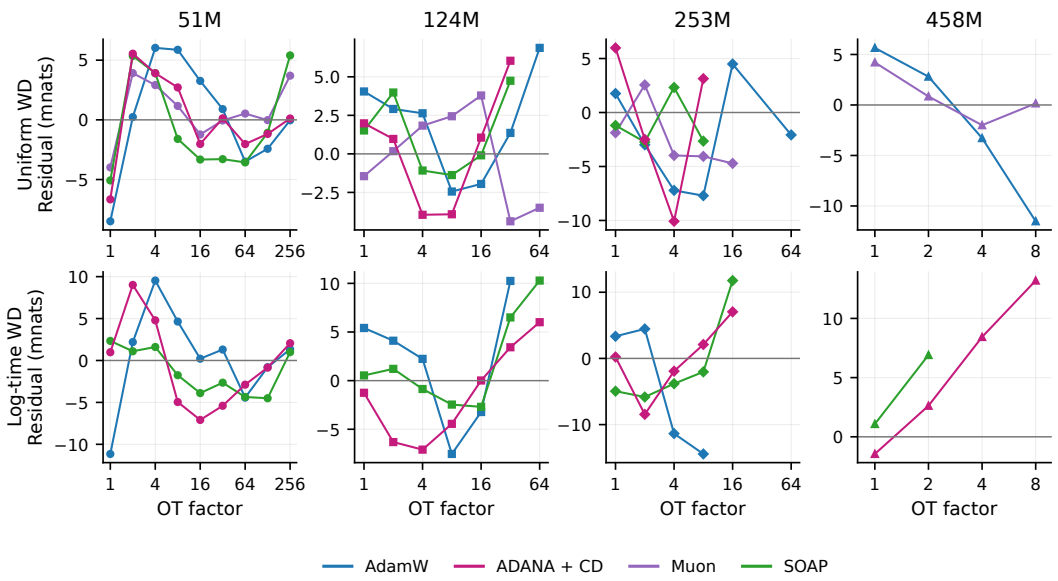

Figure 30: **The shared-m Skaling fit retains structured residuals across model sizes and training horizons.** Residuals are fitted minus observed validation loss in mnats, shown separately for the 51M, 124M, 253M, and 458M models. The top row shows uniform weight decay; the bottom row shows log-time weight decay. ADANA includes momentum cooldown in both rows. Lines connect adjacent OT factors for the same optimizer. The overall RMSE is 4.708 mnats.

H.4 Token and parameter multipliers

Following prior work, we compare optimizers using the ratio of the training tokens required by a baseline optimizer to those used by an indicated optimizer at the same loss. At fixed model size, this is the same quantity called the *compute multiplier* by Qiu et al. (2025); Ferbach et al. (2026) and *token-equivalent speedup* by Wen et al. (2026). We refer to it as the *token multiplier*.

At fixed model size p, let $t_b(\ell; p)$ denote the normalized number of training tokens required by a baseline optimizer b to reach loss ℓ. If optimizer o achieves loss ℓ_o after normalized token count t, its token multiplier relative to b is

$$M_{T,o|b}(p,t) = \frac{t_b(\ell_o; p)}{t}.$$

A value above one means that the baseline optimizer requires more training tokens to reach optimizer o's loss, while a value below one favors the baseline. For most optimizer comparisons in this paper, we use AdamW as the baseline.

Under the shared-m Skaling fit, the fitted token count required by a baseline optimizer b to reach loss ℓ is

$$\widehat{t}_b(\ell; p) = \left[\frac{B_b}{(\ell - E)^{1/k_b} - A_b p^{-m/k_b}}\right]^{1/\beta_b}.$$

This inverse is finite when ℓ lies above the baseline's fitted finite-model high-token limit,

$$\mathcal{L}_{b,\infty}(p) = E + A_b^{k_b} p^{-m}.$$

As ℓ approaches this limit from above, the required token count diverges. If ℓ falls below the fitted limit, the baseline cannot match it with any finite token count under the fitted model.

Fit-to-fit comparisons evaluate the treatment's fitted loss and invert the fitted baseline curve. Measured-to-fit comparisons instead invert the observed treatment loss and therefore depend only on the baseline fit. Where lines connect measured-to-fit markers, they connect adjacent comparisons rather than represent fit-to-fit curves.

We mark a token comparison as extrapolated when either the inferred baseline token count or the target loss lies outside the baseline's measured range at the same model size. Extrapolated comparisons use translucent marker fills and dotted curve segments.

At fixed training tokens, the parameter multiplier is the model size required by the baseline to match the treatment's loss, divided by the treatment's model size:

$$M_{P,o|b}(p,t) = \frac{\widehat{p}_b(\ell_o;t)}{p}, \qquad \mathcal{L}_b(\widehat{p}_b,t) = \ell_o.$$

Values above one mean that the baseline requires more parameters at the same token budget. We mark comparisons as extrapolated when the inferred baseline coordinate lies outside the convex hull of its measured coordinates in log-model-size–log-token space.

I Outscaling Across Training Horizons

As in the main text, we use outscaling to describe an improvement in an optimizer's relative efficiency as OT increases. Here we use the scaling fits from Appendix H to examine whether that improvement is better described by an improved token-decay exponent, a lower finite-model high-token loss limit, or both.

I.1 Definition and floor–decay tradeoff

Outscaling can arise through an improved token-decay exponent, a lower high-token validation-loss limit, or both. Optimizer treatments may approach different finite-model limits because they can select different solutions or retain different residual optimization and noise floors. In these experiments, we typically cannot identify these two effects separately, so we propose a profile-fit analysis that traces their tradeoff and determines whether the observed outscaling requires an advantage in at least one.

At fixed model size P, let $\mathcal{L}_{o,\infty}(P)$ denote optimizer o's finite-model high-token loss limit, and let β_o denote its token-decay exponent under the shared-m Skaling fit. For optimizer o and baseline b, define

$$\Delta_\infty(P) = \mathcal{L}_{b,\infty}(P) - \mathcal{L}_{o,\infty}(P), \qquad \Delta_\beta = \beta_o - \beta_b.$$

Positive Δ_∞ means that optimizer o approaches a lower finite-model loss limit, while positive Δ_β means that its excess loss decays faster with additional training tokens.

To expose this tradeoff, we fix a candidate value of Δ_∞, refit the remaining scaling parameters, and record the corresponding best-fitting value of Δ_β. Each profile therefore shows the token-decay exponent difference implied by a particular assumption about the high-token-limit difference.

At each fixed high-token-limit difference, we refit all seven treatment branches. Let $R(d)$ be the minimum residual sum of squares at $\Delta_\infty = d$, and let $R_{\min}$ be the minimum before fixing this difference. We display profile points satisfying

$$n \log\left(\frac{R(d)}{R_{\min}}\right) \le 3.84,$$

where $n = 156$ is the number of fitted losses. We use the same cutoff for all four functional forms.

We repeat this procedure for four of the alternative functional forms: the additive-OT and Skaling families each with either a model exponent shared across optimizers or optimizer-specific model exponents. This separates sensitivity to the model–token coupling from sensitivity to the model-exponent constraint.

We interpret these profiles as compatible floor–decay decompositions rather than precise independent estimates of either quantity.

I.2 Uniform weight decay without momentum cooldown

We first compare the optimizers under uniform weight decay, without momentum cooldown. Figure 31 shows final validation losses, and Figure 32 expresses these comparisons as equivalent OT factors and token multipliers relative to AdamW.

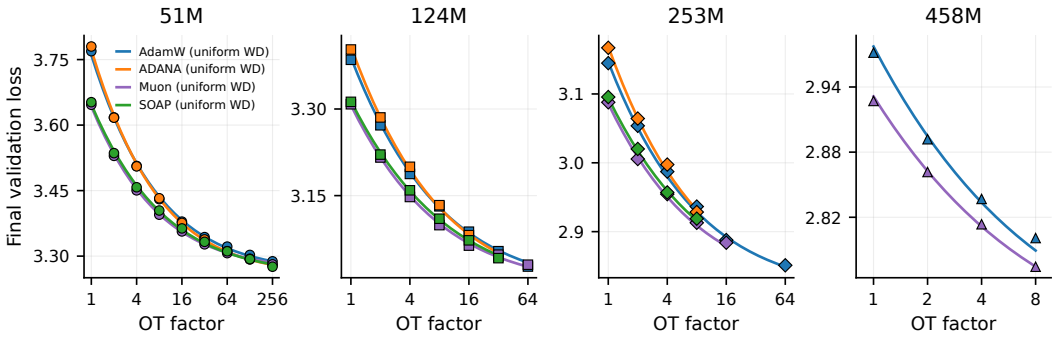

Figure 31: **Final validation losses under uniform weight decay, without momentum cooldown.** Points show the lowest final validation loss after independently sweeping the base learning rate at each optimizer, model size, and OT factor. Curves show the fitted loss functions described in Appendix H. The 458M comparison includes only AdamW and Muon.

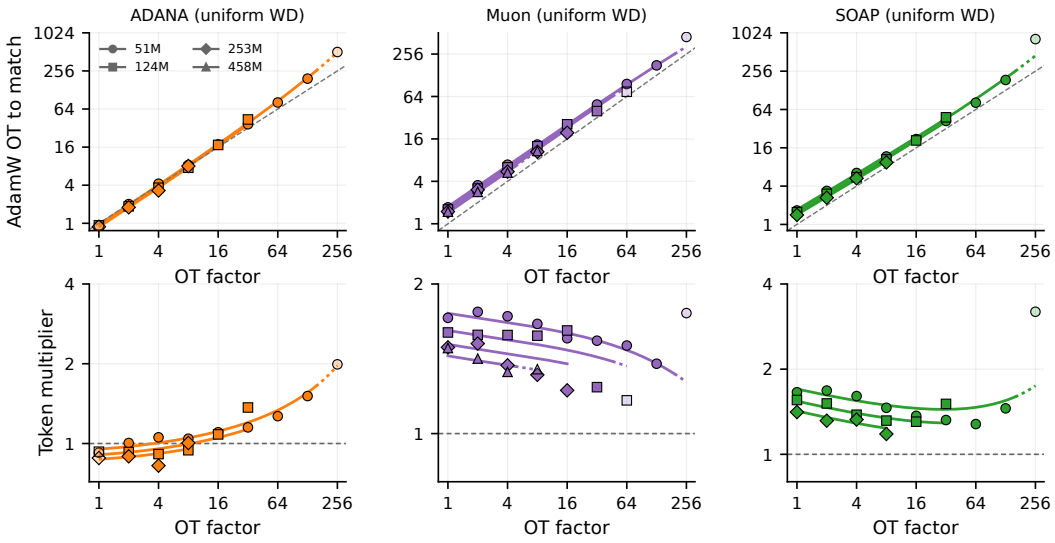

Figure 32: **Equivalent OT and token multipliers for the uniform-WD treatments in Figure 31.** Top: the AdamW OT factor required to match each optimizer's loss. Bottom: the corresponding token multiplier. Curves compare fitted losses; markers invert observed optimizer losses through the fitted AdamW curve. Dotted segments and translucent marker fills indicate comparisons outside AdamW's measured token or loss range at the same model size.

I.3 Uniform weight decay with ADANA cooldown

We next add momentum cooldown to ADANA while keeping uniform weight decay for all optimizers. Figure 33 shows final validation losses; Figures 34 and 35 give the corresponding pairwise comparisons and floor–decay decompositions.

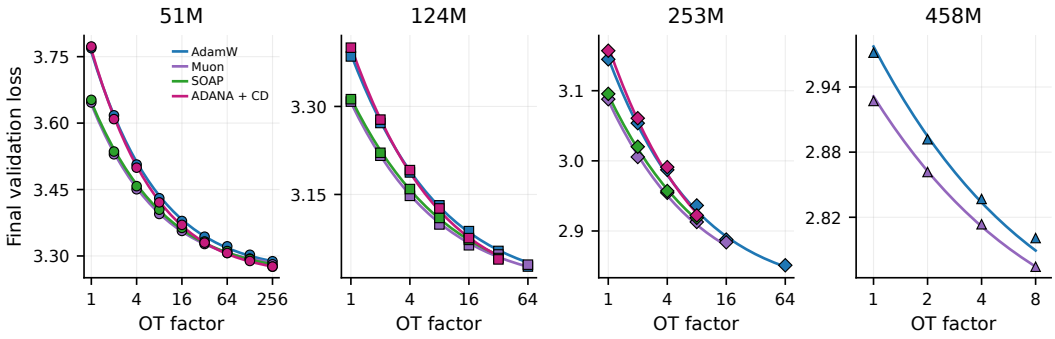

Figure 33: Same as Figure 31, with momentum cooldown applied to ADANA.

Relative to AdamW. The three non-Adam optimizers exhibit different horizon dependence (Figure 34). ADANA begins slightly behind AdamW, but its AdamW-relative token multiplier increases with overtraining factor. Muon is strongest over most short and intermediate horizons, although its advantage over AdamW remains comparatively flat. SOAP's advantage decreases through intermediate OT factors and rises at the longest horizons. The floor–decay profiles show that these trends need not have the same asymptotic explanation: ADANA must improve upon AdamW in its token-decay exponent, its high-token loss limit, or both, whereas SOAP's late gain is more compatible with a lower fitted high-token limit despite a worse token-decay exponent.

