AERO TRAINING LOOP

TECHNICAL WHITE PAPER

Performance White Paper — 1.36B SwiGLU C4 Pretraining

Apples-to-apples comparison against AdamW at 1×–8× Chinchilla, with Stanford's *Fantastic Pretraining Optimizers* landscape as published reference.

Ralph Gobeli · Charles Black

DevOpt Labs, Inc.

June 2026

Contents

— *Executive Summary (in plain English)*

— *Abstract*

— *About AERO*

1 Methodology — what “speedup” means here

2 Test setup

3 Why AERO is fast — two independent levers

4 Comparison to the published optimizer landscape

5 Test 1 — measured token-efficiency speedups (1 node of 8×H100)

6 Wall-clock results

7 Test 2 — 4-node 32×A100 cluster (comms-bound regime)

8 Summary

9 Seed invariance — the advantage is not a lucky seed

10 Outlook — why this is likely a lower bound

— *References*

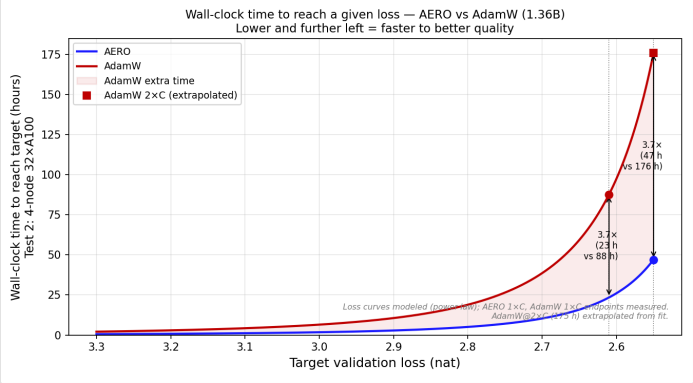
Executive Summary (in plain English)

DevOpt Labs, June 2026

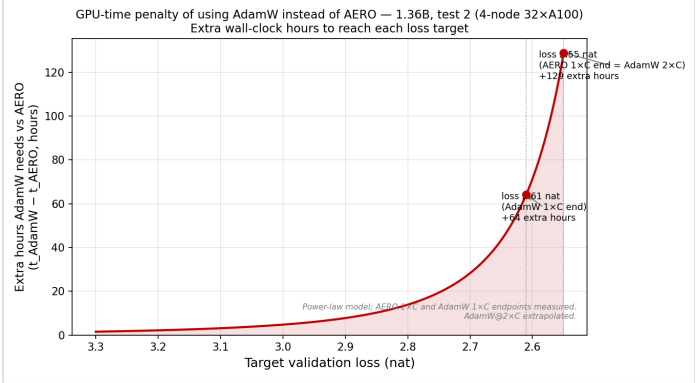
AERO is a drop-in replacement for the current best-practice AI training loop. In a head-to-head test on a 1.36B-parameter model, AERO reached the same quality as a carefully tuned industry-standard baseline (AdamW) using **half the training steps** — and on a production multi-machine cluster, in roughly **a quarter of the wall-clock time**. The practical consequence is simple: **AERO cuts the cost of training a model by at least half.**

The bottom line — at least half the training cost

- **Half the training steps** to reach the same model quality, on *any* hardware — directly measured, not a projection. That alone halves the compute bill.
- **As little as 27% of the wall-clock time** end-to-end on a 4-machine, 32-GPU cluster — 47 hours vs. 175 hours to the same quality (on runs of 2× C or greater).
- **~10× the speed gain** of the best published alternative optimizer over the same baseline.



Hours of cluster time to reach each quality target — AERO (blue) vs AdamW (red). AERO reaches AdamW's best quality in a fraction of the time. *(Figure 4; full detail in §7.)*



The extra hours AdamW must burn to match AERO at each quality target — the cost penalty grows as quality improves. *(Figure 5; full detail in §7.)*

Why it is faster — two effects that stack

AERO is faster for two separate reasons, and they build on each other. First, the method reaches a given quality in about **half as many training steps** — that comes from the underlying math, so it holds on any hardware. Second, when training is spread across many machines, AERO needs **less communication between them on every step**, so each step also finishes faster. Fewer steps, and faster steps: multiplied together, that's how half the steps turns into roughly a **quarter of the total time** on a real cluster.

Why you can trust it

This isn't a fluke or a hopeful projection: the half-the-steps result is **measured directly**, it **holds up when we change the random starting point**, and AERO ran start-to-finish **without instability**. The full rigor — how it compares to other published optimizers, and exactly how speedup is measured — is in the body of the paper.

Abstract

This white paper evaluates AERO, a **hardware-agnostic AI training loop** from DevOpt Labs, Inc., against a well-tuned AdamW baseline on identical 1.36B-parameter SwiGLU pretraining on the very well known C4 web corpus (GPT-2 BPE tokenizer, 50,257-token vocabulary), following the methodology of Stanford's *Fantastic Pretraining Optimizers and Where to Find Them* (Wen et al., 2025).

On a single 8×H100 node (Test 1), AERO reaches a validation loss of **2.55 nat at 1× Chinchilla** (a "nat" is a natural-log unit of loss; lower is better) — the same loss AdamW reaches only at 2× Chinchilla — for a **directly measured 2.0× token-efficiency speedup**. A matched AERO run at 2× Chinchilla (2.49 nat) confirms the advantage holds beyond the 1× C anchor. On a communication-bound 4-node 32×A100 cluster (Test 2) this hardware-independent advantage (Lever 1) compounds with a measured 1.86× per-step throughput advantage (Lever 2) for a combined **3.7× end-to-end wall-clock speedup** (on runs of 2× C or greater). A second random seed on a 130M-parameter proxy reproduces the gap to two decimal places, confirming the result is seed-invariant rather than a lucky draw. We argue the measured 2× is a conservative lower bound on production-typical advantage.

Keywords: training loop, optimizer, token efficiency, scaling laws, distributed training, AdamW, Chinchilla, H100, A100.

About AERO

AERO is a **hardware-agnostic AI training loop** from DevOpt Labs — a drop-in replacement for the standard forward / backward / optimizer step, not a single optimizer substitution. The same code path runs unchanged on H100 and A100, single-node and multi-node, fast intra-node fabrics (NVLink/NVSwitch, SXM) and bandwidth-constrained inter-node links. It produces two distinct advantages: **Lever 1** (hardware-independent token efficiency — fewer training tokens to reach a target loss) and **Lever 2** (hardware-dependent per-step throughput — less inter-device comms per step). This paper measures both on the same model and data: Test 1 (1-node 8×H100) isolates Lever 1 (2.0×); Test 2 (4-node 32×A100, comms-bound) compounds it with Lever 2 (1.86×) for a combined 3.7× end-to-end wall-clock speedup on runs of 2× C or greater.

