

AdamN: Accelerating Deep Learning Training via Nested Momentum and Exact Bias Handling

Mohamed Aboulsaad and Adnan Shaout *

The Electrical and Computer Engineering Department, The University of Michigan, Dearborn, MI 48128, USA; mboulsa@umich.edu

* Correspondence: shaout@umich.edu

Abstract

This paper introduces AdamN, a nested-momentum adaptive optimizer that replaces the single Exponential Moving Average (EMA) numerator in Adam/AdamW with a compounded EMA of gradients plus an EMA of that EMA, paired with an exact double-EMA bias correction. This yields a smoother, curvature-aware search direction at essentially first-order cost, with longer, more faithful gradient-history memory and a stable, warmup-free start. Under comparable wall-clock time per epoch, AdamN matches AdamW's final accuracy on ResNet-18/CIFAR-100, while reaching 80% and 90% training-accuracy milestones ~127 s and ~165 s earlier, respectively. On pre-benchmarking workloads (toy problems and CIFAR-10), AdamN shows the same pattern: faster early-phase convergence with similar or slightly better final accuracy. On language modeling with token-frequency imbalance—Wikitext-2-style data with training-only token corruption and a 10% low-resource variant—AdamN lowers rare-token perplexity versus AdamW without warmup while matching head and mid-frequency performance. In full fine-tuning of Llama 3.1-8B on a small dataset, AdamN reaches AdamW's final perplexity in roughly half the steps ($\approx 2.25 \times$ faster time-to-quality). Finally, on a ViT-Base/16 transferred to CIFAR-100 (batch size 256), AdamN achieves 88.8% test accuracy vs. 84.2% for AdamW and reaches 40–80% validation-accuracy milestones in the first epoch (AdamW reaches 80% by epoch 59), reducing epochs, energy use, and cost.

Keywords: AdamN; AdamW; debiasing; deep learning; Hessian; LLM; nested momentum; NLP; optimization; SGD

1. Introduction

The training of deep neural networks fundamentally relies on stochastic optimization, wherein model parameters are iteratively adjusted in directions indicated by minibatch gradients while ensuring stability of step sizes and preservation of generalization performance. The design of optimization algorithms is largely characterized by two dimensions: the mechanisms by which gradient directions are smoothed or accumulated, and the strategies by which step sizes are adapted across parameters. This dichotomy underpins the primary trade-offs observed among existing methods in terms of convergence speed, stability, and generalization quality. Despite the proliferation of algorithms in recent years, four families have emerged as the most widely adopted in practice. Stochastic gradient descent (SGD) with momentum remains a strong baseline, noted for its simplicity and robust generalization properties [1,2].

Academic Editors: Luca Di Nunzio, Gian Carlo Cardarilli and Sergio Spanò

Received: 13 January 2026

Revised: 29 January 2026

Accepted: 30 January 2026

Published: 3 February 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

AdaGrad rescales coordinates using the (cumulative) squared-gradient history, helping on sparse features [3]. RMSProp introduces the adaptation per parameter through an exponential moving average (EMA) of squared gradients. Adam integrates momentum with adaptive scaling and applies bias corrections [4,5], with AMSGrad providing a convergence-motivated variant [6]. Based on this, AdamW decouples the weight decay from Adam’s adaptive update, improving regularization control [7]. For a broader context, see the survey by Bottou, Curtis, and Nocedal [8].

Why a new optimizer? Despite there being many Adam variants, the following three gaps remain relevant in practice:

- (1) **No principled “momentum-of-momentum.”** The standard numerator uses one EMA. Simply stacking EMAs lengthens memory, but *double-counts cold-start bias*, shrinking early steps unless they are patched.
- (2) **Warmup dependency.** Schedules often rely on bespoke warmups to achieve stable early scaling, which complicates pipelines and increases failed runs.
- (3) **Opaque early-time scaling.** With one EMA numerator, there is no orthogonal knob to extend memory while keeping the instantaneous pass-through of new gradient information well controlled.

This paper presents a new optimizer called AdamN. AdamN’s main contribution is an EMA-of-EMA numerator with an exact double-EMA debiasing factor f_t that removes both inner and outer cold-start bias. The denominator and weight decay (WD) remain Adam-compatible (EMA of squared gradients; *decoupled WD*), so the method is drop-in with the same Big-O cost.

This contribution translates into the following real operational benefits:

- (4) **Faster time-to-quality:** Earlier accuracy milestones (e.g., 70–90% train accuracy) are reached sooner, shortening iteration cycles for model/ablation development and accelerating time-to-market.
- (5) **Lower energy and cost per experiment:** Reaching target accuracy in fewer epochs reduces GPU-hours and power draw for both research and production retraining.
- (6) **Schedule-friendly starts:** Exact debiasing reduces dependence on hand-tuned warmups, which simplifies pipelines and cuts failed runs caused by unstable early steps.
- (7) **Throughput on shared clusters:** Shorter “wall-time to useful model” improves cluster utilization; teams can run more variations under quota.
- (8) **Edge and small-budget training:** When computation or thermal headroom is limited (on-device or tiny clouds), AdamN’s faster launch and stable scaling help hit acceptable accuracy within tight budgets.

Research Questions (RQs) We organize our study around five questions and their evaluation metrics:

RQ1—Time-to-quality. Does AdamN reach target accuracy faster than AdamW/SGD while keeping final accuracy comparable?

Metric: wall-clock seconds to hit train/val milestones 40%, 50%, 60%, 70%, 80%, and 90%.

RQ2—Final quality at equal budget. At a fixed epoch/time budget, does AdamN match or exceed final test accuracy?

Metric: test accuracy at $E = 100$ and at a fixed wall-clock budget.

RQ3—Sensitivity and robustness. How sensitive is AdamN to the newly introduced β_2 -ramp, LR, and WD vs. AdamW/SGD?

Metric: mean $\pm$ SD (95% CI) of final test accuracy over 3–5 seeds per grid point.

RQ4—What “makes” AdamN work? Are gains primarily from the nested-EMA numerator or exact double-EMA debiasing?

Metric: ablation deltas on time-to-milestones and final accuracy when toggling nested on/off and exact vs. simple debiasing.

RQ5—Generalization to NLP under imbalance. Do AdamN’s rare-token update-efficiency gains hold beyond vision?