Muon versus SOAP. Muon leads through most short and intermediate horizons, while SOAP overtakes it at the longest measured horizons for the 51M and 124M models (Figure 35). Across the profiled functional forms, this crossover is consistently associated with a tradeoff: SOAP approaches a lower fitted high-token limit, while Muon retains the better token-decay exponent.

ADANA versus matrix-preconditioned optimizers. ADANA begins behind both Muon and SOAP but becomes increasingly competitive as the overtraining factor grows. ADANA has the lowest loss among these uniform-WD treatments at $128\times$ OT on the 51M model and $32\times$ OT on the 124M model, while SOAP achieves a slightly lower loss at $256\times$ OT on the 51M model. Thus, scheduled memory catches the matrix-preconditioned treatments over the measured overtraining axis without uniformly dominating them. The profile comparisons leave open whether ADANA would ultimately win through a better token-decay exponent, a lower high-token limit, or both.

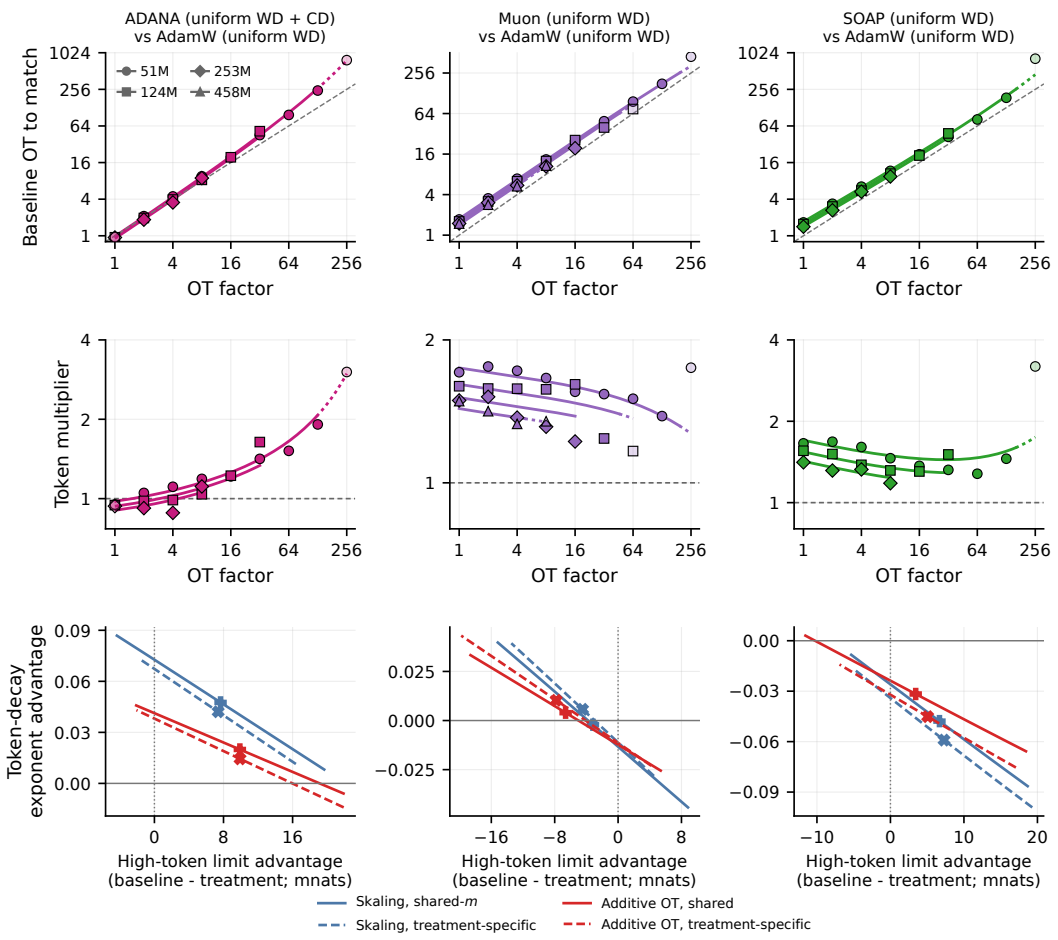

Figure 34: **Uniform-WD comparisons against AdamW, with momentum cooldown applied to ADANA.** Columns compare ADANA, Muon, and SOAP with AdamW. The top and middle rows show equivalent OT and token multipliers using the conventions of Figure 32. The bottom row shows the 51M floor–decay tradeoff under four functional forms; symbols mark best fits. Positive values on either axis favor the indicated optimizer.

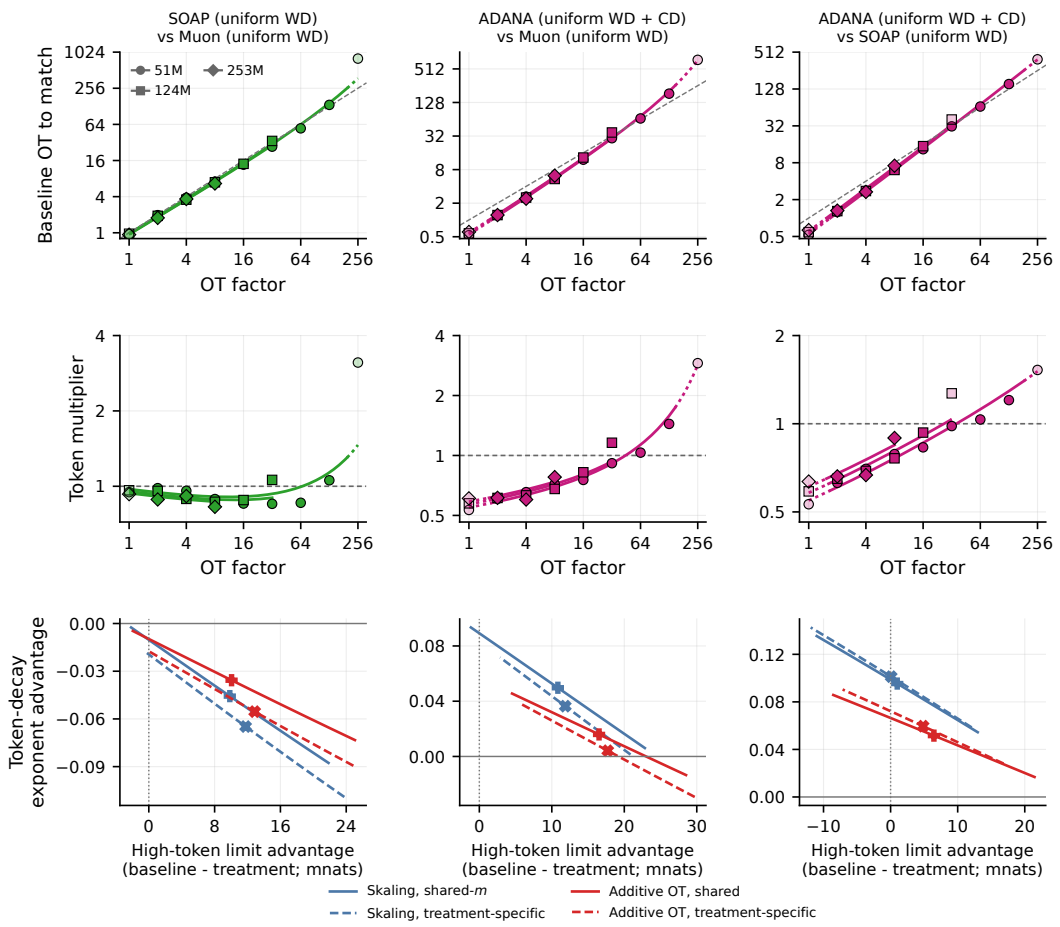

Figure 35: **Pairwise comparisons among the non-Adam uniform-WD treatments.** Columns compare SOAP with Muon, ADANA with Muon, and ADANA with SOAP. ADANA includes cooldown; conventions follow Figure 34.

I.4 Selected treatments and AdamW baseline sensitivity

We then compare ADANA with log-time weight decay and momentum cooldown, SOAP with log-time weight decay, and Muon with uniform weight decay. We use uniform-WD AdamW as the common baseline and separately examine sensitivity to using log-time-WD AdamW. These comparisons do not use horizon-tuned fixed memory, which was evaluated only on the 51M model.

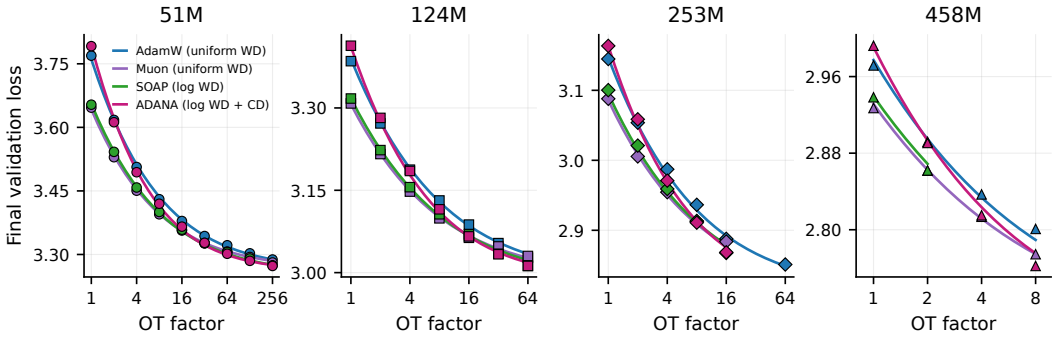

Figure 36: **Final validation losses under the selected optimizer treatments.** AdamW and Muon use uniform weight decay, SOAP uses log-time weight decay, and ADANA uses log-time weight decay with momentum cooldown. Points show the lowest final validation loss after independently sweeping the base learning rate; curves show fitted losses.

Figure 36 shows final validation losses before converting them into the pairwise comparisons in Figures 37–39. ADANA's relative token efficiency increases against AdamW and Muon as the training horizon grows. It also closes its initial gap with SOAP, although SOAP's gains at the highest OT factors make their relative scaling less clear. Across the four functional forms, the floor–decay decompositions favor an improved ADANA token-decay exponent.

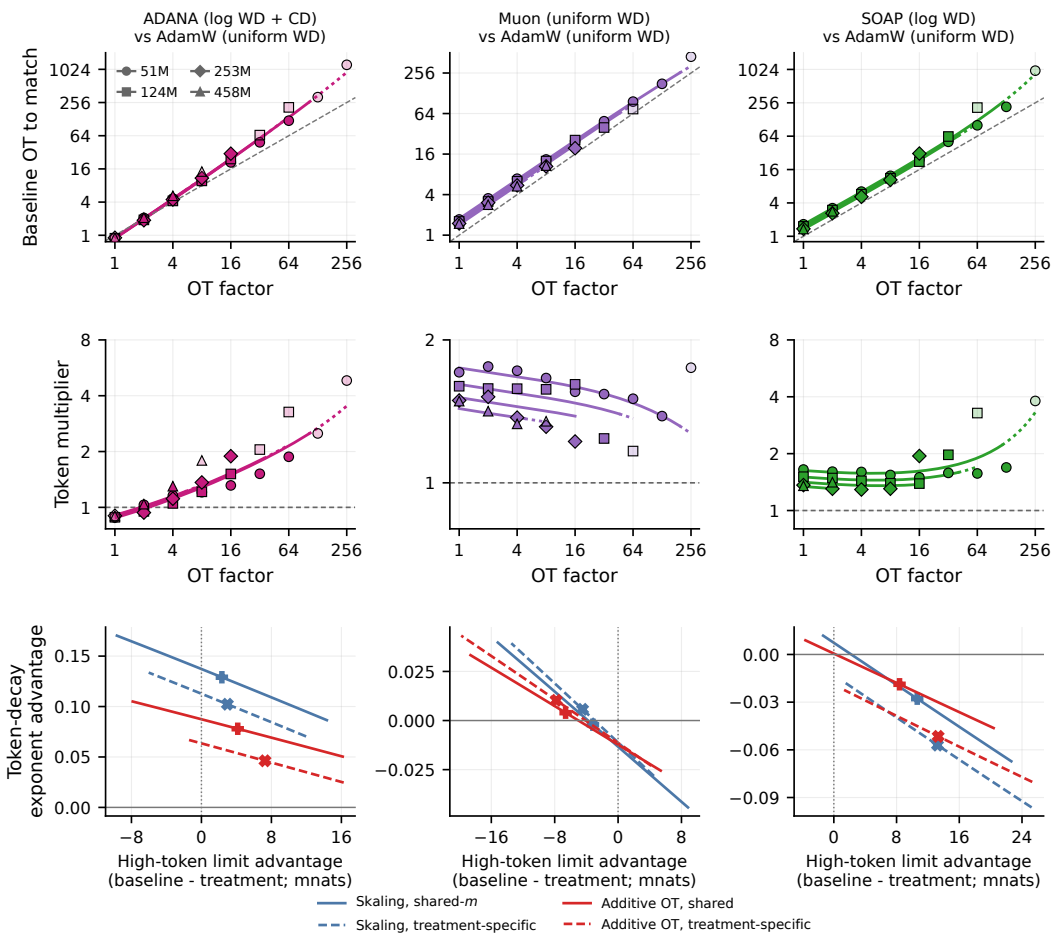

Figure 37: **Selected optimizer treatments relative to uniform-WD AdamW.** Columns compare ADANA, Muon, and SOAP with AdamW, using the treatments in Figure 36 and conventions of Figure 34.