Headline result — speedup is measured, not projected

Four matched anchors on identical Test 1 hardware (1.36B SwiGLU, packed C4, 1 node of 8×H100 80GB GPUs): **AdamW@1× C =** 2.61, **AdamW@2× C =** 2.55, **AERO@1× C =** 2.55, **AERO@2× C =** 2.49. The two end-of-run values coincide at 2.55 nat, so the token-efficiency speedup at AERO's loss is **2.0× by direct measurement** — the literal ratio of training budgets, no scaling-law fit required. On the comms-bound Test 2 (4-node 32×A100 cluster) that 2.0× (Lever 1) compounds with a measured 1.86× per-step throughput advantage (Lever 2) into a combined wall-clock speedup of 3.7× (on runs of 2× C or greater) — AERO takes 27% of the AdamW wall-clock time to reach the same val/loss target on that hardware. For context, Stanford (Wen et al., 2025) reports ~1.0–1.1× token-efficiency speedup for the best alternative optimizers (Muon, Soap, NAdamW) at their largest scale (1.2B) — i.e. a ~10% speed gain over AdamW. AERO's measured 2.0× is a 100% speed gain — 10× the improvement that the best published optimizer delivers over AdamW, under the same methodology.

Test 1 — 1 node of 8×H100 80GB GPUs · same model, dataset, optimizer hyperparameters across both rows · token-efficiency speedup is an optimizer property and is identical between tests

2.61	2.55	**2.0×**	**50%**
AdamW measured @ 1× C	AERO measured @ 1× C	tokens-to-target speedup	**of AdamW steps needed**
baseline val/loss	*0.06 nat below baseline*	*directly measured (AdamW @ 2× C = 2.55)*	*AERO finishes in half the training steps to reach the same loss*

Test 2 — 4-node 32×A100 cluster (comms-bound regime) · same model/dataset as Test 1, run on production-typical multi-node hardware (45 GB/s inter-node link) · Lever 1 (steps) compounds with Lever 2 (per-step throughput) on this topology — full breakdown in §7

3.65 days	1.96 days	**3.7×**	**27%**
AdamW @ 1× C wall-clock	AERO @ 1× C wall-clock	combined wall-clock speedup	**of AdamW wall-clock needed**
87.6 h, 760 ms/step on 32×A100	*47.0 h, 408 ms/step on 32×A100*	*AERO@1×C vs AdamW@2×C at matched val/loss; Lever 1 (2.0×) × Lever 2 (1.86×)*	*AERO finishes the same loss target in 27% the time*

All eight KPIs are measured values on completed runs. Test 1 reports token efficiency only (single 8×H100 node intra-node fabric is fast enough that AERO and AdamW sit at near-parity on per-step throughput, so wall-clock is not compared at Test 1). Test 2 compounds the same hardware-independent token-efficiency advantage with the per-step throughput advantage in the comms-bound regime. The highlighted tiles carry the headline ×-speedups; the plain tile beside each re-expresses it as the *fraction of AdamW resource needed*. See §5 for the step-efficiency framing and §7 for the full Lever 1 × Lever 2 decomposition.

1. Methodology — what "speedup" means here

Key concepts (plain English)

The training loop. Training a large language model repeats three operations millions of times: run the model forward, measure how wrong it was, then update its weights with an optimizer such as AdamW. AERO replaces this entire loop — not just the optimizer — with a single integrated system that runs unchanged on any GPU.

Validation loss. The goal of training is to make the model's internal representation match real-world data as closely as possible. The number that measures this is *validation loss*, measured in *nats* (natural-log units of loss) — lower is better. Frontier labs spend enormous sums to push it down by small amounts: a 0.05 nat improvement can separate a competitive release from an uncompetitive one, and the gap between billion-dollar training runs.

The Chinchilla compute multiple. The industry yardstick for "how much training" is the Chinchilla multiple, from DeepMind's 2022 scaling-law paper: one Chinchilla (1× C) is the compute-optimal budget of roughly 20 training tokens per model parameter — about 27 billion tokens (~415,000 steps) for the 1.36B model tested here. Production models are commonly trained to around 8× Chinchilla (informally "epochs"), because each additional multiple buys a further measurable drop in validation loss before diminishing returns set in. Faster training at a given multiple converts directly into either lower cost at the same quality or higher quality at the same cost — which is where AERO's advantage lives.

> **Always compare at the end of a whole Chinchilla**
>
> Speedups measured at *less than* one full Chinchilla are not meaningful: before a run reaches the end of a Chinchilla budget the two loss curves can take different shapes en route to the target, so a mid-run snapshot can flatter either optimizer. Every comparison in this paper is taken at the **end of a whole Chinchilla multiple** (1×, 2×, …), where the curves have settled and the numbers are apples-to-apples. Figure 3 (§4) makes this concrete: the shaded band marks where AERO's 2× advantage first opens up, but rather than reading the result off that early crossing we carry both runs all the way out to their full 1× C and 2× C endpoints — so the headline 2.0× is measured at the end of a complete Chinchilla budget, where it is robust, not at the point where the advantage merely begins.

How speedup is measured

This paper follows the convention established by the Stanford *Fantastic Pretraining Optimizers and Where to Find Them* paper (Wen et al., 2025). All speedup numbers measure **token efficiency**: for a given target validation loss L, the speedup of training loop X over baseline AdamW is $D_{AdamW}(L) / D_X(L)$, where $D(L)$ is the number of training tokens (or equivalently steps at fixed batch size) needed to first reach loss L. Per-step token throughput is held constant across candidate and baseline.

Both AERO and AdamW are run at the same bf16 mixed precision on the same data on the Test 1 hardware (1 node of 8×H100 80GB); Test 2 (§7) uses the same precision and code path on a 4-node 32×A100 cluster. Test 1 reports token efficiency only — on a single 8×H100 node, intra-node NVLink / NVSwitch (~900 GB/s all-reduce bandwidth) is fast enough that AERO and AdamW sit at near-parity on per-step throughput; wall-clock differences only emerge in the communication-bound Test 2 regime (§7), where Lever 2 becomes a first-order cost.

2. Test setup

Table 1. Test setup. Same model, data, precision, and code path across all four measured anchors and across both Test 1 (8×H100) and Test 2 (32×A100, §7) hardware regimes.

Field	Value
Model	1.36B-parameter SwiGLU transformer (untied embeddings)
Tokenizer / data	GPT-2 BPE (50,257 vocab); packed C4-EN, sequence length 2048, effective batch 32 (per-GPU 4 × 8 GPUs × 1 GA)
Hardware (Test 1)	1 node of 8×H100 80GB GPUs, intra-node NVLink / NVSwitch (~900 GB/s all-reduce bandwidth)
Precision	bf16 mixed precision (torch.autocast, dtype bfloat16) for the forward / backward compute, with fp32 master weights, fp32 gradients, fp32 optimizer state, and cross-entropy loss in fp32; no fp16 and no gradient scaling — identical for AERO and AdamW
LR schedule	Cosine decay, peak LR = 1.9e−3, warmup 8,000 steps
Total pretraining horizon	1× Chinchilla ≈ 27 B tokens ≈ 415,000 steps
AdamW baseline	Two matched-tuning anchors (LR / WD / β's re-tuned per Chinchilla budget): **2.61 nat at 1× C** and **2.55 nat at 2× C**. Joint fit (α held at the Stanford-consistent 0.32): $L_{AdamW}(C) = 2.31 + 0.30 \cdot C^{-0.32}$. E = 2.31 is an *extrapolated lower bound, not a measured asymptote*; a third AdamW anchor at 4× C is required before the tail of the fit can be trusted.