Metric: validation/test perplexity per frequency bin (head/mid/tail), tail PPL, and effective LR (row-wise RMS) on embeddings; fixed-budget final PPL.

The paper is organized as follows: Section 2 reviews the literature and the motivation for a new optimizer. Section 3 reviews the notation and setup of the related work. Section 4 presents *AdamN: Method and Full Derivation*, including the exact nested-EMA weights and the bias-correction factor f_t . Section 5 provides an algorithmic reference. Section 6 reports experiments: proof of concept and *CIFAR-10* pre-benchmarking, followed by the main *CIFAR-100* benchmark with ResNet-18 and ViT-B16. Section 7 presents the ablation. Section 8 discusses limitations and practical guidance, and Section 9 concludes.

2. Literature Survey and Motivation for a New Optimizer

2.1. Foundations Before SGD

Before stochastic methods became dominant, optimization in numerical analysis and early machine learning relied on full-batch techniques. The classical method of steepest descent, introduced by Cauchy in the 19th century [9], updates parameters along the negative gradient of the full dataset loss. While stable, this approach is computationally expensive for large datasets and scales poorly.

Other classical approaches include the following:

- (1) Newton’s method, introduced in the 17th century [10], and its quasi-Newton variants such as BFGS (developed in the 1970s [11–13]), which use curvature information for faster convergence on convex problems but require storing and inverting Hessian approximations.
- (2) Coordinate descent, dating back to early-20th-century optimization [14,15], which updates one parameter at a time and was practical for small-dimensional problems.
- (3) Conjugate gradient methods (Hestenes and Stiefel, 1952 [16]), which improved convergence for quadratic objectives by exploiting conjugacy instead of simple steepest directions.

These methods were effective in smaller-scale convex settings but became infeasible as datasets and neural networks grew. The breakthrough of stochastic gradient descent (SGD) by Robbins and Monro (1951) [17] was to replace the full gradient with an unbiased minibatch estimator. This drastically reduced per-step computation and enabled scaling, at the cost of higher variance—a trade-off that ultimately aided exploration and generalization in deep learning.

2.2. First-Order Foundations

SGD and Momentum: SGD with a global learning rate remains the default for large-scale supervised training. Polyak’s heavy-ball momentum [1] and deep-learning practice [2] accelerate progress in curved valleys by accumulating low-frequency components of the gradient. Nesterov’s look-ahead gradient [18] often improves stability on ill-conditioned objectives. Strengths: simplicity, strong generalization in vision, and predictable schedule design. Limitations: a single global step size struggles with anisotropy and poorly scaled features; early progress can be slow.

Adaptive Methods (AdaGrad/RMSProp): AdaGrad rescales coordinates using the (cumulative) squared-gradient history, helping on sparse features [3]. RMSProp trades the cumulative sum for an EMA so step sizes do not vanish [19]. These methods reduce sensitivity to global LR but can suffer from stale or overly smooth denominators when noise/curvature changes quickly.

2.3. Adam and Decoupled Weight Decay

Adam introduced EMAs for both first and second moments with explicit bias corrections that address cold-start underestimation; this makes early training more stable and typically faster [5]. AMSGrad provides a convergence-favoring variant by enforcing a nondecreasing second-moment estimate [6]. They are stable defaults, but the *single numerator EMA* forces a trade-off between memory and responsiveness; extending memory usually hurts early-step magnitude unless one introduces warmups.

AdamW: L2 added to the loss inside an adaptive method does not behave like classical weight decay; decoupling the decay step (AdamW) restores clean regularization and often improves validation accuracy at no extra cost [7]. It solves WD coupling but *does not address numerator memory vs. early scaling*.

2.4. Recent Developments

A wave of optimizers extends or rethinks Adam along several axes. AdaBelief modifies the second moment to track the “belief” in the gradient, often improving generalization relative to Adam [20]. Sharpness-aware methods such as SAM and its scale-invariant variant ASAM perturb parameters toward neighborhoods of lower loss curvature to improve generalization [21,22]. Shampoo brings practical block-wise second-order preconditioning at scale [23]. AdamP introduces a geometry-aware projection to mitigate over-adaptation and boost vision generalization [24]. Adan uses an adaptive Nesterov-style numerator for faster convergence, particularly in vision [25]. D-Adaptation removes manual learning-rate tuning by adapting the global scale automatically [26]. Sophia leverages a clipped Hessian estimate to realize a scalable stochastic second-order method with strong results for LLM pretraining [27]. Lion—discovered via symbolic search—updates with a sign-based momentum rule and is competitive in large-scale training [28].

These advances improve specific regimes (e.g., curvature awareness, sign-based steps, and scale-free training), yet none of them provide a principled, bias-corrected momentum-of-momentum that (a) extends numerator memory, (b) **exactly** fixes compounded cold-start bias, and (c) keeps Adam-like simplicity and cost.

2.5. Large-Batch Regime and Scaling

Large batches can hurt generalization by pushing toward sharper minima [29], although careful schedules and system design can scale training dramatically (e.g., 1 h ImageNet) [30]. Trust-ratio-style methods (e.g., LAMB) enable stable very-large-batch training by normalizing the step to the parameter norm [31].

2.6. What Current Methods Still Miss

Despite extensive progress [8], several gaps persist that motivate our approach:

- (1) No principled “momentum-of-momentum.” Adam uses a single EMA in the numerator. If one naively stacks EMAs to lengthen memory, the inner EMA’s cold-start bias remains uncorrected, shrinking early steps.
- (2) Warmup dependence. Many pipelines rely on hand-tuned warmups to overcome timid or erratic starts, even with Adam’s bias corrections [5].
- (3) Under-parameterized trade-off between responsiveness and inertia. With only one numerator EMA, tuning smoothness vs. freshness is constrained; there is no clean way to expose an additional, orthogonal knob that lengthens memory while keeping early steps properly scaled.
- (4) Opaque early-time scaling. Interactions between η , β_1 , β_2 , and the denominator can obscure how much of the current gradient passes through each step.