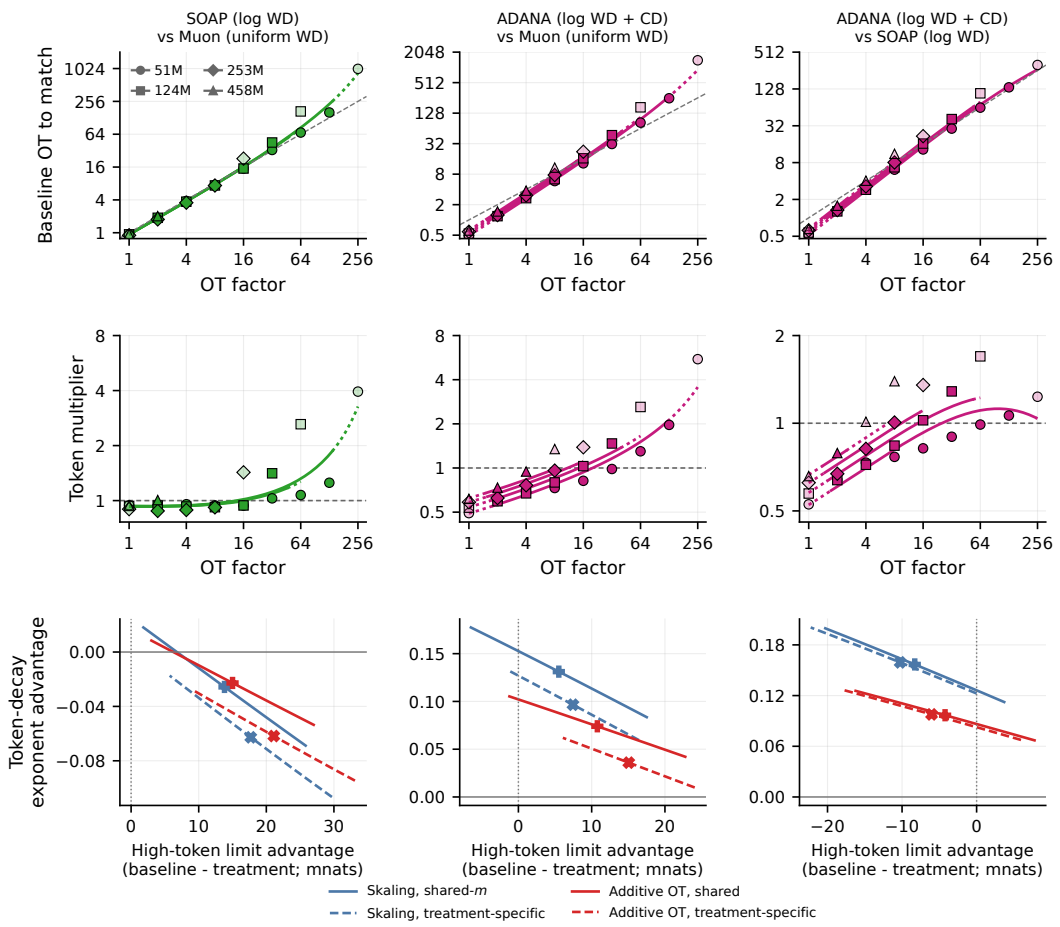

Figure 38: **Pairwise comparisons among the selected non-Adam treatments.** Columns compare SOAP with Muon, ADANA with Muon, and ADANA with SOAP, using the treatments in Figure 36 and conventions of Figure 34.

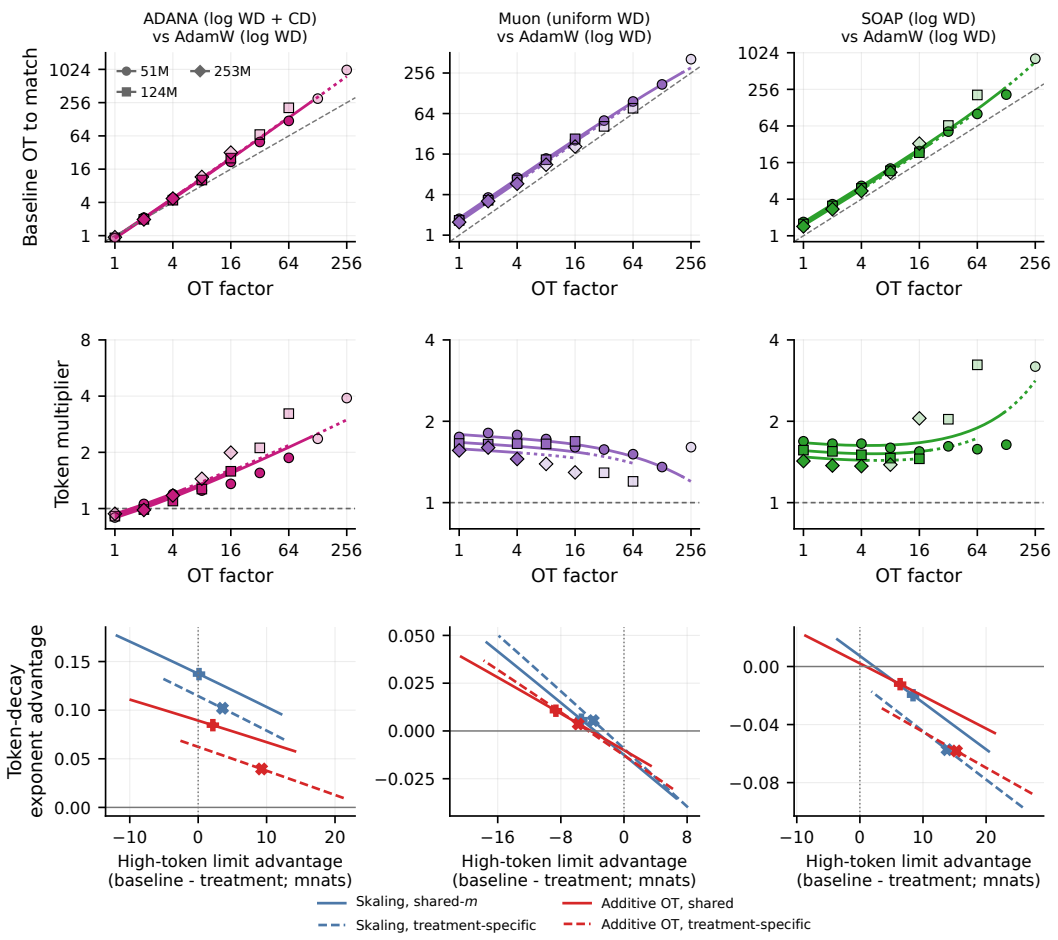

Figure 39: Same as Figure 37, using log-time-WD AdamW as the baseline. This comparison covers the 51M, 124M, and 253M models.

I.5 Equivalent OT scaling and the DANA PLRF prediction

Effective optimization time. An effective optimization clock is a reparameterization of training time that measures how much progress the optimizer has made through the problem's spectrum, rather than simply counting updates. As an example, consider gradient descent on a quadratic with a time-dependent learning rate η_t. Along an eigendirection with curvature λ, the residual evolves approximately as

$$r_{t,\lambda} \approx \exp\left(-\lambda \sum_{s<t} \eta_s\right) r_{0,\lambda}.$$

The accumulated learning rate $\sum_{s<t} \eta_s$, rather than the update count alone, therefore acts as an effective optimization clock. Two learning rate schedules that reach the same accumulated value make approximately the same progress along that eigendirection.

The PLRF analysis of DANA derives an analogous clock for its coupled gradient and momentum dynamics (Ferbach et al., 2025). In the notation of that analysis,

$$\vartheta(t) = 1 + 2\gamma_2 B t + \left(\int_0^t \sqrt{B\gamma_3(s)}\,\mathrm{d}s\right)^2,$$

where B is the batch size, γ_2 is the direct-gradient step size, and $\gamma_3(t)$ is the time-dependent step size for the momentum channel. Note that $\gamma_3(t)$ in the DANA notation is distinct from the constant ADANA coefficient g_3 used elsewhere in this paper. The term linear in t is the direct-gradient contribution to the effective clock, while the squared-integral term arises from the coupled momentum dynamics. For the DANA schedule

$$\gamma_3(t) \propto t^{-\kappa},$$

the latter term grows as

$$\left(\int_0^t s^{-\kappa/2}\,\mathrm{d}s\right)^2 \propto t^{2-\kappa}.$$

ADANA retains this time exponent through its equivalent EMA formulation, in which the momentum direction is multiplied by $(1+t)^{1-\kappa}$ (Ferbach et al., 2026).

From optimizer updates to equivalent OT. In our experiments, batch size and sequence length are fixed, so the number of optimizer updates is proportional to the number of training tokens and hence to the overtraining factor at fixed model size. If the baseline follows a clock linear in update count while ADANA follows the DANA clock above, the corresponding equivalent-OT relation is

$$f_{\text{baseline}} \propto f_{\text{ADANA}}^{2-\kappa}.$$

For our choice $\kappa = 0.85$, this gives a predicted log–log slope

$$2 - \kappa = 1.15.$$

Equivalently, DANA's effective clock grows relative to the linear baseline with exponent $1 - \kappa = 0.15$.

The exponent $2 - \kappa$ describes DANA's effective optimization clock; the resulting loss exponent also depends on the data and target spectra and on the dominant loss contribution.

Comparison with the transformer experiments. Relative to uniform-WD AdamW, the fitted equivalent-OT curves for ADANA with log-time weight decay and cooldown have log–log slopes of 1.24, 1.21, 1.19, and 1.18 for the 51M, 124M, 253M, and 458M models, respectively, over their plotted ranges (Figure 41). These slopes exceed the theoretical reference $2-\kappa = 1.15$. For uniform-WD ADANA without cooldown, the corresponding slopes are 1.12, 1.06, and 1.04 for the first three model sizes.

DANA theory compares DANA with SGD on PLRF, whereas our experiments compare ADANA with AdamW and add weight decay and cooldown schedules. Agreement in equivalent-OT exponents therefore does not identify a particular PLRF phase in transformer training.

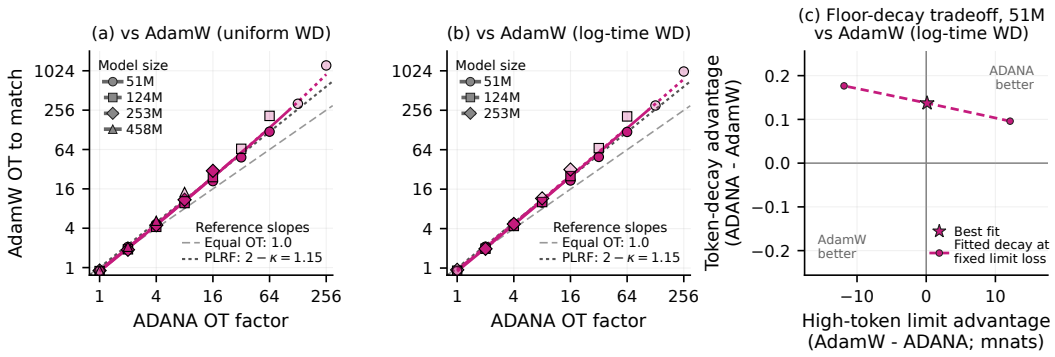

Figure 40: **ADANA with log-time weight decay and momentum cooldown outscales both AdamW baselines.** Left and middle: equivalent OT relative to AdamW with uniform and log-time weight decay, respectively. Reference lines have slopes 1 and $2 - \kappa = 1.15$; the latter is a theoretical reference, not a fitted slope or predicted prefactor. Right: the 51M floor–decay tradeoff relative to log-time-WD AdamW under shared-m Skaling. Positive values on either axis favor ADANA; the star marks the best fit.