AdamW baseline is well-tuned — Stanford-trap compliance (summary)

Stanford's headline finding is that a single peak-LR re-tune on the GPT-3 recipe can collapse a claimed 2× optimizer speedup to ~1.1×, so the speedup reported here depends on a demonstrably well-tuned AdamW. Our AdamW configuration sits at the consensus midpoint of the GPT-3 / Cerebras-GPT / MPT-1B / Stanford / nanoGPT recipes (β_1=0.9, β_2=0.95, WD=0.1, cosine + linear warmup, grad-clip 1.0); the single most-leveraged hyperparameter, peak LR = 1.9e−3, is within 5% of Stanford's coordinate-descent optimum at 1.2B / 1× C (2.0e−3). Our two-anchor scaling fit gives **α ≈ 0.32**, matching Stanford's 4-point measured AdamW slope at 1.2B (0.30–0.32) — the "well-tuned AdamW" fingerprint, not the under-tuned one. AERO and AdamW are each tuned to their own near-optimal hyperparameters per Stanford's per-optimizer-rigorous-tuning pillar, both AdamW anchors are independent matched-tuning runs (not single-config transfer across budgets), and all headline values are end-of-training measurements. No published pure-C4 GPT-2-tokenizer 1.3B reference exists in directly comparable form, so absolute-loss benchmarking against the open literature is not possible; the defense rests on hyperparameter parity, the Stanford-matching scaling slope, and compliance with Stanford's three methodological pillars (matched-tuning per budget, end-of-training measurement, per-optimizer rigorous tuning).

3. Why AERO is fast — two independent levers

AERO is a tightly-coupled training stack — CUDA kernel, auxiliary-state representation, periodic maintenance, and update rule designed as one system. Its advantage over an AdamW-based training loop is the product of two compounding effects.

Lever 1 — step efficiency. AERO reaches a target validation loss in fewer optimization steps than AdamW under matched fair-test conditions. This is the lever Stanford's "speedup" metric measures directly. At 1× Chinchilla on this 1.36B model, AERO reaches **2.55 nat**; AdamW reaches 2.55 nat only at 2× Chinchilla, so the token-efficiency speedup is **2.0× by direct measurement** (literal ratio of training budgets — no scaling-law fit involved). Single-seed noise on the AdamW anchor is roughly ±0.02 nat, which propagates to a speedup band of **1.93–2.07×**. Lever 1 is algorithmic — it is the same number on any hardware that runs the same code path.

Lever 2 — per-step throughput. AERO reduces the amount of optimizer / update traffic that must move across devices, which directly reduces per-step wall-clock in configurations where inter-device communication is the bottleneck. On a single 8×H100 node the intra-node fabric (NVLink / NVSwitch, ~900 GB/s) is fast enough that AERO and AdamW sit at near-parity on per-step throughput, so Lever 2 is small there and Test 1 does not report it. In the production-typical multi-node regime (Test 2, 4-node 32×A100, 45 GB/s inter-node link), the inter-node all-reduce becomes the dominant per-step cost for AdamW and AERO runs at **1.86× AdamW's per-step rate**, which compounds with the same Lever 1 to yield combined wall-clock speedup of **3.7×** on runs of 2× C or greater.

4. Comparison to the published optimizer landscape

This section places our in-house result alongside the published optimizer landscape, split across two charts: **Figure 1a** shows the relevant results from the Stanford *Fantastic Pretraining Optimizers* paper (their 1.2B models on C4/EN validation), and **Figure 1b** shows our in-house 1.36B C4/EN-only runs, AERO vs a well-tuned AdamW. In Figure 1b, AERO at 1× and 2× C are measured anchors (2.55 and 2.49 nat); 4×/8× C are projections from the fit $L_{aero}(C) = 2.31 + 0.24 \cdot C^{-0.415}$ that pins to the measured anchors and shares AdamW's extrapolated lower bound (E = 2.31). At each budget AERO reaches roughly the quality AdamW attains only at twice the compute — at 1× C this is a direct measurement (AERO@1× = AdamW@2× = 2.55), and the measured 2× point (AERO 2.49 vs AdamW@4× 2.50) confirms the same advantage — so we report a conservative ~2.0× speedup throughout.

Figure 1 — C4/EN validation loss vs Chinchilla ratio: published reference (1a) and in-house result (1b)

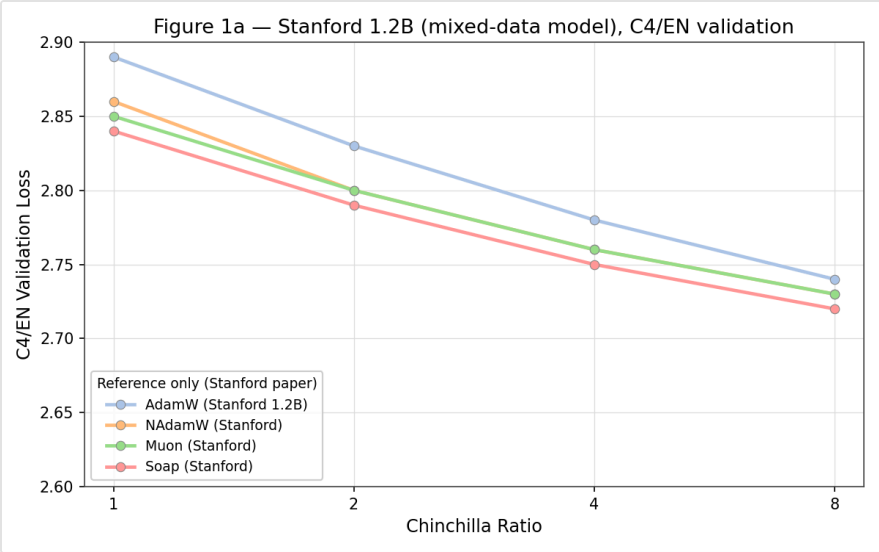


Figure 1a. *Reference only.* The published Stanford optimizer landscape — 1.2B models (trained on additional, more complex data) evaluated on C4/EN validation loss across Chinchilla ratios 1×–8×, for AdamW, NAdamW, Muon, and Soap. Among the alternatives, the best (Soap/Muon/NAdamW) sit only ~1.0–1.1× ahead of well-tuned AdamW at this scale.

About Figures 1a and 1b — why the in-house loss is lower

For illustration purposes only, the Stanford paper results are shown in Figure 1a and our in-house testing results in Figure 1b. Both use the same validation dataset (C4/EN); however, the Stanford paper uses a model trained on additional, more complex data, whereas our runs use a C4/EN-only model. The lower in-house AdamW baseline reflects this simpler training data (and the specific model/tokenizer) — the AdamW optimizer itself is unmodified.