2.7. Motivation for AdamN

AdamN targets these gaps while remaining drop-in-compatible with Adam/AdamW. AdamN has the following characteristics:

- (1) Nested numerator (EMA-of-EMA) yields a triangular-with-exponential-tail-kernel, longer memory, and smoother directions than a single EMA.
- (2) Exact double-EMA debiasing provides a closed-form factor that removes both inner and outer cold-start shrinkage, eliminating the need for ad hoc warmups while preserving stability.
- (3) Transparent scaling. The decomposition into (i) freshness, (ii) exact bias factor, and (iii) adaptive denominator clarifies instantaneous step size and informs principled LR scaling.
- (4) Cost and compatibility. It has the same Big-O cost as Adam/AdamW with one extra buffer and retains decoupled weight decay and modern scheduling [7].
- (5) This directly targets the memory vs. early-scale gap while remaining drop-in- and scheduler-friendly.

2.8. Systematic Comparison with Related Nested/Multi-Momentum Approaches

To substantiate our claim regarding the novelty of AdamN's bias correction, we systematically analyzed recent optimizers that employ multiple momentum-like terms, as shown in Table 1.

Table 1. Systematic comparison with related nested/multi-momentum approaches.

Optimizer	Numerator Structure	Nested EMA?	Exact Double-EMA Debias?
Adam/AdamW	Single EMA of $\mathbf{g}_t$	No	N/A
Adan	$\mathbf{m}_t + (1 - \beta_2)(\mathbf{g}_t - \mathbf{g}_{t-1})$	No (additive)	No
AdaBelief	Single EMA of $\mathbf{g}_t$ (modified denom.)	No	N/A
Lion	$\text{sign}(\beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t)$	No	N/A
AdamN	EMA of (EMA of $\mathbf{g}_t$)	Yes	Yes

Adan's 'momentum difference' term $(\mathbf{g}_t - \mathbf{g}_{t-1})$ is structurally distinct from applying an EMA to the output of another EMA. Mathematically, Adan's numerator can be written as $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)(\mathbf{g}_t + \beta_2(\mathbf{g}_t - \mathbf{g}_{t-1}))$, which is a weighted combination of current and past gradients but does not exhibit the triangular-kernel weighting pattern (Equation (21)) that characterizes nested EMAs. Consequently, Adan does not require—nor does it provide—a double-EMA bias correction factor.

3. Notation and Setup

Let the parameters (all layers, all tensors: convolution kernels, linear weights, biases, etc.) be concatenated into a single vector $\theta \in \mathbb{R}^d$. At step t , given a minibatch B_t , the stochastic gradient $\mathbf{g}_t$ of the minibatch empirical loss with respect to all parameters can be defined as follows:

$$\mathbf{g}_t \triangleq \nabla_{\theta} \mathcal{L}(\theta_t; \mathcal{B}_t) \quad (1)$$

Let the learning rate be $\eta > 0$, the stabilizer be $\epsilon > 0$, and the gradient at step t be $\nabla J(\theta_t)$. When present, let the weight decay be $\lambda \geq 0$, decoupled unless otherwise noted.

Modern deep networks are trained by *stochastic first-order* methods that update parameters using minibatch gradients. To reason precisely about later variants (momentum, Adam/AdamW, and our proposed AdamN), we first fix notation for the minibatch loss and its gradient and make clear how learning rate, numerical stabilization, and *weight decay* (WD) enter

the update. This section establishes a common baseline against which the effects of momentum and adaptivity can be interpreted.

3.1. SGD and Momentum (Reference Form)

- (1) *Vanilla SGD*: SGD updates parameters by moving against the minibatch gradient with a global learning rate. It looks only at the current gradient to pick the next step. It is simple but slow, and it can get stuck or bounce around. Its updated parameters are as follows:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta \nabla J(\theta_t) \\ \theta_{t+1} - \theta_t &= \eta \nabla J(\theta_t) \\ \Delta \theta_t &= \eta \nabla J(\theta_t)\end{aligned}\tag{2}$$

With decreasing η , SGD achieves classical stochastic rates in convex settings. With constant η , it converges to a noise floor whose radius depends on η and the variance of the gradient. Despite its simplicity, SGD remains competitive in large, supervised tasks, especially with data augmentation and carefully designed learning-rate schedules.

Limitations:

A single global step size as seen in (2) can be inadequate for highly anisotropic problems.

Early training can lag adaptive methods on sparse or poorly scaled features.

Large batch caution: very large batches may reduce test accuracy by converging to sharper minima, as noted by [9]; see also large batch scheduling successes in [11].

- (2) *Polyak (Heavy-Ball) Momentum*: This adds velocity (v_t). Instead of just the current gradient, it mixes in part of the last update's direction. That way, steps get smoother and faster in steady directions and reduce zigzagging in high-curvature regions by accumulating low-frequency gradient information.

$$\begin{aligned}v_{t+1} &= \mu v_t + \nabla J(\theta_t) \\ \theta_{t+1} &= \theta_t - \eta v_{t+1} \\ \Delta \theta_t &= \eta v_{t+1}\end{aligned}\tag{3}$$

where μ is the momentum term $\in [0, 1]$ that decides how much of the past velocity is kept.

- (3) *Nesterov Accelerated Gradient (NAG)*: NAG computes the gradient at a look-ahead position [18] as follows:

$$v_t = \mu v_{t-1} - \eta \nabla \mathcal{L}(\theta_t + \mu v_{t-1})\tag{4}$$

$$\theta_{t+1} = \theta_t + v_t\tag{5}$$

This anticipatory step in (5) typically improves stability and effective conditioning. In practice, set momentum $\mu \in [0.9, 0.99]$ (e.g., 0.9 for noisier tasks and 0.95–0.99 for smoother regimes). We choose the base learning rate by a brief sweep or LR-range test, then use a cosine decay or step decay schedule. For very deep networks or highly non-stationary early gradients, we include a short warmup (e.g., 3–10 epochs or 1–5% of total steps) to avoid unstable starts.

3.2. Adam (Adaptive Moment Estimation)

Adam combines a momentum-like first moment (EMA of gradients) with an adaptive second moment denominator (EMA of squared gradients) and applies bias corrections so that early estimates are not underestimated [5].

- (1) The first moment (momentum of gradients) is derived as follows:

$$\begin{aligned}m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ &\Rightarrow (1 - \beta_1) \sum_{i=1}^t \beta_1^{t-i} g_i\end{aligned}\tag{6}$$

- (2) The second moment (EMA of squared gradients) is derived as follows:

$$\begin{aligned} v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\ \Rightarrow (1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} g_i^2 \end{aligned} \quad (7)$$

The result is a fast, directionally smooth, scale-aware update that is straightforward to tune. The bias correction addresses the cold-start bias, stabilizing the critical early iterations where gradients are volatile. This combination explains Adam's widespread adoption in NLP, vision, and reinforcement learning.