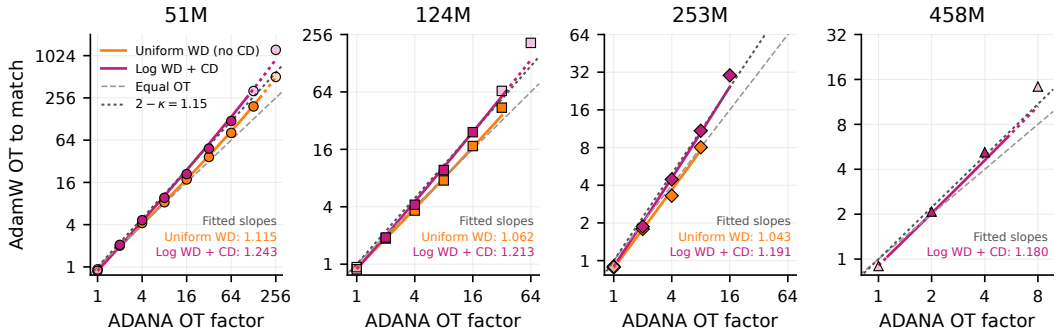

Figure 41: **ADANA's equivalent-OT scaling across model sizes and treatments.** We compare uniform-WD ADANA without cooldown against ADANA with log-time weight decay and cooldown, using uniform-WD AdamW as the baseline. Annotations report log–log slopes fitted over each curve's plotted range. Reference lines have slopes 1 and $2 - \kappa = 1.15$, both passing through $(1, 1)$. Plotting conventions follow Figure 32.

J BATCH SIZE AND OPTIMIZER UPDATE COUNT

At fixed token count, changing the batch size also changes the number of sequential optimizer updates. If T is the token budget, B is the batch size in sequences, and L is the sequence length, then the number of updates is approximately

$$S = \frac{T}{B \times L}.$$

Increasing B therefore simultaneously reduces the number of optimizer updates and the stochastic variation between updates. An optimizer can appear to scale poorly with batch either because it benefits less from larger batches or because it benefits more from the additional updates available at smaller batches. The fixed-token batch comparisons in Ferbach et al. (2026) and Marek et al. (2025) have the same limitation.

We obtain these curves by jointly optimizing the base learning rate and β_2 for AdamW, with $\beta_1 = 0.9$ fixed, and the base learning rate and g_3 for ADANA, with $\kappa = 0.85$ and $\delta = 8$ fixed. For both optimizers, we use uniform weight decay with $c_{\text{uniform}}(f) = 8\sqrt{f}$. The per-update decay coefficient is c/S, giving a parameter decay fraction $s_t c/S$. At fixed token count, holding c fixed therefore scales the coefficient with batch size (Bergsma et al., 2025a).

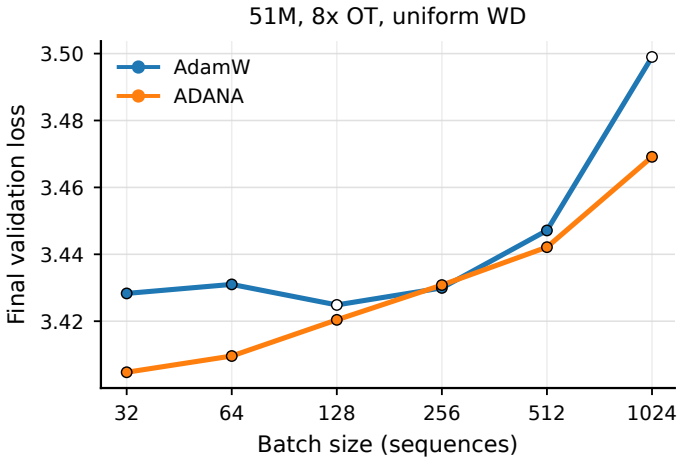

Figure 42: **Final validation loss versus batch size at fixed training tokens.** We train the 51M model for $8\times$ OT with uniform weight decay, varying the global batch size from 32 to 1024 sequences of length 2048. At each batch size, we jointly tune the base learning rate and AdamW's β_2 or ADANA's g_3. ADANA does not use cooldown. Open markers at batches 128 and 1024 indicate that near-optimal AdamW settings reach a tested hyperparameter boundary.

Figures 42 and 43 show the batch-size dependence of final validation loss and the selected hyperparameters. ADANA achieves a lower loss at $B = 32$, but its loss increases from $B = 64$ onward. AdamW remains close to its $B = 32$ loss through $B = 256$, before rising at larger batches. The AdamW selections at $B = 128$ and $B = 1024$ remain boundary-limited. Table 16 reports the final losses and selected hyperparameters.

AdamW's selected second-moment memory generally decreases with batch size, with substantial deviations from the $1/B$ guide at $B = 32$ and $B = 128$. The guide corresponds to constant token memory BLM_2, following the batch-transfer prescription of Marek et al. (2025). For ADANA, the selected g_3 increases approximately linearly with B, while its jointly selected base learning rate decreases.

ADANA's advantage may depend on the number of sequential optimizer updates. In the PLRF derivation of DANA, when $\gamma_3(t) \propto t^{-\kappa}$, the momentum contribution to the effective optimization clock grows as $BS^{2-\kappa}$, compared with a baseline contribution proportional to BS. Its relative contribution therefore grows as $S^{1-\kappa}$. This suggests that ADANA's advantage should become more visible over longer training horizons at a fixed batch size. At fixed tokens, however, increasing B reduces S and may obscure this advantage.

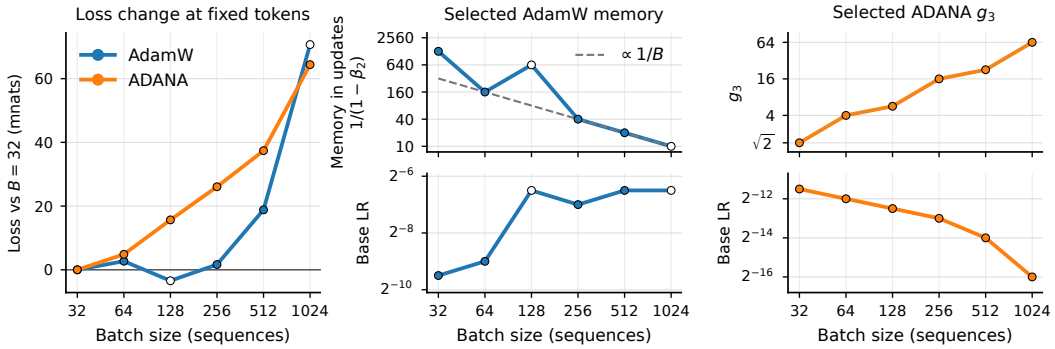

Figure 43: **ADANA's loss begins to rise at smaller batch sizes than AdamW's.** We use the experiment in Figure 42. Left: loss relative to each optimizer's own $B = 32$ value. Middle: selected AdamW memory $M_2 = 1/(1-\beta_2)$; the $1/B$ guide corresponds to constant token memory BLM_2, where $L = 2048$. Right: selected ADANA g_3. Lower panels show the jointly selected base learning rates. Open markers follow Figure 42.

ADANA may also be particularly effective in the high-noise regime. Analyses of accelerated-SGD-style methods suggest that some multi-timescale methods derive their largest advantage from noisy, small-batch gradients (Morwani et al., 2025). Increasing the effective batch size could then reduce ADANA's advantage independently of the accompanying reduction in optimizer updates.

Broader joint hyperparameter optimization may also change the relative batch scaling. Although the optimal g_3 changes with batch size, δ controls the averaging window, while g_3 and κ control the strength and growth of the momentum contribution. Similarly, optimizing AdamW's β_2 at fixed β_1 does not test whether its first- and second-moment timescales should be transferred together. We have separately studied how ADANA's optimal weight-decay coefficient scales with training horizon at a fixed batch size, but not how it changes across batch sizes. Additional batch-conditioned weight-decay sweeps could therefore test whether weight-decay transfer contributes to the observed performance differences.

Finally, in language model training, each batch element is a sequence rather than an independent token, making the appropriate notion of effective batch size ambiguous. One natural measure is the total number of tokens processed per update, $B \times L$. However, tokens within the same sequence share context and are likely more correlated than tokens drawn from different batch elements. Two configurations with the same $B \times L$ may therefore have different gradient noise: increasing B while shortening L may provide more independent information than decreasing B while lengthening L. Comparing different (B, L) factorizations at matched tokens per update would help identify which notion of batch size governs optimizer performance.

Varying horizon at fixed batch, batch at fixed update count, and batch at fixed tokens would help distinguish how update count and batch size affect relative efficiency.

K Complete Hyperparameter Sweeps

This section reports the learning rate and nested-hyperparameter sweeps underlying our comparisons. Stars mark the minimum final validation loss within each sweep; upward triangles indicate losses above the displayed range. Solid curves with filled markers satisfy the 1-mnat interiority criterion in Section B.5; dashed curves with hollow markers indicate boundary-limited sweeps.

K.1 Final validation losses

The following tables report final validation losses at the best learning rate, rounded to three decimal places, along with the best learning rate exponent. A dash indicates no measurement. In Tables 12–14, parentheses give the selected $\log_2$ base learning rate.

Table 12: Uniform weight decay without momentum cooldown. Loss is in nats per token.

Model	OT	AdamW	ADANA	Muon	SOAP
51M	1	3.769 (−7.5)	3.780 (−13)	3.647 (−6)	3.652 (−5.5)
	2	3.618 (−8)	3.617 (−13)	3.530 (−5)	3.536 (−5.5)
	4	3.506 (−8)	3.506 (−13)	3.451 (−6)	3.458 (−6)
	8	3.430 (−7)	3.432 (−12.5)	3.395 (−5.5)	3.404 (−5.5)
	16	3.379 (−7)	3.376 (−12)	3.357 (−5)	3.363 (−6)
	32	3.344 (−5.5)	3.338 (−12)	3.327 (−5.5)	3.333 (−6)
	64	3.322 (−7.5)	3.311 (−12.5)	3.307 (−6.5)	3.311 (−7)
	128	3.303 (−5.5)	3.292 (−12)	3.294 (−7.5)	3.293 (−6)
	256	3.288 (−8)	3.280 (−12.5)	3.281 (−8)	3.276 (−7)
124M	1	3.385 (−8)	3.403 (−13)	3.308 (−6.5)	3.312 (−6)
	2	3.272 (−7)	3.285 (−13)	3.216 (−5.5)	3.221 (−5.5)
	4	3.188 (−8)	3.200 (−12.5)	3.148 (−6)	3.159 (−5.5)
	8	3.131 (−7)	3.133 (−15)	3.099 (−6)	3.110 (−6)
	16	3.088 (−7)	3.081 (−14)	3.063 (−7)	3.073 (−8.5)
	32	3.054 (−8.5)	3.045 (−13.5)	3.048 (−6)	3.042 (−8)
	64	3.027 (−9)	—	3.030 (−7)	—
253M	1	3.145 (−8.5)	3.167 (−13.5)	3.088 (−6.5)	3.096 (−7)
	2	3.054 (−8.5)	3.064 (−12)	3.006 (−6.5)	3.020 (−5)
	4	2.987 (−7.5)	2.997 (−11.5)	2.954 (−6.5)	2.957 (−6.5)
	8	2.936 (−7)	2.928 (−15)	2.913 (−6.5)	2.919 (−5.5)
	16	2.888 (−10)	—	2.884 (−7.5)	—
	64	2.851 (−7.5)*	—	—	—
458M	1	2.972 (−9)	—	2.927 (−6.5)	—
	2	2.892 (−9)	—	2.862 (−8.5)	—
	4	2.837 (−8.5)	—	2.814 (−6.5)	—
	8	2.801 (−7.5)	—	2.774 (−7.5)	—

*The 253M, 64× AdamW point is used in fitting and appendix comparisons but omitted from the main-text plots.

Table 13: Log-time weight decay without momentum cooldown. Muon with log-time weight decay is included here although it is excluded from the primary joint fit. Loss is in nats per token.

Model	OT	AdamW	ADANA	Muon	SOAP
51M	1	3.776 (−8)	3.791 (−13.5)	3.652 (−6)	3.653 (−7)
	2	3.620 (−8)	3.615 (−13.5)	3.531 (−6)	3.542 (−8)
	4	3.507 (−9)	3.503 (−13)	3.450 (−6)	3.458 (−5.5)
	8	3.435 (−8.5)	3.421 (−12)	3.394 (−5.5)	3.400 (−5.5)
	16	3.384 (−7.5)	3.369 (−11.5)	3.355 (−5.5)	3.358 (−6)
	32	3.344 (−7.5)	3.331 (−13)	3.325 (−6)	3.326 (−7)
	64	3.323 (−7)	3.304 (−12.5)	3.318 (−5.5)	3.306 (−5.5)
	128	3.301 (−7)	3.287 (−10.5)	3.308 (−6.5)	3.291 (−7)
	256	3.286 (−8)	3.274 (−13)	3.332 (−6)	3.274 (−7)
124M	1	3.390 (−9.5)	3.414 (−13)	3.313 (−6.5)	3.317 (−6)
	2	3.276 (−9)	3.282 (−13)	3.214 (−6.5)	3.223 (−5.5)
	4	3.192 (−8)	3.195 (−12.5)	3.143 (−7)	3.156 (−5.5)
	8	3.140 (−9)	3.120 (−15)	3.095 (−7)	3.106 (−6)
	16	3.091 (−7)	3.070 (−14.5)	3.068 (−8)	3.070 (−5.5)
	32	3.046 (−10)	3.038 (−13.5)	3.047 (−6.5)	3.034 (−8.5)
	64	—	—	—	3.012 (−8.5)
253M	1	3.150 (−9)	3.170 (−13.5)	3.089 (−6.5)	3.100 (−7)
	2	3.052 (−8.5)	3.063 (−12.5)	3.001 (−6.5)	3.021 (−6)
	4	2.996 (−7.5)	2.996 (−13.5)	2.942 (−8)	2.959 (−6.5)
	8	2.947 (−8.5)	2.913 (−15.5)	2.903 (−9)	2.913 (−6)
	16	—	—	—	2.868 (−9)
458M	1	—	—	—	2.938 (−6)
	2	—	—	—	2.862 (−7.5)

Table 14: Momentum cooldown treatments. CD denotes momentum cooldown; corresponding treatments without cooldown appear in Tables 12 and 13. Loss is in nats per token.