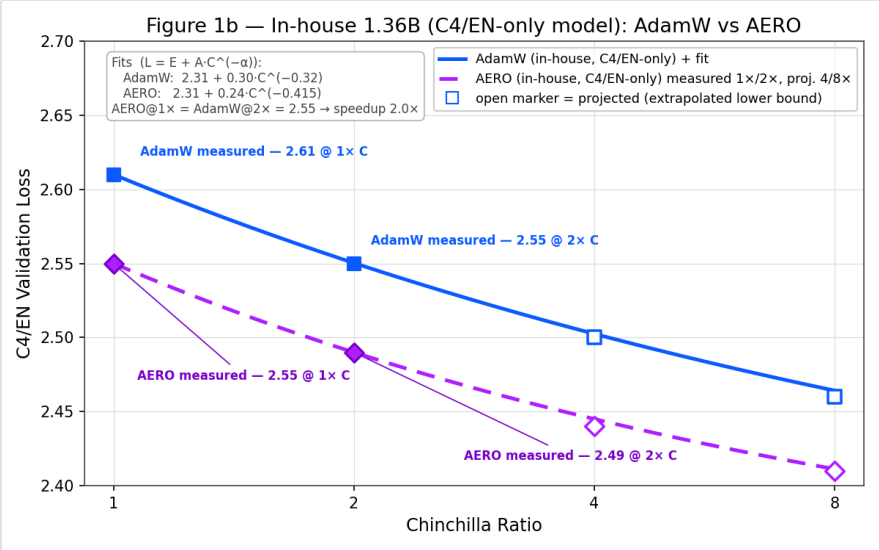


Figure 1b. Our in-house result — 1.36B C4/EN-only model, AERO vs a well-tuned AdamW. **AdamW** is measured at two anchors (filled squares: 2.61 at 1× C, 2.55 at 2× C); the solid line is the fit $L_{AdamW}(C) = 2.31 + 0.30 \cdot C^{-0.32}$ through both anchors (E = 2.31 is an extrapolated lower bound, not a measured asymptote); open markers are 4×/8× C projections. **AERO** is measured at 1× and 2× C (filled purple diamonds: 2.55 and 2.49); AERO@1× lands at *exactly* AdamW's measured 2× C anchor (2.55), so the speedup is **2.0× by direct measurement**. The dashed purple line is the AERO fit ($L_{aero} = 2.31 + 0.24 \cdot C^{-0.415}$); open diamonds are the 4×/8× C projections (2.44, 2.41). Across budgets AERO reaches at C what AdamW needs ~2C to match, holding the ~2.0× advantage. Our AdamW is tuned to the well-tuned level (see §2 callout), not the under-tuned "Stanford trap" baseline.

Where Figure 1b's measured anchors come from — the underlying training curves

The four measured anchors plotted in Figure 1b (AdamW@1× C = 2.61, AdamW@2× C = 2.55, AERO@1× C = 2.55, AERO@2× C = 2.49) are not isolated end-of-run numbers — they are the terminal values of full validation-loss trajectories over the entire 415K-step (1× C) and 830K-step (2× C) horizons. Figures 2 and 3 show those raw curves so the reader can trace each Figure 1b data point back to the run that produced it.

How to read Figure 2 — same training budget, better model

What it shows: AERO and AdamW run head-to-head on an *identical* 1× Chinchilla budget (~415K steps), with the same model, data, precision, and code path — only the optimizer differs. The vertical axis is model quality (validation loss, lower is better); the horizontal axis is training progress (steps).
Takeaway: AERO's curve stays below AdamW's for the entire run and finishes lower — 2.55 vs 2.61 nat. Given the same amount of training, AERO produces the better model.

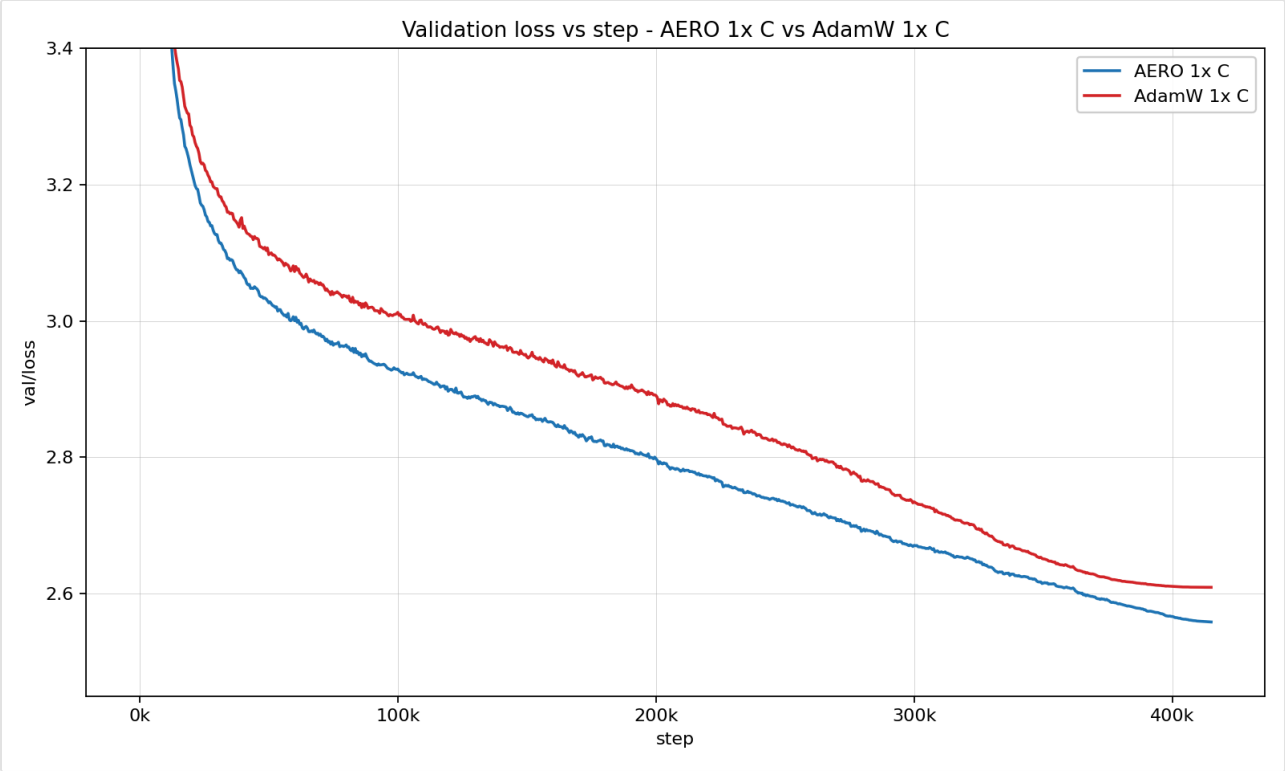


Figure 2. Same-budget comparison at 1× Chinchilla — AERO 1× C (blue) vs AdamW 1× C (red), validation loss across the full ~415K-step horizon on Test 1 hardware. Both runs use the identical model, data, precision, and code path; only the optimizer differs. AERO sits strictly below AdamW for the entire run and finishes at **2.55 nat** vs AdamW's **2.61 nat**. These two end-of-run values are the **AERO@1× C** and **AdamW@1× C** anchors in Figure 1b.

How to read Figure 3 — same quality, half the tokens

What it shows: one AERO run and one AdamW run overlaid, deliberately on *different* budgets — AERO at 1× Chinchilla (~415K steps) and AdamW at 2× Chinchilla (~830K steps) — to compare how much training each optimizer needs to reach the same validation loss. The vertical axis is model quality (validation loss, lower is better); the horizontal axis is how much training (steps / tokens) it took. Although the two budgets differ by design, both runs are judged on the same yardstick — final validation loss — so the comparison is consistent.
Takeaway: both optimizers reach the same loss, but AERO (blue) gets there in about half the horizontal distance of AdamW (green). The dashed line and shaded band mark where that 2× gap opens up, and it holds through AERO's finish. Same quality, roughly half the training steps.