The standard bias-correction derivation is as follows:

First, we do not need the gradients themselves to be constant, only that their mean and second moment stay the same assuming the gradient statistics are stationary. Concretely:

- (1) The average gradient is constant: $E[g_i] = \mu$ for all steps i .
- (2) The average squared gradient is constant: $E[g_i^2] = \nu$ for all steps i .
- (3) By using (6) and (7), taking expectations of the first and second moments of Adam under this assumption will be as follows:

$$\begin{aligned} E[m_t] &= (1 - \beta_1) \sum_{i=1}^t \beta_1^{t-i} E[g_i] \\ &= \mu(1 - \beta_1) \sum_{k=0}^{t-1} \beta_1^k \\ &\Rightarrow \mu(1 - \beta_1^t) \end{aligned} \quad (8)$$

$$\begin{aligned} E[v_t] &= (1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} E[g_i^2] \\ &= \nu(1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k \\ &\Rightarrow \nu(1 - \beta_2^t) \end{aligned} \quad (9)$$

- (4) For $E[m_t]$, the factor $(1 - \beta_1^t)$ in (8) is less than 1 for small t , so $E[m_t]$ is smaller than μ .
- (5) As $t \rightarrow \infty$, $\beta_1^t \rightarrow 0$ and $1 - \beta_1^t \rightarrow 1$, so the bias goes away.
- (6) The same holds for v_t : $E[v_t] = \nu(1 - \beta_2^t)$, so it also starts too small.

That is why they are biased toward zero. We started at zero, and the EMA underestimates the true mean and variance early on by a factor of $1 - \beta^t$.

Without factors $(1 - \beta_i^t)$ where $i \in [1, 2]$, the first and second moments are systematically underestimated during initialization [5]. This underestimation is most pronounced when β_2 is large and can cause instability in the first stages of training; AMSGrad [6] analyzes related convergence issues.

Bias-corrected estimates:

Divide out those factors to remove the bias:

$$\hat{m}_t = \frac{m_t}{(1 - \beta_1^t)}, \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (10)$$

The final update expressed in terms of past gradients g_j is as follows:

$$\Delta\theta_t = \eta \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon} \quad (11)$$

Substitute (10) into (11) and simplify, then the weighted sum over g_j is as follows:

$$\Delta\theta_t = \sum_{j=1}^t \alpha_{t,j} g_j$$

where

$$\alpha_{t,j} = \eta \frac{(1 - \beta_1)}{1 - \beta_1^t} \frac{\beta_1^{t-j}}{\sqrt{\frac{(1 - \beta_2)}{1 - \beta_2^t} \sum_{k=1}^t \beta_2^{t-k} g_k^2 + \epsilon}} \quad (12)$$

Current gradient coefficient:

For the most recent gradient g_t , set $(j = t)$ in (12):

$$\alpha_{t,t} = \eta \frac{(1 - \beta_1)}{1 - \beta_1^t} \frac{1}{\sqrt{\frac{(1 - \beta_2)}{1 - \beta_2^t} \sum_{k=1}^t \beta_2^{t-k} g_k^2 + \epsilon}} \quad (13)$$

Defaults. $\beta_1 = 0.9$, $\beta_2 \in [0.99, 0.999]$, $\epsilon = 10^{-8}$. Sometimes we reduce β_2 for a greater response and apply a warmup if the early gradients spike.

Caveats:

- (1) It can yield slightly weaker generalization than tuned SGD on some vision tasks; schedule design and regularization are critical.
- (2) Warmup is often required for very deep or highly regularized models.

Notes:

- (1) If gradients are not stationary, the correction still removes the *zero-init bias*.
- (2) If a gradient is large, v_t grows, which shrinks the effective learning rate. If it is small, the step size stays larger.
- (3) v_t is an adaptive scale factor, not another momentum. Adam has only *one* momentum: m_t .

Thus, Adam combines:

- (1) *Momentum* (past gradients for direction).
- (2) *Adaptive scaling* (past magnitudes for step size).

3.3. AdamW (Decoupled Weight Decay)

Motivation: In standard Adam, adding L2 regularization to the loss does not match classical weight decay, because the adaptive denominator mixes shrinkage with rescaling. AdamW resolves this by decoupling the weight decay step from the adaptive update [7] as follows:

- (1) Decay:

$$\theta \leftarrow (1 - \eta\lambda) \quad (14)$$

- (2) Adaptive step:

$$\theta \leftarrow \theta - \eta \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon) \quad (15)$$

This restores weight decay as pure multiplicative shrinkage and frequently improves validation/test performance compared to Adam + L2 at equal computation.

Defaults: Same as Adam, with $\lambda \in [10^{-4}, 10^{-2}]$ depending on the model and augmentation.

What to expect:

Cleaner, schedule-independent control of regularization.

Often stronger generalization than Adam + L2, with identical computational cost.

Regularization: L2 vs. decoupled decay:

L2 regularization adds $\frac{\lambda}{2}|\theta|^2$ to the loss, contributing $\lambda\theta$ to the gradient. In Adam, this term is normalized by $(\sqrt{\hat{v}_t} + \epsilon)$, coupling shrinkage with adaptation.

The decoupled decay in (14) applies $\theta \leftarrow (1 - \eta\lambda)$ outside the adaptive update, yielding uniform shrinkage independent of local rescaling [7]. This separation improves controllability and often leads to better generalization.

4. AdamN: Contributions and Method

4.1. Contributions

We introduce AdamN, a nested-momentum adaptive optimizer which has the following qualities:

- (1) It introduces a second EMA on the numerator (a true nested momentum: $g \rightarrow v \rightarrow a$) which acts as a triangular kernel with an exponential tail, yielding a smoother and longer memory update direction than a single EMA.
- (2) It derives an exact double-EMAs bias correction $f_t(\beta_1, \beta_2)$ that removes the cold-start shrinkage inherited from nested EMAs. Intuitively, the nested numerator behaves as a data-adaptive controller that tempers early step magnitudes, preventing overshoot without requiring hand-tuned warmup.
- (3) It retains Adam-style adaptive braking through a second moment s_t , using decoupled weight decay.
- (4) It demonstrates faster early progress at stable scaling on ResNet-18/CIFAR-100 while matching or slightly exceeding AdamW's final accuracy at similar computation.