Model	OT	AdamW Uniform WD + CD	ADANA Uniform WD + CD	ADANA Log-time WD + CD
51M	1	3.771 (−8)	3.773 (−13)	3.792 (−13.5)
	2	3.617 (−7.5)	3.609 (−13.5)	3.612 (−13.5)
	4	3.509 (−7.5)	3.499 (−13)	3.494 (−13)
	8	3.428 (−7)	3.421 (−12.5)	3.420 (−12)
	16	3.380 (−7)	3.370 (−11.5)	3.366 (−11)
	32	3.344 (−5.5)	3.330 (−10.5)	3.328 (−13)
	64	3.321 (−7.5)	3.307 (−12.5)	3.302 (−12.5)
	128	3.302 (−5.5)	3.289 (−12)	3.285 (−11)
	256	3.287 (−8)	3.276 (−12.5)	3.273 (−13)
124M	1	—	3.400 (−13.5)	3.413 (−13.5)
	2	—	3.278 (−13.5)	3.282 (−13.5)
	4	—	3.191 (−12.5)	3.185 (−15.5)
	8	—	3.126 (−14.5)	3.115 (−15)
	16	—	3.076 (−13.5)	3.066 (−14.5)
	32	—	3.039 (−13.5)	3.033 (−15)
	64	—	—	3.012 (−15)
253M	1	—	3.157 (−13.5)	3.163 (−13.5)
	2	—	3.061 (−12)	3.059 (−16.5)
	4	—	2.991 (−11)	2.971 (−16)
	8	—	2.922 (−15)	2.911 (−15.5)
	16	—	—	2.869 (−15.5)
458M	1	—	—	2.992 (−13.5)
	2	—	—	2.891 (−16.5)
	4	—	—	2.815 (−16)
	8	—	—	2.762 (−16)

Table 15: Horizon-tuned fixed memory on the 51M model with uniform weight decay. Each optimizer independently selects the best memory window and base learning rate at each OT factor. $M = 1/(1-\beta)$, where β denotes AdamW/SOAP β_2 or Muon momentum. Loss is in nats per token.

Optimizer	OT	Final loss	$\log_2$ base LR	M
AdamW	1	3.766	-8	20
	2	3.611	-8	20
	4	3.508	-8	20
	8	3.430	-7	40
	16	3.380	-7	40
	32	3.344	-6.5	40
	64	3.319	-5.5	80
	128	3.299	-7.5	1280
Muon	1	3.639	-5	20
	2	3.528	-6	40
	4	3.450	-5	40
	8	3.394	-5.5	80
	16	3.356	-6	160
	32	3.326	-7	320
	64	3.305	-7	320
	128	3.290	-8	640
SOAP	1	3.650	-5.5	40
	2	3.539	-5.5	20
	4	3.457	-6	40
	8	3.403	-5.5	80
	16	3.362	-6	40
	32	3.331	-6	640
	64	3.308	-6	160
	128	3.292	-7	1280

Table 16: Batch-size study on the 51M model at $8\times$ OT with uniform weight decay. We report final losses and jointly selected hyperparameters. Batch size counts sequences of length 2048; loss is in nats per token.

Optimizer	Batch	Final loss	$\log_2$ base LR	M_2	g_3
AdamW	32	3.428	-9.5	1280	—
	64	3.431	-9	160	—
	128	3.425*	-6.5	640	—
	256	3.430	-7	40	—
	512	3.447	-6.5	20	—
	1024	3.499*	-6.5	10	—
ADANA	32	3.405	-11.5	—	$\sqrt{2}$
	64	3.410	-12	—	4
	128	3.420	-12.5	—	$4\sqrt{2}$
	256	3.431	-13	—	16
	512	3.442	-14	—	$16\sqrt{2}$
	1024	3.469	-16	—	64

*Boundary-limited AdamW selections at batches 128 and 1024, corresponding to open markers in the figures. The reported minima are measured values, not extrapolations.

K.2 Learning rate schedule selection

Table 17: Fixed memory settings and coefficients of the uniform WD schedule for the linear-to-zero and cosine-to-zero comparison on the 51M parameter model. At every listed coefficient, each schedule receives a separate base learning rate sweep.

Optimizer	Fixed memory settings	c at $1\times$	c at $8\times$	c at $32\times$
AdamW	$\beta_1 = 0.9,\ \beta_2 = 0.95$	$\{2, 4, 8, 16\}$	$\{4, 8, 16, 32\}$	$\{16, 32, 64, 128\}$
ADANA	$g_3 = 8,\ \kappa = 0.85$	$\{4, 8, 16\}$	$\{8, 16, 32, 64\}$	$\{16, 32, 64, 128\}$
Muon	$\beta = 0.95$	$\{2, 4, 8, 16\}$	$\{8, 16, 32, 64\}$	$\{16, 32, 64, 128\}$
SOAP	$\beta_1 = 0.95,\ \beta_2 = 0.99,\ \beta_{\text{Sh}} = 0.95$	$\{2, 4, 8, 16\}$	$\{4, 8, 16, 32, 64\}$	$\{16, 32, 64, 128\}$

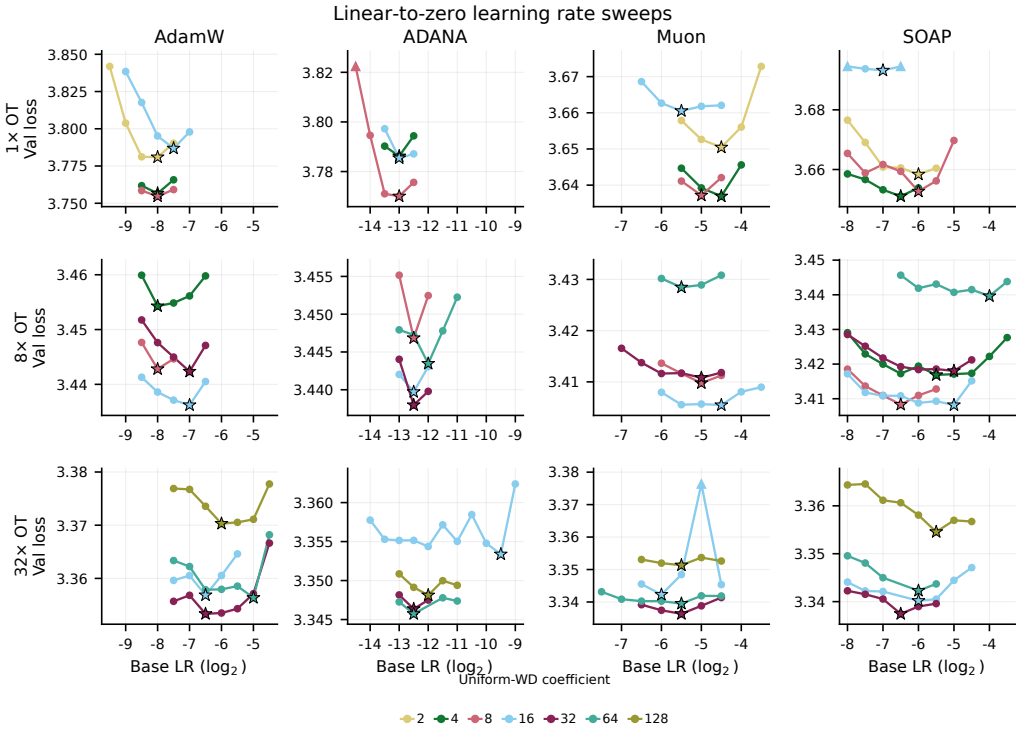

Figure 44: **Base learning rate sweeps at each uniform WD coefficient on the 51M parameter model with linear decay to zero.** Each colored curve holds the WD coefficient fixed while varying the base learning rate; stars mark the minimum final validation loss within each sweep. Momentum hyperparameters are AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$), ADANA ($g_3 = 8, \kappa = 0.85$), Muon ($\beta = 0.95$), and SOAP ($\beta_1 = 0.95, \beta_2 = 0.99, \beta_{\text{Sh}} = 0.95$). All 48 displayed sweeps satisfy the 1-mnat interiority criterion.

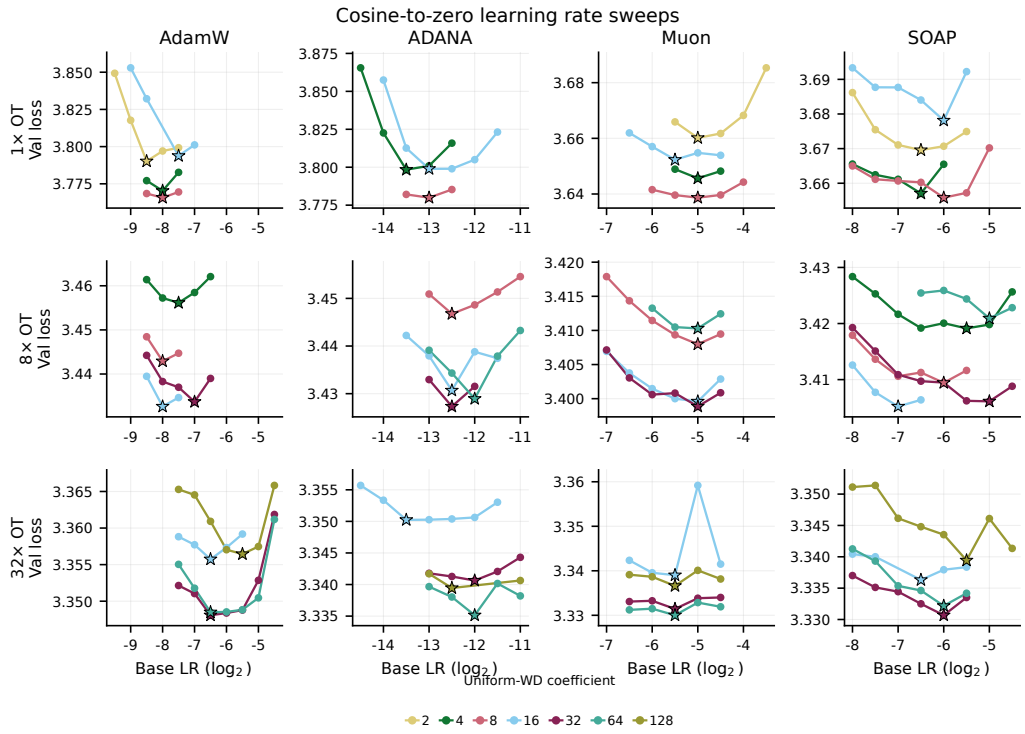

Figure 45: **Base learning rate sweeps at each uniform WD coefficient on the 51M parameter model with cosine decay to zero.** Each colored curve holds the WD coefficient fixed while varying the base learning rate; stars mark the minimum final validation loss within each sweep. Solid lines with filled markers denote sweeps that satisfy the 1-mnat interiority criterion. Momentum hyperparameters are AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$), ADANA ($g_3 = 8, \kappa = 0.85$), Muon ($\beta = 0.95$), and SOAP ($\beta_1 = 0.95, \beta_2 = 0.99, \beta_{\text{Sh}} = 0.95$). These sweeps provide the uniform WD grid used in the learning rate schedule comparison. All 48 displayed sweeps satisfy the 1-mnat interiority criterion.

K.3 WEIGHT DECAY SWEEPS

The WD schedule comparison and coefficient scaling analysis in Appendix E use nested sweeps. At every WD coefficient, we first select the minimum of an independently extended base learning rate sweep; only after each contributing learning rate sweep satisfies the interiority criterion in Section B.5 do we compare coefficients. The summary figures in Appendix E collapse this pointwise evidence by minimizing over the learning rate. Here we report the complete underlying grids.

Table 18 gives the fixed momentum treatment and tested coefficients for every optimizer, WD schedule, and OT factor.

Table 18: WD coefficient grids for the uniform and log-time WD sweep figures. Each OT cell lists the coefficients c included in the corresponding base learning rate sweeps.