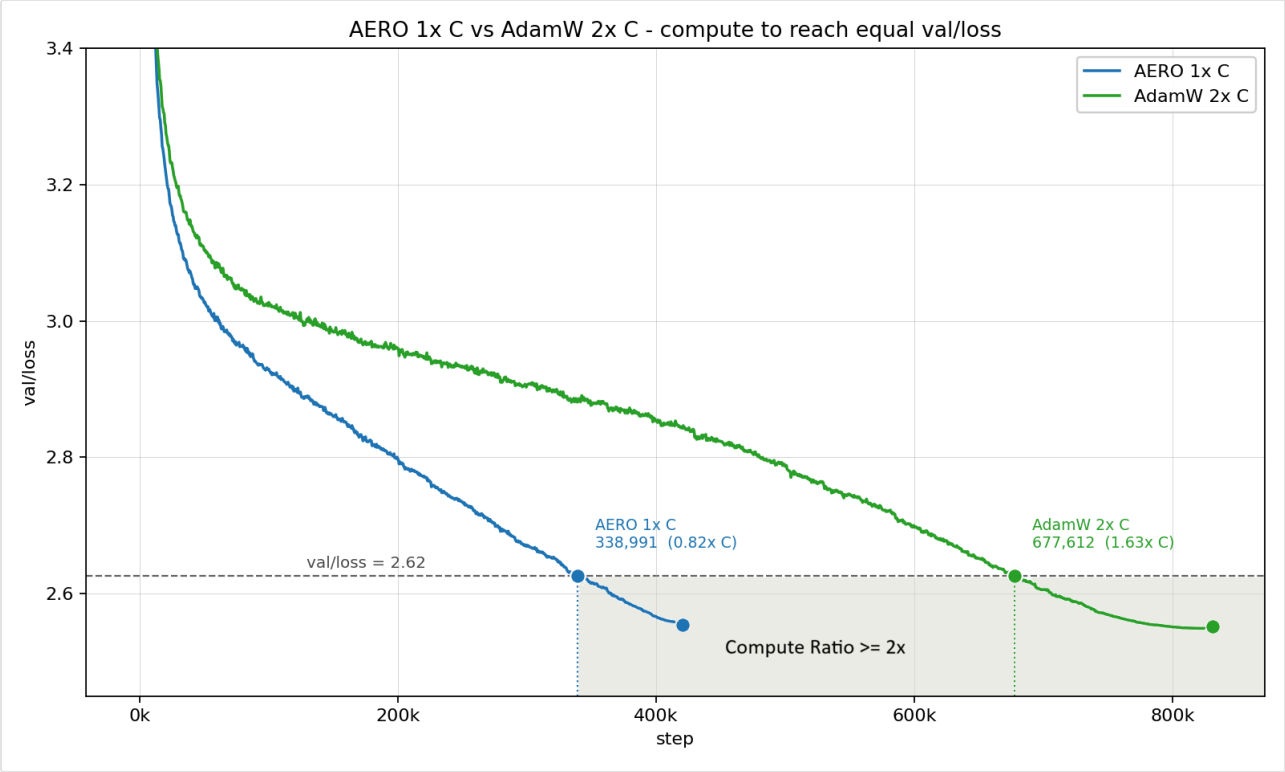


Figure 3. Iso-loss compute comparison — AERO 1× C (blue) vs AdamW 2× C (green). The dashed horizontal line (2.62 nat) marks the loss level at which AERO's 2× step advantage over AdamW first opens up: AERO reaches it at ~339K steps (0.82× C) while the AdamW 2× C run needs ~678K steps (1.63× C) to reach the *same* loss — exactly twice as many steps. That ratio then holds through AERO's 2.55-nat finish, where AdamW requires the full 2× C budget to match. The measured compute ratio is **2.0×** — this is the direct-measurement basis for the headline 2.0× token-efficiency speedup, i.e. the coincident **AERO@1× C = AdamW@2× C = 2.55** point in Figure 1b.

5. Test 1 — measured token-efficiency speedups (1 node of 8×H100)

2.61	2.55	2.0×	50%
AdamW measured @ 1× C	AERO measured @ 1× C	tokens-to-target speedup	**of AdamW steps needed**
baseline val/loss	*0.06 nat below baseline*	*directly measured (AdamW @ 2× C = 2.55)*	*AERO finishes in half the training steps to reach the same loss*

Table 2 lists the four matched-tuning anchor runs that produce every measured number in this paper. Tables 3–4 then convert those anchors into the headline token-efficiency speedup and the per-budget loss values out to 8× C (measured at 1× and 2× C, projected beyond) under a conservative same-asymptote reading that holds the ~2.0× advantage.

Table 2. The four matched-tuning anchor runs that produce every measured speedup in this paper. All four are on identical Test 1 hardware (1 node of 8×H100 80GB GPUs), same model, same packed C4 data, same precision, same code path — only the optimizer and Chinchilla budget differ.

Run	Budget	Final val/loss (nat)	Role
AdamW @ 1× C	27 B tokens (~415K steps)	2.61	AdamW baseline at 1× C
AdamW @ 2× C	54 B tokens (~830K steps)	2.55	Second AdamW anchor — coincides exactly with AERO@1× C, giving the fit-free 2.0× speedup
AERO @ 1× C	27 B tokens (~415K steps)	2.55	Canonical AERO 1× C anchor
AERO @ 2× C	54 B tokens (~830K steps)	2.49	AERO 2× C anchor — confirms the ~2.0× advantage holds beyond 1× C

Table 3. Two valid speedup framings of the same anchor set. Both ratios are *fully measured* — no scaling-law fit is involved.

Anchor loss	What reaches it & when	Compute ratio	Convention
2.61 nat (AdamW's 1× C final)	AdamW reaches 2.61 at 1.00× C; AERO clears 2.61 mid-cosine at ~0.71× C (still descending past this point).	**1.41×**	Conservative early-stop reading
2.55 nat (AERO's 1× C final)	AERO reaches 2.55 at 1.00× C; AdamW reaches 2.55 at 2.0× C exactly (measured) — the two coincide, so the ratio is the literal ratio of training budgets.	**2.0×**	Stanford-paper convention — the headline number, by direct measurement

Table 4. Per-budget val/loss and token-efficiency speedup. AdamW@1×/2× C is measured; AdamW@4×/8× C is extrapolated through both anchors via the fitted law $L(C) = 2.31 + 0.30 \cdot C^{-0.32}$. AERO@1×/2× C is measured; 4×/8× C is projected via the same-asymptote fit ($L_{aero}(C) = 2.31 + 0.24 \cdot C^{-0.415}$, pinned to the measured 1×/2× C anchors). Speedup is the ratio of Chinchilla budgets at matched validation loss: AERO at budget C reaches what AdamW reaches only at 2C — directly measured at 1× C (AERO@1× = AdamW@2× = 2.55) and confirmed at 2× C — so the conservative figure is ~2.0× throughout.