4.2. Method and Foundation

Let us also define EMA elements as $\beta_1, \beta_2, \beta_3 \in [0, 1)$, then v_t, a_t and s_t are defined as follows:

$$\underset{\substack{\text{EMA of} \\ \text{gradients}}}{v_t} = \beta_1 v_{t-1} + (1 - \beta_1) g_t = (1 - \beta_1) \sum_{k=1}^t \beta_1^{t-k} g_k \quad (16)$$

$$\underset{\substack{\text{EMA of } v_t}}{a_t} = \beta_2 a_{t-1} + (1 - \beta_2) v_t = (1 - \beta_2) \sum_{k=1}^t \beta_2^{t-k} v_k \quad (17)$$

$$\underset{\substack{\text{EMA of} \\ \text{squared gradients}}}{s_t} = \beta_3 s_{t-1} + (1 - \beta_3) g_t^2 = (1 - \beta_3) \sum_{k=1}^t \beta_3^{t-k} g_k^2 \quad (18)$$

where β_1 controls the memory of v_t (1st moment), β_2 controls the memory of a_t (nested numerator), and β_3 controls the memory of s_t (2nd moment/brakes). From (16) and (17), substitute v_k into a_t and collect weights on g_j , which will give the following:

$$a_t = (1 - \beta_2)(1 - \beta_1) \sum_{j=1}^t g_j \left(\underset{\substack{\text{kernel on } g_j}}{\sum_{k=j}^t \beta_2^{t-k} \beta_1^{k-j}} \right) \quad (19)$$

where t is the current time step/iteration (1,2,3, ...), k is a summation index over time when unrolling an EMA (in which v_k terms contribute to a_t), and j is a summation index over gradient timestamps (in which g_j terms go into v_k and then a_t).

The weight decay $\lambda \geq 0$ is decoupled. All operations involving powers, division, and square roots are applied elementwise.

4.3. Nested Numerator as a Triangular Kernel

Evaluate the inner sum in (19) as follows:

$$\text{kernel on } g_j = \beta_2^{t-j} \sum_{m=0}^{t-j} \left(\frac{\beta_1}{\beta_2} \right)^m \quad (20)$$

Hence, the exact weights on g_j are as follows:

$$w_{t,j} = \begin{cases} (1 - \beta_1)(1 - \beta_2) \frac{\beta_2^{t-j+1} - \beta_1^{t-j+1}}{\beta_2 - \beta_1}, & \beta_1 \neq \beta_2, \\ (t - j + 1) \beta^{t-j} (1 - \beta)^2, & \beta_1 = \beta_2 = \beta. \end{cases} \quad (21)$$

Thus

$$a_t = \sum_{j=1}^t w_{t,j} g_j \quad (22)$$

The equal-betas case in (21) illustrates why nested EMAs possess much longer effective memory.

In this setting, the numerator acts as a triangular kernel with an exponential tail: the linear factor $(t - j + 1)$ increases with the age of each gradient before being exponentially damped by β^{t-j} . As a result, when β is large, the numerator becomes more inertial, retaining the past information significantly longer.

4.4. Bias Correction

Simple (insufficient) bias correction (Adam-style warmup): The nested numerator is unbiased via

$$\hat{a}_t = \frac{a_t}{1 - \beta_2^t} \quad (\text{leaves inner-EMA bias}) \quad (23)$$

With simple correction, the early steps are still biased low because a_t inherits bias from v_t .

Exact bias correction: The nested numerator is unbiased via

$$\hat{a}_t = \frac{a_t}{f_t(\beta_1, \beta_2)} \quad (24)$$

This is the exact analog of Adam's $\hat{m}_t = m_t / (1 - \beta_1^t)$ in (10) but for a double EMA of g_t . It amplifies early step safely and often shortens time-to-50/60-percent milestones.

It can also be formally shown that this expression corresponds to the total accumulated weight at time t .

To derive $f_t(\beta_1, \beta_2)$, unroll the recurrences (assume $E[g_j] \equiv g$ for all j to compute the expectation/gain factor) as follows:

(1) First EMA:

$$v_k = (1 - \beta_1) \sum_{j=1}^k \beta_1^{k-j} g_j \Rightarrow E[v_k] = (1 - \beta_1) g \sum_{j=1}^k \beta_1^{k-j} \quad (25)$$

Therefore:

$$E[v_k] = (1 - \beta_1) g \cdot \frac{1 - \beta_1^k}{1 - \beta_1} = (1 - \beta_1^k) g. \quad (26)$$

$$\boxed{E[v_k] = (1 - \beta_1^k) g}$$

(2) Second EMA:

$$a_t = (1 - \beta_2) \sum_{k=1}^t \beta_2^{t-k} v_k \Rightarrow E[a_t] = (1 - \beta_2) \sum_{k=1}^t \beta_2^{t-k} E[v_k] \quad (27)$$

Plug in (26):

$$\begin{aligned} E[a_t] &= (1 - \beta_2) \sum_{k=1}^t \beta_2^{t-k} (1 - \beta_1^k) g \\ &= \left[(1 - \beta_2) \sum_{k=1}^t \beta_2^{t-k} - (1 - \beta_2) \sum_{k=1}^t \beta_2^{t-k} \beta_1^k \right] g = f_t(\beta_1, \beta_2) \cdot g \end{aligned} \quad (28)$$

After evaluating the two geometric sums in (28), the exact debiasing factor for nested momentum will be defined as follows:

$$f_t(\beta_1, \beta_2) = \begin{cases} (1 - \beta_2^t) - (1 - \beta_2) \frac{\beta_1(\beta_2^t - \beta_1^t)}{\beta_2 - \beta_1}, & \beta_1 \neq \beta_2, \\ (1 - \beta^t) - (1 - \beta) t \beta^t, & \beta_1 = \beta_2. \end{cases} \quad (29)$$

Remark on stationarity: The derivation of $f_t(\beta_1, \beta_2)$ assumes stationary gradient statistics ($E[g_j] \equiv g$), following the same analytical framework used to derive Adam's bias correction [5]. This assumption serves to identify the functional form of the zero-initialization bias—the systematic underestimation caused by $v_0 = a_0 = 0$ —rather than to model the true (non-stationary) gradient distribution during training.