		WD coefficient c			
Optimizer	Fixed memory settings	$1\times$ OT	$8\times$ OT	$32\times$ OT	$64\times$ OT
Uniform WD					
AdamW	$\beta_1 = 0.9,\ \beta_2 = 0.98$	$4, 8, 16$	$8, 16, 32$	$16, 32, 64, 128$	$16, 32, 64, 128$
ADANA	$g_3 = 8,\ \kappa = 0.85$	$4, 8, 16$	$8, 16, 32, 64$	$16, 32, 64, 128$	$16, 64, 128, 256$
Muon	$\beta = 0.98$	$4, 8, 16$	$8, 16, 32, 64$	$16, 32, 64, 128$	$16, 32, 64, 128, 256$
SOAP	$\beta_1 = 0.95,\ \beta_2 = 0.98,\ \beta_{\mathrm{Sh}} = 0.95$	$4, 8, 16$	$8, 16, 16\sqrt{2}, 32$	$16, 32, 32\sqrt{2}, 64$	$8, 32, 64, 128$
Log-time WD					
AdamW	$\beta_1 = 0.9,\ \beta_2 = 0.98$	$1, 2, 4, 8$	$1, 2, 4, 4\sqrt{2}, 8$	$1, 2, 4, 8, 16$	$2, 8, 16, 32, 64$
ADANA	$g_3 = 8,\ \kappa = 0.85$	$1, 2, 4, 8$	$1, 2, 4, 8, 16$	$1, 2, 4, 8, 16, 32$	$2, 8, 16, 32, 64$
Muon	$\beta = 0.98$	$1, 2, 4, 8$	$1, 2, 4, 8$	$1, 2, 4, 8, 16$	$2, 4, 8, 16, 32$
SOAP	$\beta_1 = 0.95,\ \beta_2 = 0.98,\ \beta_{\mathrm{Sh}} = 0.95$	$1, 2, 4, 8$	$2, 4, 4\sqrt{2}, 8$	$1, 2, 4, 8, 16, 32$	$1, 2, 4, 8, 16, 32$

Log-time WD coefficient sweeps. Figure 47 reports the corresponding $1\times$-anchored log-time WD coefficient sweeps.

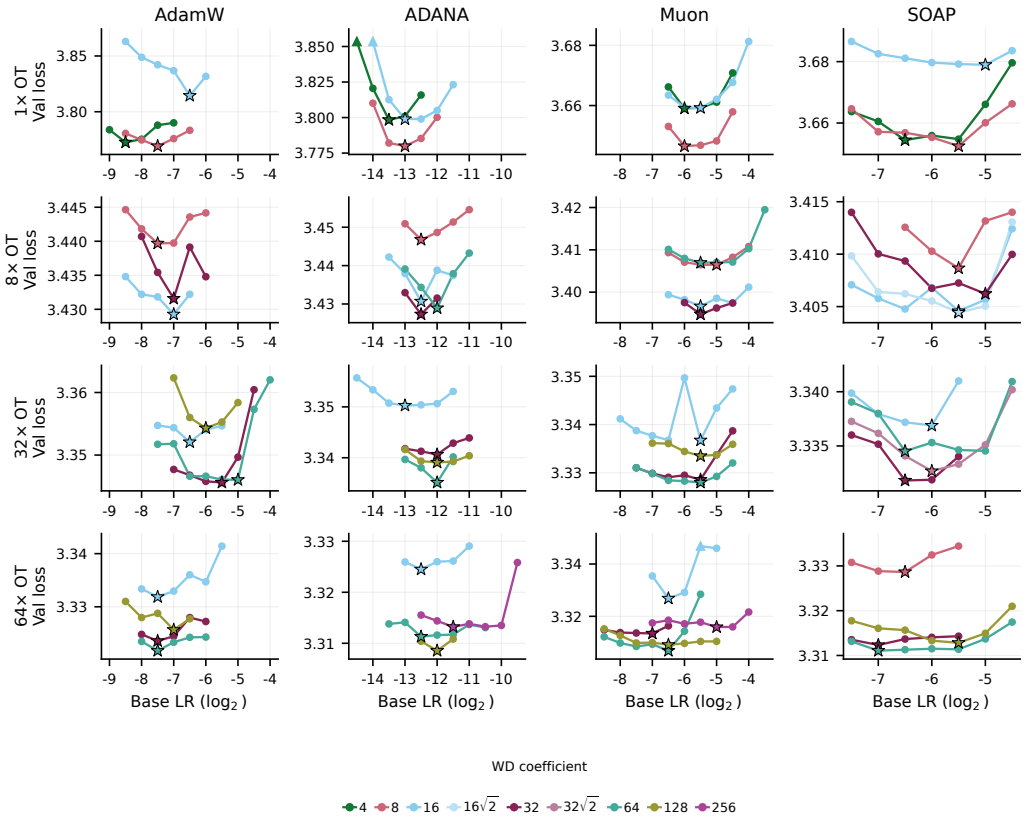

Figure 46: **Base learning rate sweeps at each uniform WD coefficient on the 51M parameter model with cosine decay to zero.** Columns are optimizers and rows are OT factors. Each colored curve holds c fixed; circles are measured learning rates and black-edged stars mark the minimum within each sweep. All 60 sweeps satisfy the interiority criterion in Section B.5. Facets use independent vertical ranges to expose the local learning rate basin; upward triangles retain observed losses above a facet's displayed range.

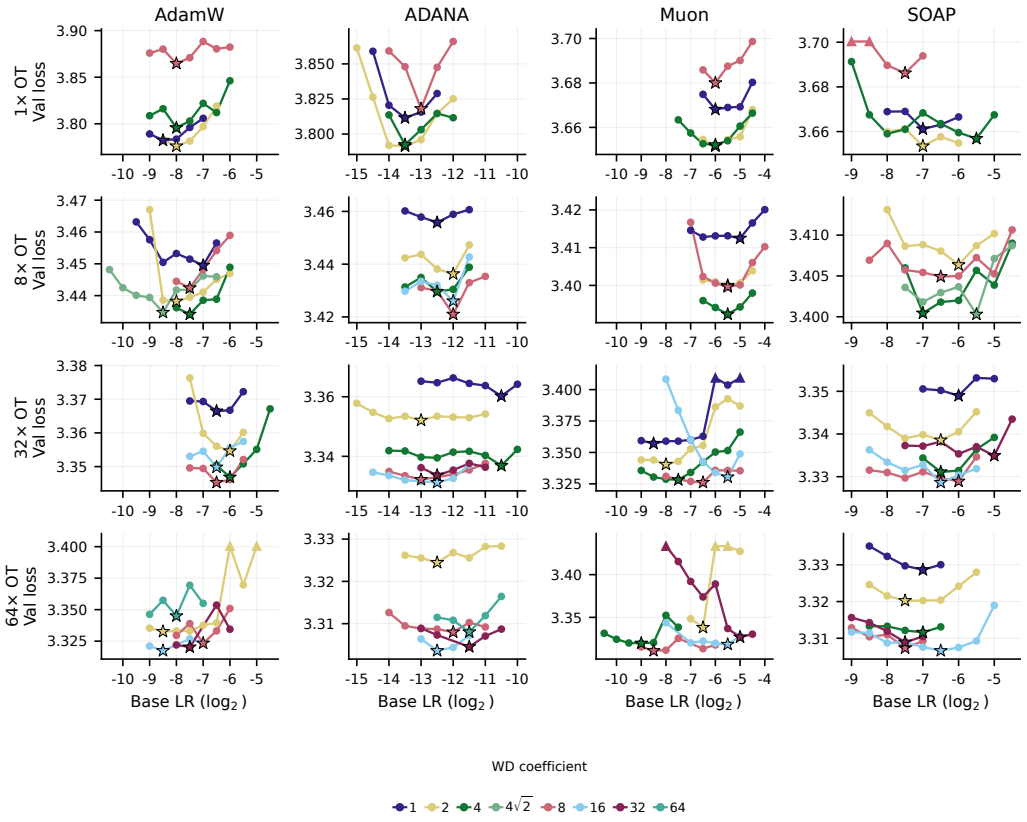

Figure 47: **Base learning rate sweeps at each log-time WD coefficient on the 51M parameter model with cosine decay to zero.** Columns are optimizers and rows are OT factors. Each colored curve holds c fixed; circles are measured learning rates and black-edged stars mark the minimum within each sweep. We use the $1\times$-anchored offset $\tau = 0.1S_{1\times}$. All 77 sweeps satisfy the interiority criterion in Section B.5. Facets use independent vertical ranges, and upward triangles retain observed losses above a facet's displayed range.

K.4 UNIFORM-WD AND LOG-TIME-WD SCALING SWEEPS

Figures 48 and 49 show every independently tuned learning rate sweep used in the uniform-WD and log-time-WD scaling comparisons.

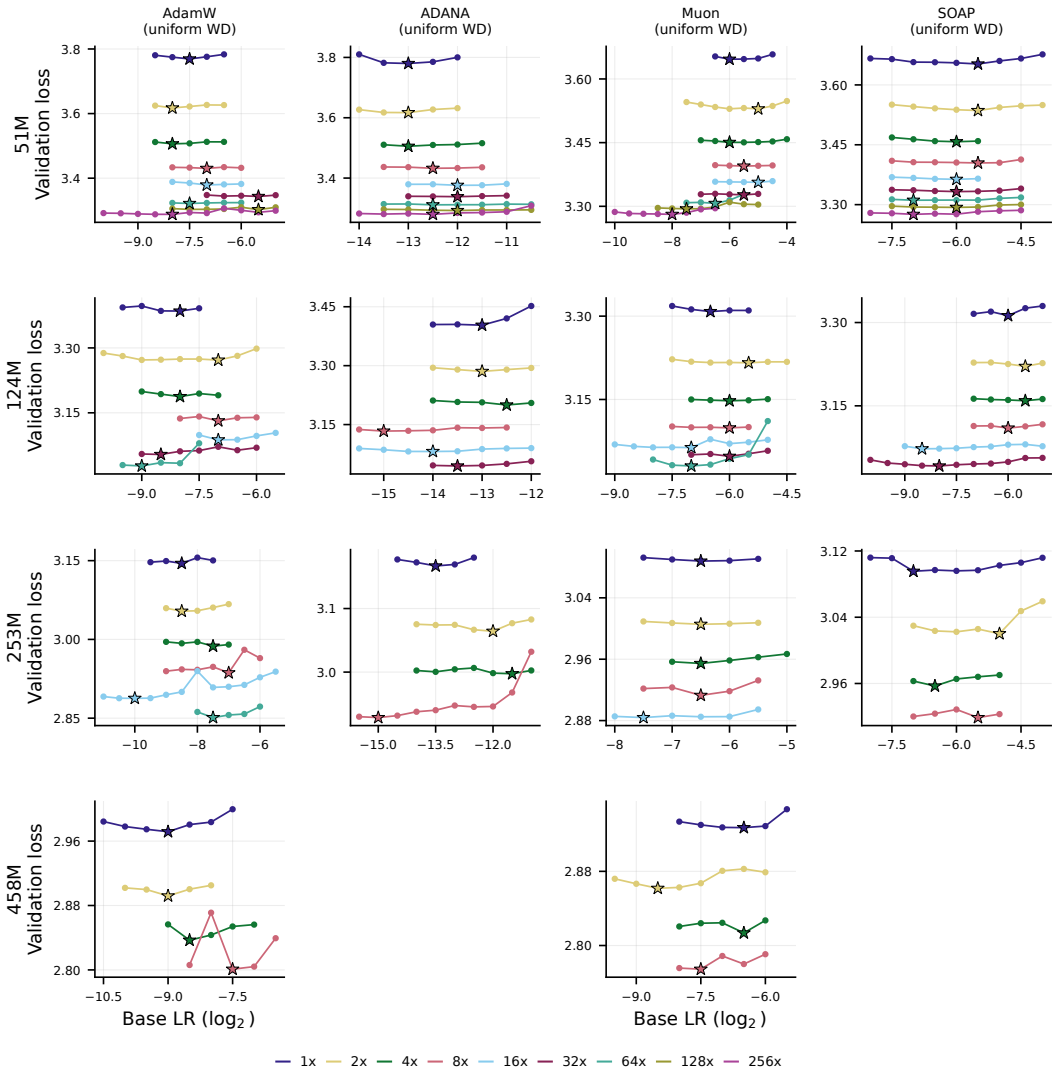

Figure 48: **Learning rate sweeps with uniform weight decay.** Columns show optimizers, rows show model sizes, and colors identify OT factors. We use cosine learning rate decay to zero and $c_{\text{uniform}}(f) = 8\sqrt{f}$, without momentum cooldown. All 89 sweeps satisfy the 1-mnat interiority criterion.