Chinchilla	tokens (B)	AERO val/loss	AdamW val/loss	D_{AdamW} equivalent	speedup
1×	27	2.55	2.61	~54 B (2.0× C)	2.0×
2×	54	2.49	2.55	~109 B (4× C)	~2.0×
4×	109	~2.44	~2.50	~218 B (8× C)	~2.0×
8×	218	~2.41	~2.46	~436 B (16× C)	~2.0×

AdamW at 1×/2× C is measured; AdamW at 4×/8× C is computed from the fitted law $L_{AdamW}(C) = 2.31 + 0.30 \cdot C^{-0.32}$ through both anchors (E = 2.31 is an extrapolated lower bound, not a measured asymptote). AERO at 1× and 2× C is measured (2.55, 2.49); AERO at 4×/8× C is projected from the fit $L_{aero}(C) = 2.31 + 0.24 \cdot C^{-0.415}$, which shares AdamW's extrapolated lower bound (E = 2.31). At each budget AERO reaches roughly the quality AdamW attains only at twice the compute (AERO@1× = AdamW@2× = 2.55 by direct measurement; AERO@2× = 2.49 vs AdamW@4× = 2.50), so we report a conservative ~2.0× token-efficiency speedup throughout.

6. Wall-clock results — deferred to §7

Wall-clock for Test 1 (1 node of 8×H100) is not reported — intra-node NVLink / NVSwitch is fast enough that AERO and AdamW sit at near-parity on per-step throughput and Lever 2 is small in absolute terms there. See §7 for the Test 2 4-node 32×A100 wall-clock measurement, where Lever 1 (2.0×) and Lever 2 (1.86×) compound to a combined **3.7×** end-to-end wall-clock speedup on runs of 2× C or greater.

7. Test 2 — 4-node 32×A100 cluster (comms-bound regime)

Test 2 reruns the same model, data, precision, and code path on a production-typical 4-node 32×A100 cluster where the inter-node interconnect (45 GB/s) is more than an order of magnitude slower than intra-node SXM (600 GB/s) and per-step time is communication-bound for AdamW. This is the regime where Lever 2 becomes a first-order cost and compounds with the hardware-independent Lever 1 from Test 1.

Measured per-step time: AdamW 760 ms/step, AERO 408 ms/step — a per-step ratio of **1.86×**. For a 1.36B model in bf16, AdamW's full inter-node all-reduce dominates per-step cost on the 45 GB/s link; AERO's update geometry sends less over that link and overlaps more compute with the remaining inter-node phase.

3.65 days
AdamW @ 1× C wall-clock
87.6 h on 32×A100, loss 2.61

1.96 days
AERO @ 1× C wall-clock
47.0 h on 32×A100, loss 2.55

3.7×
combined wall-clock speedup
AERO@1×C (47 h) vs AdamW@2×C (~175 h) at matched val/loss; Lever 1 (2.0×) × Lever 2 (1.86×)

27%
of AdamW wall-clock needed
AERO finishes the same loss target in 27% the time

Wall-clock derivation: AdamW @ 1× C = 415K steps × 760 ms = 87.6 h; AERO @ 1× C = 415K × 408 ms = 47.0 h. AdamW would need 2× C (830K steps × 760 ms = 175.2 h) to match AERO's 2.55 nat val/loss. Matched-loss wall-clock ratio = 47.0 / 175.2 = **27%**. Lever 1 is algorithmic and identical to Test 1; Lever 2 grows monotonically as the inter:intra bandwidth ratio worsens, so frontier-class deployments with slower aggregate networking will see more amplification than the 1.86× reported here. One caveat: the per-step ratio (1.86×) is a single measured timing rather than an average over many runs with error bars, so the combined 3.7× is best read as a well-grounded estimate rather than a precision figure. The hardware-independent Lever 1 (2.0×), by contrast, is a direct end-of-budget measurement.

Wall-clock to a loss target — the Test 2 advantage, visualized

How to read Figures 4 and 5 — the speedup in cluster hours

What they show: the same Test 2 result recast in the terms a deployment team plans around. Figure 4 plots the hours of 32×A100 cluster time needed to reach each quality target, AERO vs AdamW. Figure 5 plots the *extra* hours AdamW must spend to match AERO at each target. In both, lower validation loss (further left) means a better model.

Takeaway: AERO reaches AdamW's best quality in a fraction of the wall-clock time — 47 h vs ~175 h, about **3.7×** faster — and the time penalty for staying on AdamW grows the higher you push quality (+64 h at AdamW's own 1× C loss, +129 h at AERO's 1× C loss). Higher quality costs far fewer hours — and therefore far less — on AERO.

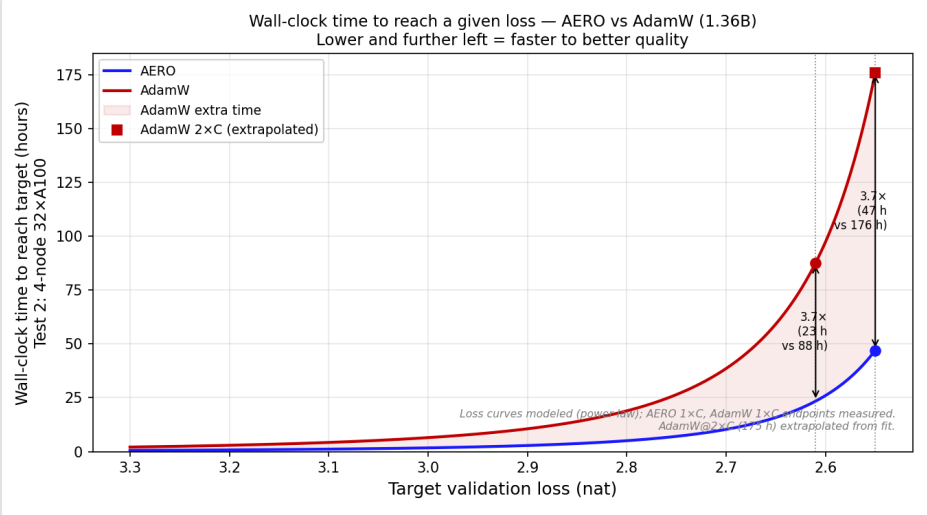


Figure 4. Wall-clock hours to reach each target validation loss on Test 2 (4-node 32×A100), AERO (blue) vs AdamW (red); reading right-to-left is the direction of better quality. AERO's curve sits far below AdamW's and the gap widens as the loss target tightens. At AERO's 1× C endpoint (2.55 nat) AERO needs **47 h** while AdamW needs the full 2× C budget — ~175 h (extrapolated) — a **3.7×** wall-clock gap. Loss curves are power-law models pinned to measured AERO 1× C and AdamW 1× C endpoints; the AdamW 2× C point (175 h) is extrapolated from the fit.