In practice, gradient statistics are highly non-stationary, particularly in early epochs. However, the bias correction remains effective because it addresses the initialization artifact: at $t = 1$, the unnormalized estimates v_1 and a_1 are scaled by factors that vanish as $t \rightarrow 0$, regardless of the gradient's true mean. The correction factor $f_t(\beta_1, \beta_2)$ compensates for this cold-start shrinkage.

For double EMAs, the non-stationarity concern is more pronounced because the outer EMA accumulates bias from the inner EMA's already-biased estimates. Our exact factor accounts for this compounding effect (the cross-term $\frac{\beta_1(\beta_2^t - \beta_1^t)}{\beta_2 - \beta_1}$ in Equation (29)), which simple cascaded corrections would miss. Empirically, AdamN's consistent performance across diverse tasks—where gradient statistics change dramatically during training—validates the practical robustness of our approach.

4.5. Freshness

By plugging $j = t$ into (21), the instantaneous pass-through of g_t into a_t (pre-bias) is given as follows:

$$w_{t,t} = (1 - \beta_1)(1 - \beta_2). \text{ (uncorrected)} \quad (30)$$

The bias correction scales all weights by $1/f_t$ to remove the early-time shrinkage, so the effective normalized coefficient is as follows:

$$\hat{w}_{t,t} = \frac{(1 - \beta_1)(1 - \beta_2)}{f_t(\beta_1, \beta_2)}. \text{ (biascorrected)} \quad (31)$$

Freshness quantifies how much new gradient information enters the numerator at each step. Unlike cumulative weights, it isolates the instantaneous pass-through for the newest gradient.

For example, by recalling (16) and (17), AdamN's freshness is as follows:

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) g_t, \quad a_t = \beta_2 a_{t-1} + (1 - \beta_2) v_t$$

Expanding shows that the coefficient in the current g_t in a_t is as follows:

$$\underbrace{(1 - \beta_1)}_{\text{from } v_t} \underbrace{(1 - \beta_2)}_{\text{from } a_t} = \boxed{(1 - \beta_1)(1 - \beta_2)}.$$

That boxed term is what we call *freshness*—how much *new gradient information* is injected at this step (before bias-correction/normalization/LR), while AdamW freshness is as follows: $\boxed{1 - \beta_1}$ as seen in (6).

Learning rate scaling and β_2 : The tolerance to learning rate of AdamN depends directly on the freshness term $(1 - \beta_1)(1 - \beta_2)$.

When β_2 is small (e.g., $\beta_1 = 0.9, \beta_2 = 0.1$), the freshness is $\Rightarrow (1 - 0.9)(1 - 0.1) = 0.09$, nearly the same as AdamW's $\Rightarrow (1 - 0.9) = 0.1$.

In this case, AdamN tolerates only about $0.1/0.09 \approx 1.1 \times$ higher learning rates—effectively without an advantage.

By contrast, when β_2 is large (e.g., $\beta_1 = 0.6, \beta_2 = 0.9$), the freshness shrinks to 0.04, and the bias correction amplifies the initial steps.

To get comparable aggressiveness, a simple first-order scaling is defined as follows:

$$\alpha_N \approx \alpha_W \cdot \frac{1 - \beta_1^W}{(1 - \beta_1^N)(1 - \beta_2^N)} \approx \alpha_W \cdot \frac{0.1}{0.04}.$$

Here, AdamN can run safely at $\sim 2.5 \times$ the learning rate of AdamW, achieving faster early progress without destabilization.

Thus, the LR advantage of AdamN manifests itself primarily in the high β_2 regime, where its nested numerator would otherwise be too inertial without correction.

This behavior is illustrated in Table 2.

Table 2. Effect of β_2 on AdamN's freshness and LR tolerance.

Setting	Fresh. (AdamW)	Fresh. (AdamN)	α_N/α_W
$\beta_1 = 0.9, \beta_2 = 0.1$	0.10	0.09	$\approx 1.1 \times$
$\beta_1 = 0.9, \beta_2 = 0.6$	0.10	0.04	$\approx 2.5 \times$

4.6. Interpretation of β_2 .

Effect of β_2 on AdamN dynamics: Large β_2 introduces two competing effects that must be understood jointly:

1. *Reduced freshness (raw numerator).* The uncorrected weight on the current gradient g_t in a_t is $(1 - \beta_1)(1 - \beta_2)$, which vanishes as $\beta_2 \rightarrow 1$. This makes the numerator increasingly *inertial*—dominated by past gradients rather than new information.
2. *Increased bias-correction amplification.* The factor $f_t(\beta_1, \beta_2)$ grows slowly for large β_2 , causing the bias-corrected numerator $\hat{a}_t = a_t/f_t$ to be amplified more aggressively in early epochs.

These effects can appear contradictory: large β_2 reduces sensitivity to new gradients (effect 1) while potentially increasing step magnitude (effect 2). The practical consequence is that high- β_2 configurations produce steps that are *large but delayed*; the optimizer moves confidently in outdated directions. This explains our ablation findings: $\beta_2 \geq 0.8$ degrades both speed and final accuracy because the numerator cannot adapt quickly to the changing loss landscape, regardless of step magnitude.

We therefore recommend $\beta_2 \in [0.1, 0.3]$ for most tasks, where freshness remains high and the bias correction provides stable early scaling without excessive inertia.

4.7. Instantaneous Learning Rate (instLR) and Comparison

Using (31), we define effective instantaneous scaling—coefficient—on the current gradient as follows:

$$\text{instLR}_t \equiv \alpha \hat{w}_{t,t} \frac{1}{\sqrt{\hat{s}_t} + \epsilon} \quad (32)$$

which clarifies why AdamN can launch fast yet remain stable.

This multiplier scales the contribution of g_t (and, via the weights above, all past g_j) and is the **real step size** the optimizer effectively takes at time t , not just the scheduler's base learning rate α .

The numerator is amplified correctly at cold start, while the denominator tempers spikes. At low β_2 (e.g., 0.1), exact vs. simple bias corrections are indistinguishable due to denominator absorption. But at higher β_2 (e.g., 0.8), exact bias correction yields consistently higher instLR and faster early convergence, as confirmed in Table 3.

Table 3. Comparison of AdamN with simple vs. exact bias correction ($\beta_1 = 0.9, \beta_2 = 0.8$). Exact correction yields slightly higher instLR and faster convergence in early epochs.