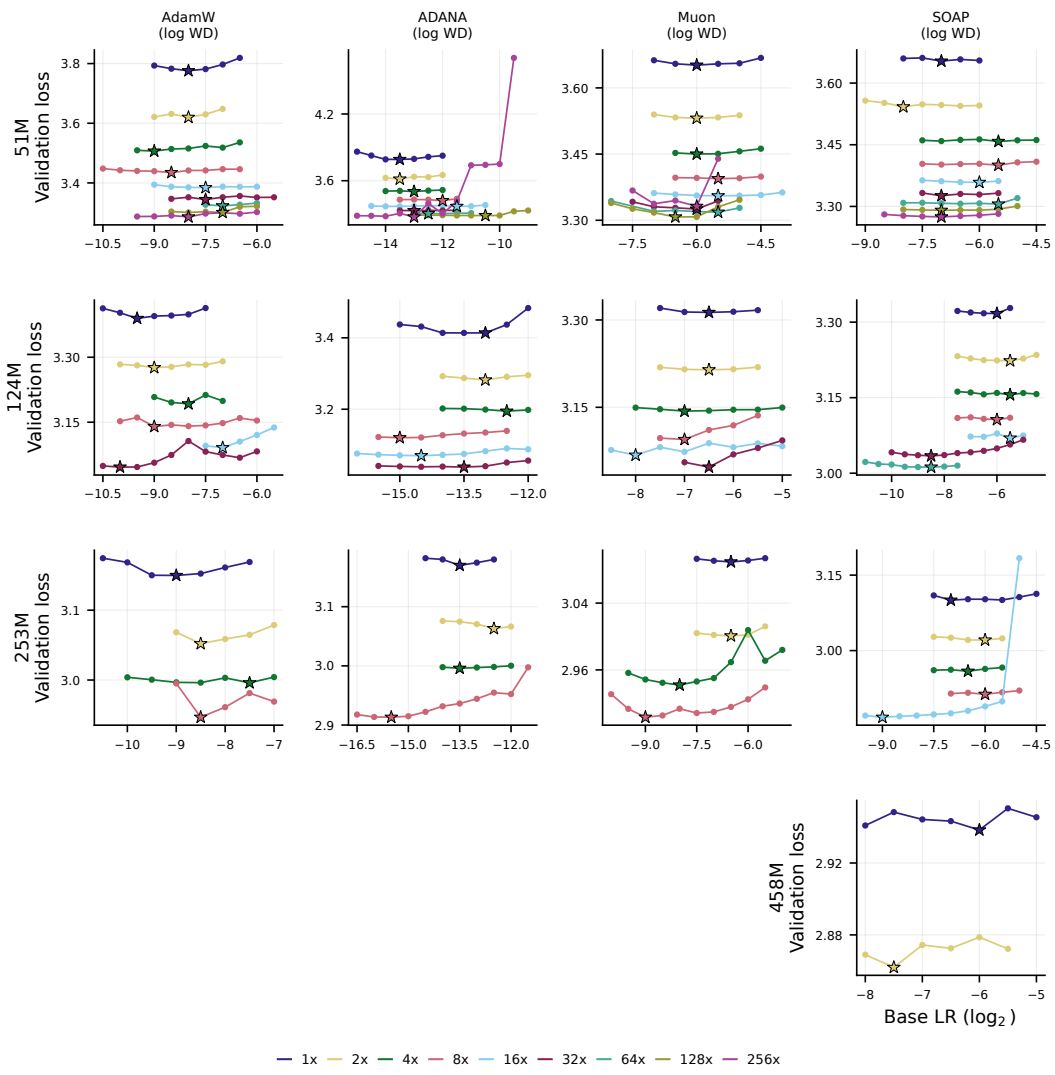

Figure 49: **Learning rate sweeps with log-time weight decay.** As in Figure 48, using the $1\times$-anchored log-time weight decay schedule with $c_{\log}(f) = 2\sqrt{f}$, without momentum cooldown. All 80 sweeps satisfy the 1-mnat interiority criterion.

K.5 FIXED-MEMORY SWEEPS

Figures 50 and 51 report the complete learning rate sweeps underlying the memory-tuning results in Appendix F and Figure 18.

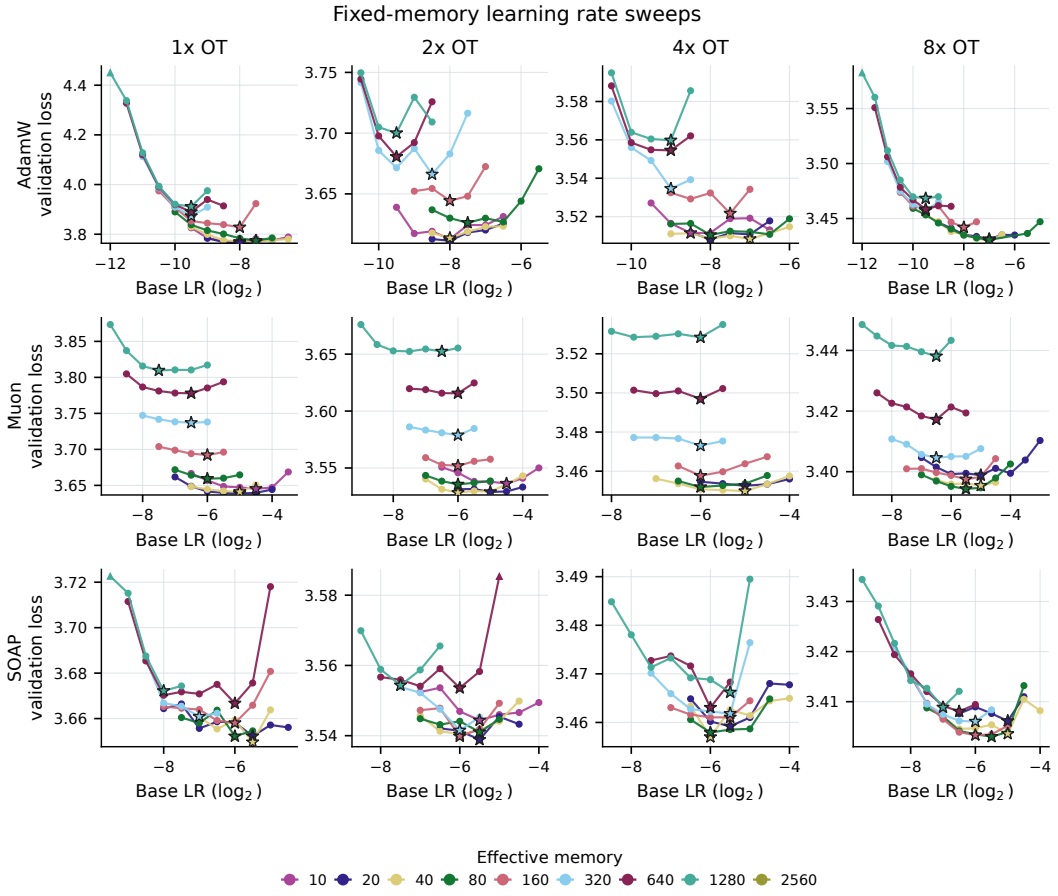

Figure 50: **Learning rate sweeps across fixed memory windows at $1\times$–$8\times$ OT.** We use the 51M model with uniform weight decay. Rows show optimizers, columns show OT factors, and colors identify effective memory $M = 1/(1-\beta)$, where β denotes AdamW/SOAP's second-moment coefficient or Muon's momentum coefficient. All 90 sweeps satisfy the 1-mnat interiority criterion.

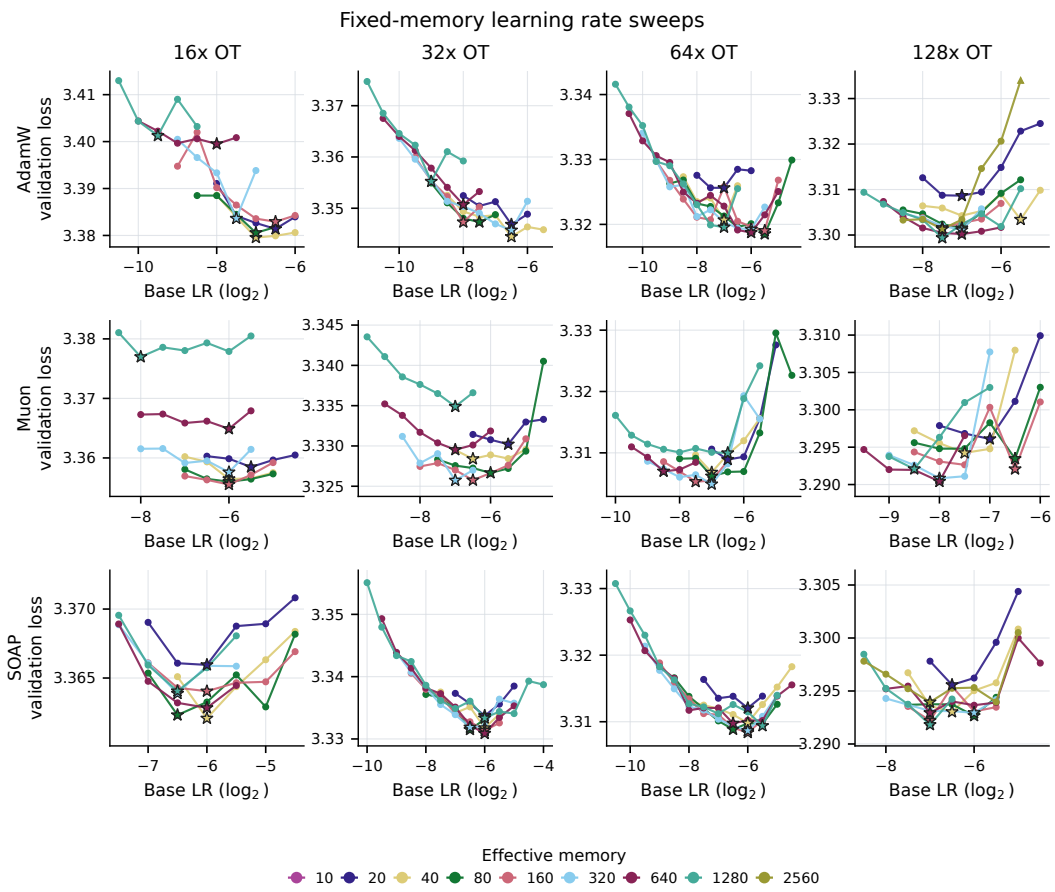

Figure 51: **Learning rate sweeps across fixed memory windows at** $16\times$**–**$128\times$ **OT.** As in Figure 50, at longer training horizons. All 86 sweeps satisfy the 1-mnat interiority criterion.

K.6 Momentum Cooldown Sweeps

Figures 52–53 report the learning rate sweep evidence underlying the momentum-cooldown comparisons in Figure 28.

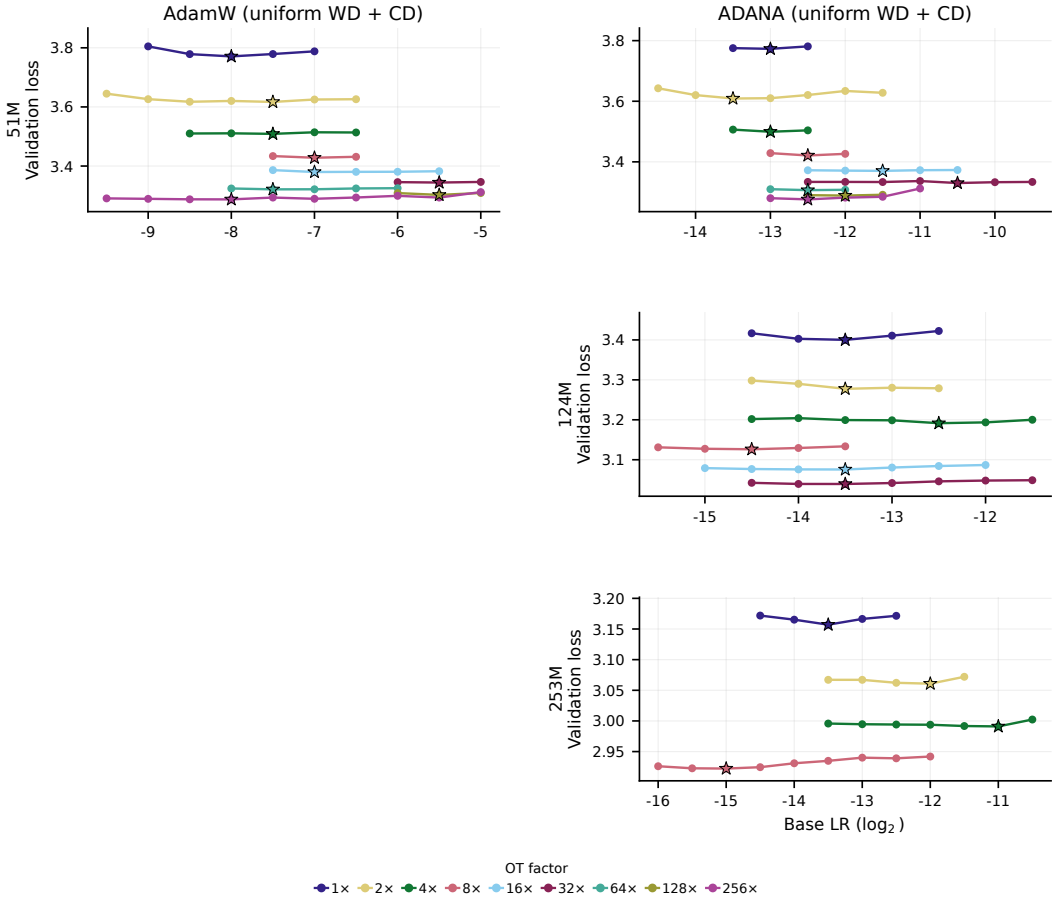

Figure 52: **Learning rate sweeps with uniform weight decay and momentum cooldown.** Columns show AdamW and ADANA, rows show model sizes, and colors identify OT factors. Corresponding sweeps without cooldown appear in Figure 48. All 28 sweeps satisfy the 1-mnat interiority criterion.