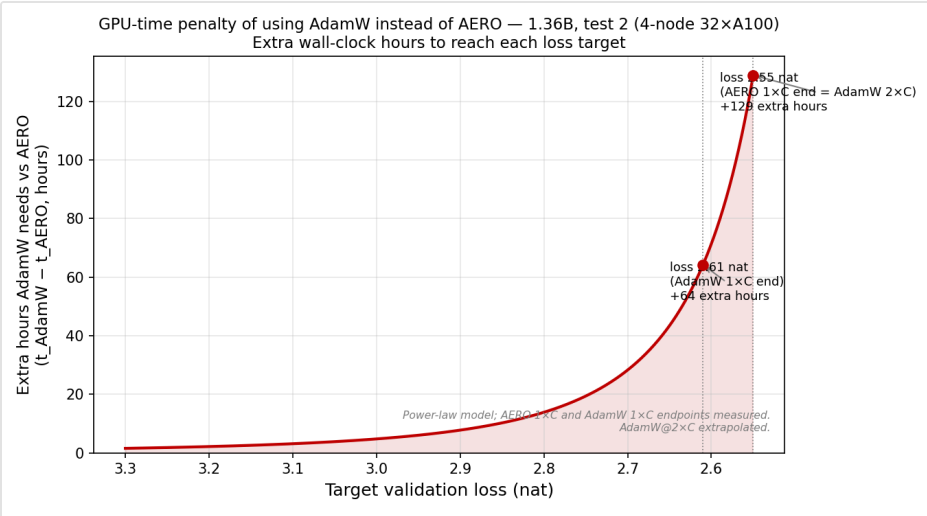


Figure 5. The same data expressed as the *GPU-time penalty* using AdamW vs. AERO — the extra wall-clock hours AdamW needs to reach each loss target on Test 2. The penalty grows steeply as quality improves: **+64 h** at AdamW's own 1× C loss (2.61 nat) and **+129 h** at AERO's 1× C loss (2.55 nat, where AERO's endpoint equals AdamW's 2× C). Endpoints measured; AdamW@2× C extrapolated from the power-law fit. These intermediate-target penalties are read off the fitted (smooth power-law) curves, so the implied AERO crossing for a target such as 2.61 nat is slightly earlier than the deliberately conservative early-stop reading in Table 3 (1.41×); the headline 2.0× at 2.55 nat is a direct end-of-budget measurement and is unaffected.

8. Summary

- **Test 1 (1 node of 8×H100) — token efficiency, directly measured:** AERO reaches 2.55 nat at 1× C; AdamW reaches the same 2.55 nat only at 2× C, so the speedup is the literal ratio of training budgets — **2.0× by direct measurement**, no scaling-law fit involved. For context, Stanford (Wen et al., 2025) reports ~1.0–1.1× token-efficiency speedup for the best alternative optimizers (Muon, Soap, NAdamW) at their largest scale (1.2B) — i.e. a ~10% speed gain over AdamW. AERO's measured 2.0× is a 100% speed gain — 10× the improvement that the best published optimizer delivers over AdamW, under the same methodology.
- **Test 2 (4-node 32×A100) — combined wall-clock on production-typical hardware:** the hardware-independent token-efficiency advantage (Lever 1, 2.0×) compounds with a measured per-step throughput advantage (Lever 2, 1.86×) into a combined wall-clock speedup of **3.7×** (on runs of 2× C or greater) — 47.0 h for AERO vs 175.2 h for AdamW at matched val/loss; a **73% GPU-hour savings**.
- **Stability:** single fixed configuration runs for the full 415K-step horizon with no loss spikes, no divergence, no restarts, and a smooth monotonic descent through the cosine schedule.
- **Confirmed at 2× C:** a matched AERO run at 2× C (val/loss 2.49) lands essentially on the same-asymptote projection, confirming the conservative ~2.0× token-efficiency advantage holds beyond the 1× C anchor; 4×/8× C remain projections.

9. Seed invariance — the advantage is not a lucky seed

A reasonable objection to a single-seed headline is that the 1.36B AERO@1× C anchor (2.55 nat) might be a fortunate draw — one good random seed that happens to flatter AERO. To rule this out we re-ran the comparison from scratch with a **second, independent random seed** on a **130M-parameter proxy model** (same architecture family, tokenizer, packed-C4 data, precision, and code path; 1× Chinchilla, ~139K steps). If AERO's lead were a seed artifact, a different seed would shrink or erase the gap.

Result — the gap is seed-invariant

Both seeds reproduce essentially the same AERO advantage. The AERO−AdamW validation-loss gap at the 139K-step horizon is **−0.14 nat** for seed 1234 and **−0.14 nat** for seed 5678 — identical to two decimal places. AERO's lead is a property of the training loop, not of seed 1234.

How to read Figure 6 — the advantage is not a lucky seed

What it shows: two AERO runs and two AdamW runs on a 130M proxy model at 1× Chinchilla (~139K steps), each optimizer run with two different random seeds (1234 and 5678) under otherwise identical conditions. The vertical axis is model quality (validation loss, lower is better); the horizontal axis is training progress (steps).
Takeaway: within each optimizer the two seeds land almost on top of each other, and AERO tracks below AdamW the whole way. The AERO−AdamW gap at the finish is −0.14 nat for *both* seeds — identical to two decimal places — so the advantage is a property of the training loop, not of a fortunate seed.

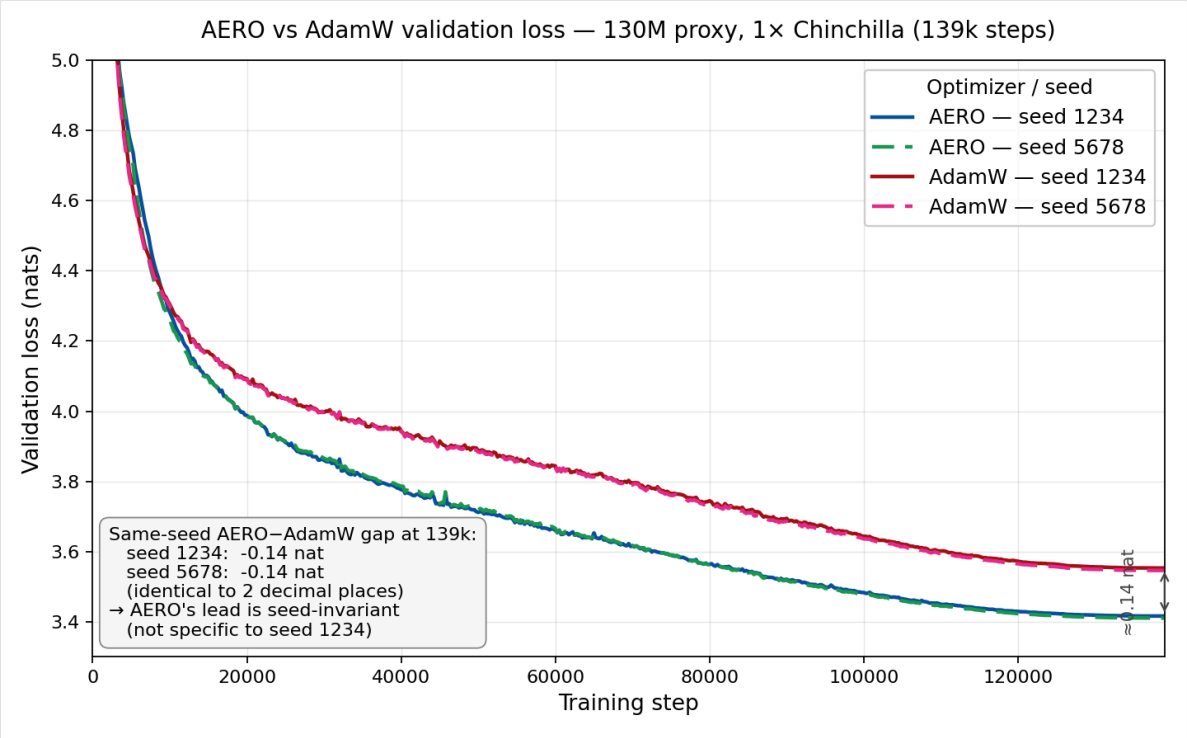


Figure 6. Seed-invariance check on a 130M proxy model at 1× Chinchilla (~139K steps). Two AERO runs (dark/light blue, seeds 1234 and 5678) and two matched AdamW runs (dark/light red, same two seeds) under otherwise identical conditions. Within each optimizer the two seeds are visually indistinguishable, and AERO tracks consistently below AdamW for the entire run. The same-seed AERO−AdamW gap at 139K steps is **−0.14 nat** for *both* seeds (identical to two decimal places), so AERO's advantage is seed-invariant — it is not specific to the seed used for the 1.36B headline runs.