Epoch	AdamN (Simple Bias)				AdamN (Exact Bias)			
	instLR	step_RMS	Train Acc	Val Acc	instLR	step_RMS	Train Acc	Val Acc
1	0.155	1.42×10^{-4}	6.55%	13.84%	0.164	1.43×10^{-4}	7.02%	14.60%
2	0.360	1.91×10^{-4}	7.56%	13.88%	0.375	1.93×10^{-4}	7.87%	15.22%
3	0.629	2.75×10^{-4}	10.58%	17.02%	0.647	2.74×10^{-4}	10.64%	19.20%
4	0.958	3.80×10^{-4}	14.88%	22.96%	0.971	3.55×10^{-4}	15.25%	23.62%
5	1.349	4.40×10^{-4}	19.61%	29.16%	1.345	5.06×10^{-4}	20.63%	29.72%
6	1.517	4.63×10^{-4}	24.85%	33.94%	1.501	4.65×10^{-4}	25.62%	33.48%

These closed forms assume the usual initialization $v_0 = a_0 = 0$. If we start from non-zero v_0, a_0 , extra terms like $\beta_1^t v_0$ and $\beta_2^t a_0$ appear, but they decay fast and are usually negligible.

Digital Signal Processing (DSP) Analogy: From the DSP discipline, the bias factor $f_t(\beta_1, \beta_2)$ plays the role of a phase correction, aligning the nested-EMA numerator with the true gradient “phase” by removing the systematic lag at cold start, while the instantaneous learning rate (instLR) acts as an amplitude envelope, setting the step magnitude after normalization. Together they resemble phase and amplitude control in modulation: f_t corrects the trajectory (phase alignment), and instLR governs update strength (amplitude). This perspective clarifies why exact f_t matters most at large β_2 (long memory), where naive debiasing leaves a persistent phase lag that directly suppresses the effective amplitude.

4.8. Update Rule (Bias-Corrected) and Weighted-Sum View

The effective per-coordinate step is as follows:

$$\Delta\theta_t = -\eta \frac{\hat{a}_t}{\sqrt{\hat{s}_t} + \epsilon}, \quad \theta_{t+1} = \theta_t + \Delta\theta_t \quad (33)$$

where $\hat{a}_t$ is the exact debiased, noise-smoothed numerator and $\hat{s}_t$ the usual second moment. This combination is a **richer diagonal preconditioner** than Adam/AdamW's—closer to second-order behavior—yet remains first-order in cost.

With decoupled decay $\theta \leftarrow (1 - \eta\lambda)\theta$ and using (18), the bias-corrected denominator, like Adam's correction, will be as follows:

$$\sqrt{\hat{s}_t} = \sqrt{\frac{s_t}{1 - \beta_3^t}} = \sqrt{\frac{(1 - \beta_3) \sum_{k=1}^t \beta_3^{t-k} g_k^2}{1 - \beta_3^t}} \quad (34)$$

Equivalently, the update is a weighted sum over past gradients, and it is defined as follows:

$$\Delta \theta_t = \sum_{j=1}^t \alpha_{t,j} g_j \quad (35)$$

With

$$\alpha_{t,j} = \eta_t \frac{w_{t,j}}{f_t(\beta_1, \beta_2)} \frac{1}{\sqrt{\hat{s}_t} + \epsilon} \quad (36)$$

5. Algorithm and Complexity

AdamN adds one EMA buffer (a_t) compared to Adam/AdamW. The recurrence relations (lines 6–8) in Figure 1 each require $O(d)$ elementwise operations; the scalar bias factor f_t is computed in $O(1)$. Total per-step complexity is $O(d)$ time and $O(3d) = O(d)$ space, identical to Adam/AdamW up to a constant factor of $\sim 1.5\times$ in memory. The explicit weighted-sum representation (Equation (22)) is provided for theoretical insight and is never computed during runtime.

- 1: Given: Base LR $\alpha = 1^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.1$, $\beta_3 = 0.999$, $\epsilon = 10^{-8}$, WD $\lambda \in R$.
- 2: Initialize: $t \leftarrow 0$, parameter vector $\theta_{t=0} \in R^n$, first moment $v_{t=0} \leftarrow 0$, nested momentum $a_{t=0} \leftarrow 0$, $\text{instMult}_{t=0} \in R$.
- 3: Repeat,
- 4: $t \leftarrow t + 1$.
- 5: $\nabla(\theta_{t-1}) \leftarrow \text{SelectBatch}(\theta_{t-1})$ (compute gradient).
- 6: $g_t \leftarrow \nabla f_t(\theta_{t-1})$,
- 7: $v_t \leftarrow \beta_1 v_{t-1} + (1 - \beta_1) g_t$.
- 8: $w_{t,j} \leftarrow (1 - \beta_1)(1 - \beta_2) \frac{\beta_2^{t-j+1} - \beta_1^{t-j+1}}{\beta_2 - \beta_1}$.
- 9: $f_t(\beta_1, \beta_2) \leftarrow 1 - \beta_2^t - (1 - \beta_2) \frac{\beta_1(\beta_2^t - \beta_1^t)}{\beta_2 - \beta_1}$.
- 10: $a_t \leftarrow \sum_{j=1}^t w_{t,j} g_j$,
- 11: $s_t \leftarrow \beta_3 s_{t-1} + (1 - \beta_3) g_t^2$.
- 12: $\hat{a}_t \leftarrow \frac{a_t}{f_t(\beta_1, \beta_2)}$,
- 13: $\hat{s}_t \leftarrow \frac{s_t}{1 - \beta_3^t}$,
- 14: $\hat{w}_{t,t} \leftarrow \frac{(1 - \beta_1)(1 - \beta_2)}{f_t(\beta_1, \beta_2)}$.
- 15: $\text{instMult}_t \leftarrow \hat{w}_{t,t} \frac{1}{\sqrt{\hat{s}_t} + \epsilon} * \text{ScheduleMultiplier}(t)$.
- 16: $\text{instLR}_t \leftarrow \alpha * \text{instMult}_t$.
- 17: $\theta_t \leftarrow \theta_{t-1} - \text{instLR}_t \frac{\hat{a}_t}{\sqrt{\hat{s}_t} + \epsilon} - \text{instMult} * \lambda \theta_{t-1}$.
- 18: Until stopping criterion is met.
- 19: Return optimized parameters θ_t .