Momentum cooldown treatment-selection sweeps. Figure 54 exposes the component-wise learning rate sweeps underlying Table 7.

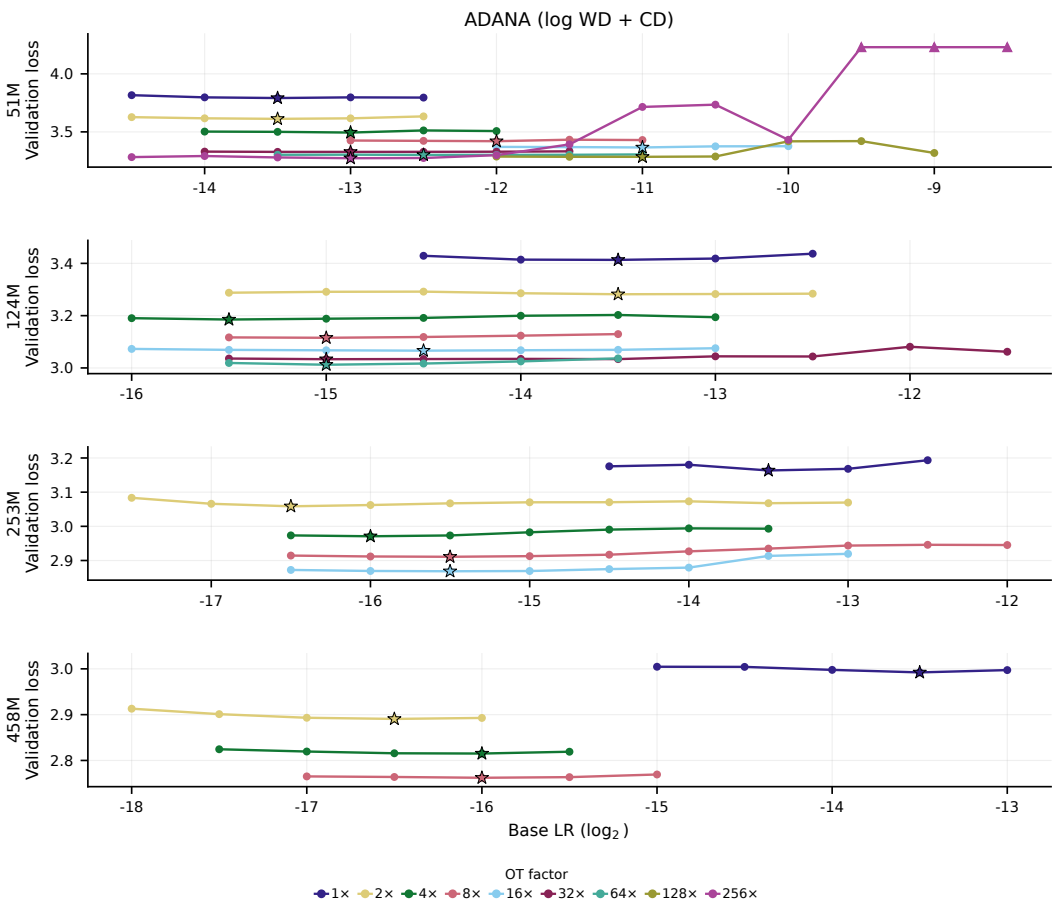

Figure 53: **ADANA learning rate sweeps with log-time weight decay and momentum cooldown.** Rows show model sizes from 51M to 458M parameters; colors identify OT factors. Corresponding sweeps without cooldown appear in Figure 49. All 25 sweeps satisfy the 1-mnat interiority criterion.

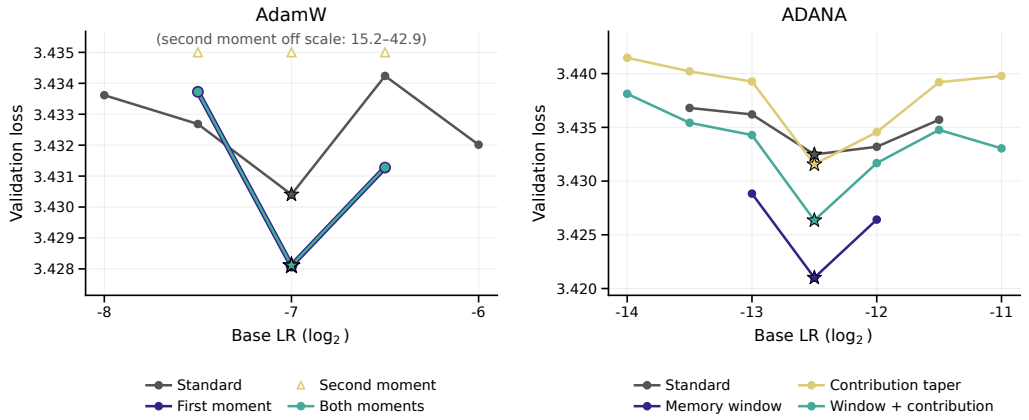

Figure 54: **Learning rate sweeps for cooldown-component selection at** $8\times$ **OT.** Each panel overlays the independently swept component treatments for one optimizer. AdamW first-moment and both-moment cooldown are nearly coincident. The three second-moment-only AdamW observations have losses from 15.2 to 42.9 and are shown as capped upward triangles; they are instability outcomes rather than ordinary open minima. All other displayed selected minima satisfy the 1-mnat learning rate interiority rule.

K.7 BATCH-SIZE SWEEPS

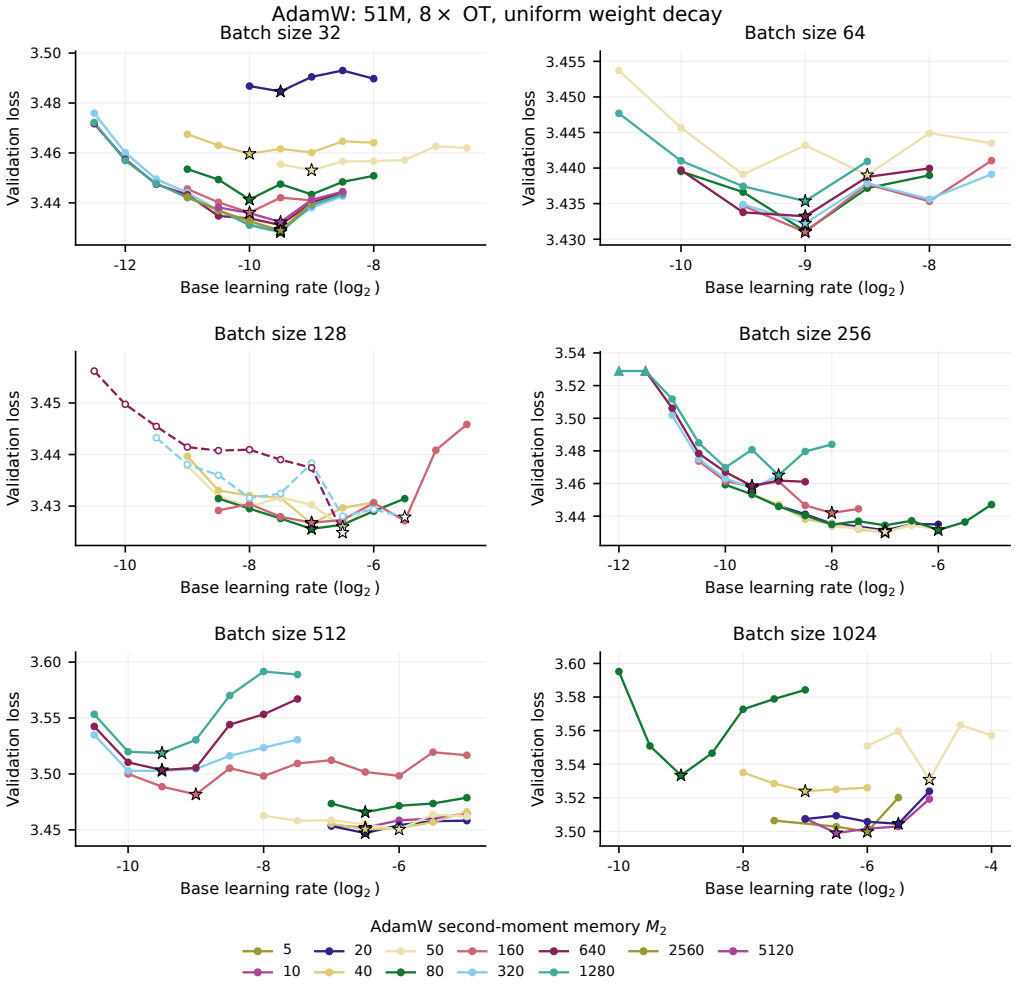

Figure 55: **Learning rate sweeps across AdamW second-moment memory and batch size.** We show 45 sweeps for the 51M, 8×-OT uniform-WD experiment. Panels show batch sizes; colors identify M_2. Stars mark sweep minima, and upward triangles indicate losses above the displayed range. Solid curves with filled markers satisfy the 1-mnat interiority criterion; dashed curves with hollow markers remain boundary-limited. The joint selections in Figure 42 additionally require interiority along the memory axis.

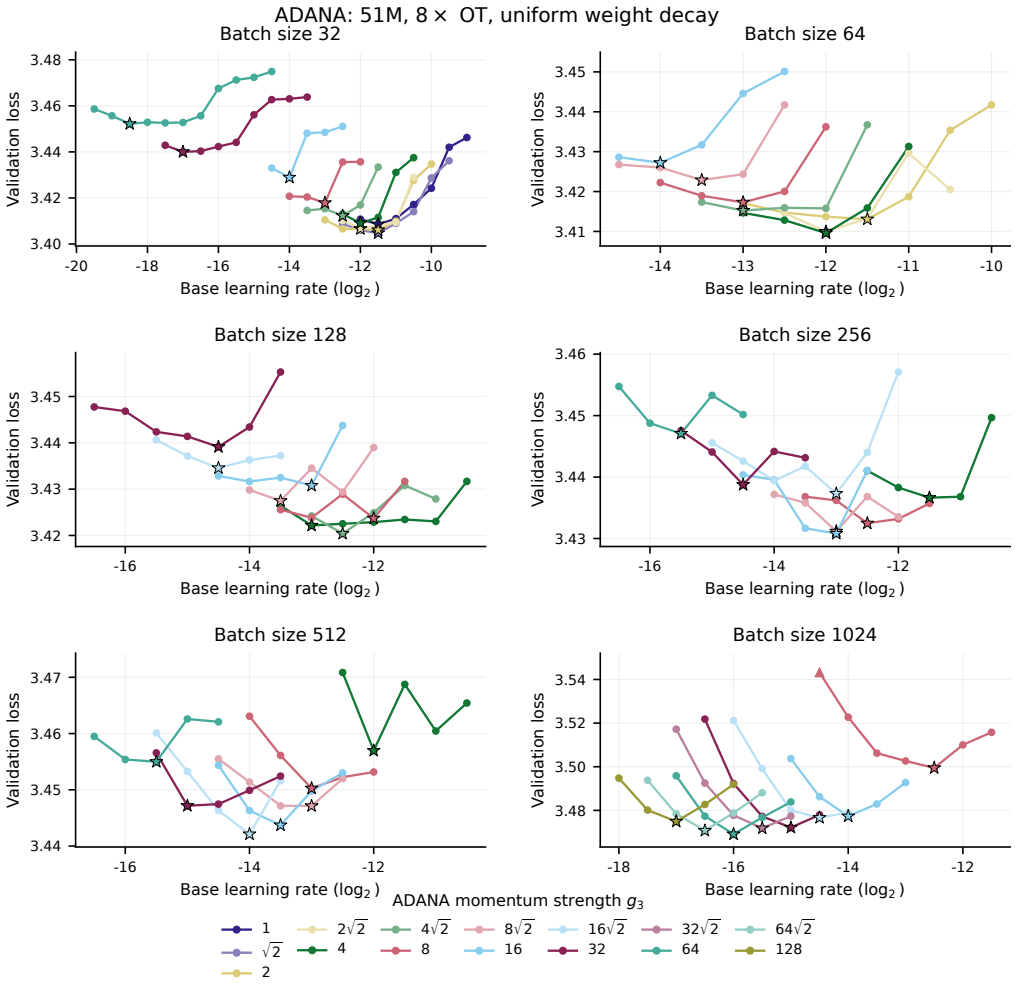

Figure 56: **Learning rate sweeps across ADANA momentum strength and batch size.** We show 46 sweeps for the same experiment as Figure 55; colors identify g_3. All 46 learning rate sweeps satisfy the 1-mnat interiority criterion. Other plotting conventions follow Figure 55.