This proxy-scale result complements the 1.36B headline in two ways. First, it confirms that the directly-measured 2.0× token-efficiency result rests on a robust, reproducible loss gap rather than seed noise. Second, it shows the AERO advantage is already present at 130M and persists to 1.36B, consistent with the smaller-scale sweep verification on the roadmap. The proxy gap (−0.14 nat at 1× C on a 130M model) is in the same direction and larger at the smaller scale than the 1.36B 1× C gap (−0.06 nat), supporting the conclusion that the headline number reflects a real, reproducible property of the AERO training loop rather than a lucky random-seed draw.

10. Outlook — why this is likely a lower bound

The 2× token-efficiency speedup measured here is obtained on the benchmark that most favors the AdamW baseline: a heavily filtered, English-only, single-domain web-crawl corpus (packed C4) at the 1× Chinchilla budget where Stanford's well-tuned-AdamW landscape sits at its strongest relative position vs alternative optimizers. AdamW's per-coordinate adaptive scaling assumes regular per-coordinate gradient statistics — an assumption that holds well on uniform corpora and poorly on heterogeneous ones; AERO's structural update geometry should average out coordinate-level noise more effectively as the input distribution becomes more heterogeneous. In addition, AERO holds a single fixed configuration cleanly across the full 415K-step horizon (no loss spikes, no divergence, no restarts), so the same recipe should transfer to noisier corpora and larger budgets without per-problem re-tuning. Both axes — data noise and total compute budget — are therefore expected to *increase* AERO's relative advantage in production-typical deployments (multi-domain enterprise mixtures, 4×–16× C hyperscaler budgets), not decrease it. The 2× headline is the floor; the production-deployment number is most likely larger.

References

1. Wen, K., Hall, D., Ma, T., & Liang, P. (2025). *Fantastic Pretraining Optimizers and Where to Find Them*. September 8, 2025. — Source of the token-efficiency methodology, the well-tuned / under-tuned AdamW framing (“Stanford trap”), and the 1.2B optimizer landscape used as the published reference in Figure 1a.
2. Hoffmann, J., Borgeaud, S., Mensch, A., et al. (2022). *Training Compute-Optimal Large Language Models* (Chinchilla). — Basis for the Chinchilla compute-budget convention (tokens ≈ 20× parameters at 1× C) used throughout this paper.
3. DevOpt Labs (2026). *AERO 1.36B SwiGLU C4 measured anchors and per-step timings.* Internal runs — Test 1 (1-node 8×H100 80GB: AdamW@1× C = 2.61, AdamW@2× C = 2.55, AERO@1× C = 2.55, AERO@2× C = 2.49) and Test 2 (4-node 32×A100: AdamW 760 ms/step, AERO 408 ms/step); plus the 130M seed-invariance proxy (seeds 1234 / 5678).

Generated from four matched-tuning anchors on Test 1 (1-node 8×H100 80GB: AdamW@1× C = 2.61, AdamW@2× C = 2.55, AERO@1× C = 2.55, AERO@2× C = 2.49) plus measured per-step times on Test 2 (4-node 32×A100: AdamW 760 ms/step, AERO 408 ms/step). Stanford landscape context: *Fantastic Pretraining Optimizers and Where to Find Them*, Wen et al., 2025. Values still underlined are projections to higher Chinchilla budgets from the fitted scaling law.

NOTICE

The information provided in this document is believed to be accurate and reliable as of the date provided. However, DevOpt Labs, Inc. (“DEVOPT LABS”) does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. DEVOPT LABS shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This publication supersedes and replaces all other documents for the DEVOPT LABS software and services described herein (the “Software”) that may have been previously supplied.

DEVOPT LABS reserves the right to make corrections, modifications, enhancements, improvements, and other changes to the Software and this document, at any time and/or to discontinue any product or service without notice. Customer should obtain the latest relevant document before placing orders and should verify that such information is current and complete.

DEVOPT LABS products and services are licensed and/or sold subject to the DEVOPT LABS standard terms and conditions supplied at the time of order acknowledgement, unless otherwise agreed in an individual agreement signed by authorized representatives of DEVOPT LABS and customer. DEVOPT LABS hereby expressly objects to applying any customer general terms and conditions with regard to the purchase or license of the DEVOPT LABS product or service referenced in this document.

The Software, and any product, system, model, application, or service that is developed, trained, optimized, or otherwise derived using the Software (each, a “Derived Product”), are not designed, authorized, or warranted to be suitable for use in high-risk applications, including medical, military, aviation, aerospace, autonomous or assisted-vehicle, or life-support systems, nor in any application where failure or malfunction of the Software or a Derived Product can reasonably be expected to result in personal injury, death, or property or environmental damage. DEVOPT LABS accepts no liability for the inclusion and/or use of the Software or any Derived Product in such applications, and therefore such inclusion and/or use is at customer’s own risk.

DEVOPT LABS makes no representation or warranty that the Software, or any Derived Product, will be suitable for any specified use without further testing or modification. Testing of all aspects of the Software is not necessarily performed by DEVOPT LABS. It is customer’s sole responsibility to ensure the Software and any Derived Product are suitable and fit for the application planned by customer and to perform the necessary testing for the application in order to avoid a default of the application, the Software, or the Derived Product. Weaknesses in customer’s product or system designs, including the manner in which the Software is trained, configured, or integrated, may affect the quality and reliability of any Derived Product and may result in additional or different conditions and/or requirements beyond those contained in this document. DEVOPT LABS does not accept any liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the Software or any Derived Product in any manner that is contrary to this document, or (ii) customer product, model, or system designs.

No license, either expressed or implied, is granted under any DEVOPT LABS patent right, copyright, or other DEVOPT LABS intellectual property right under this document. Information published by DEVOPT LABS regarding third-party products or services does not constitute a license from DEVOPT LABS to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from DEVOPT LABS under the patents or other intellectual property rights of DEVOPT LABS. Reproduction of information in this document is permissible only if reproduction is approved by DEVOPT LABS in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

ALL DEVOPT LABS DESIGN SPECIFICATIONS, REFERENCE MATERIALS, FILES, DOCUMENTATION, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” DEVOPT LABS MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, THE SOFTWARE, OR ANY DERIVED PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, DEVOPT LABS’ aggregate and cumulative liability towards customer for the Software and any products described herein shall be limited in accordance with the DEVOPT LABS terms and conditions of sale and/or license for the Software.