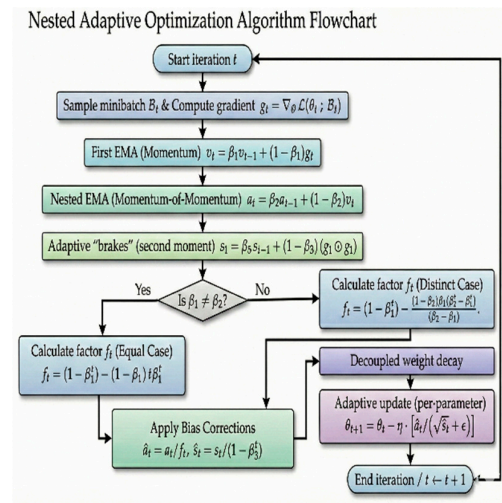


Figure 1. AdamN algorithm flowchart.

6. Experiments

Before moving to standard benchmarks such as CIFAR-100, we first conducted a series of progressively harder toy experiments to probe the dynamics of AdamN. These tests illustrate why a nested numerator plus adaptive braking is necessary.

For each baseline, we selected the hyperparameters yielding the highest validation performance and report that run.

This section will present the following experiments using the hyperparameters shown in Table 3:

- (1) A proof of concept showing why the denominator is essential.
- (2) CIFAR-10 as pre-benchmarking.
- (3) CIFAR-100 is the main benchmark.
- (4) Additional MNIST/EMNIST runs also show marked speed-to-accuracy gains.
- (5) NLP rare-token evaluation using a small transformer language model on a Wikitext-2-style corpus.

6.1. Hyperparameter Search Protocol

For each optimizer, we performed a grid search over the following spaces as seen in Table 4:

Table 4. Hyperparameter search protocol.

Optimizer	Learning Rate	Weight Decay	Other
AdamN	$\{3 \times 10^{-4}, 6 \times 10^{-4}, 1 \times 10^{-3}\}$	$\{1 \times 10^{-4}, 5 \times 10^{-4}\}$	$\beta_2 \in \{0.1, 0.2, 0.3\}$
AdamW	$\{3 \times 10^{-4}, 6 \times 10^{-4}, 1 \times 10^{-3}, 3 \times 10^{-3}\}$	$\{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$	—
Adam	$\{3 \times 10^{-4}, 6 \times 10^{-4}, 1 \times 10^{-3}, 3 \times 10^{-3}\}$	$\{0, 1 \times 10^{-4}, 5 \times 10^{-4}\}$	—
SGD	$\{1 \times 10^{-2}, 5 \times 10^{-2}, 1 \times 10^{-1}\}$	$\{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$	Momentum $\in \{0.9, 0.95\}$

All other hyperparameters ($\beta_1 = 0.9$, $\beta_3 = 0.999$, $\epsilon = 10^{-8}$ for Adam variants; cosine schedule for all) were held constant at standard defaults.

Search strategy: Full grid search (no random or Bayesian optimization).

Runs per configuration: Single run during search; 3 seeds for selected configurations.

Selection criterion: Configuration with highest validation accuracy at epoch 100.

Stopping criterion: Fixed 100 epochs (no early stopping during search).

Budget: AdamN: 18 configurations; AdamW: 12 configurations; Adam: 12 configurations; SGD: 12 configurations. Total: 54 runs on CIFAR-100/ResNet-18.

The selected hyperparameters for each optimizer, as shown in Table 5, represent the configuration achieving best validation performance in this search.

Table 5. Experiment hyperparameters.

Optimizer	Hyper-Parameter	Vision Task				NLP	Notes
		CNN		ViT			
		Full Training	Transfer Learning	Full Training	Transfer Learning		
AdamN	η	1×10^{-3}	1×10^{-4}	1×10^{-3}	1×10^{-4}	1×10^{-3}	η : Base LR. No warmup, cosine scheduling β_1 : Momentum of g_t . β_2 : Momentum of v_t . β_3 : Denominator memory ϵ : Stability WD (λ): Decoupled AMP is enabled for <i>all</i> optimizers. The β_2 ramp is an optional technique discussed in Sec. VIII. AdamW uses ‘fused = True’; Adam/SGD use ‘foreach = True’ for fair runtime comparison.
	$(\beta_1, \beta_2, \beta_3)$	(0.9, 0.1, 0.999)	(0.9, 0.1, 0.999)	(0.9, 0.1, 0.999)	(0.9, 0.1, 0.999)	(0.4, 0.1, 0.999)	
	ϵ	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	
	WD (λ)	1×10^{-8}	1×10^{-8}	1×10^{-8}	10^{-4}	1×10^{-8}	
	Batch Size	10^{-4}	10^{-5}	5×10^{-3}	256	10^{-4}	
	Epoch	128	128	256	100	-	
Adam/AdamW	η	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	AdamW uses ‘fused = True’; Adam/SGD use ‘foreach = True’ for fair runtime comparison.
	(β_1, β_2)	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)	
	ϵ	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	
	WD (λ)	5×10^{-4}	10^{-5}	$0/5 \times 10^{-2}$	$5 \times 10^{-4}/5 \times 10^{-2}$	$0/5 \times 10^{-4}$	
	Batch Size	128	128	256	10^{-2}	-	
	Epoch	100	100	100	256	10	
SGD	η	0.1	1×10^{-3}	1×10^{-1}	1×10^{-3}	1×10^{-1}	Lion: As per [16]; cosine schedule. Adan: As per [12]; cosine schedule. AdaBelief: As per [7]; Cosine schedule.
	Momentum	0.9	0.9	0.9	0.9	0.9	
	ϵ	5×10^{-4}	10^{-5}	0	5×10^{-3}	0	
	WD (λ)	5×10^{-4}	10^{-5}	0	5×10^{-3}	0	
	Batch Size	128	128	256	256	-	
	Epoch	100	100	100	100	10	
Lion	η	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	
	(β_1, β_2)	(0.9, 0.99)	(0.9, 0.99)	(0.9, 0.99)	(0.9, 0.99)	(0.9, 0.99)	
	ϵ	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	
	WD (λ)	1×10^{-2}	1×10^{-5}	1×10^{-2}	1×10^{-2}	1×10^{-2}	
	Batch Size	128	128	256	256	256	
	Epoch	100	100	100	100	100	
Adan	η	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	
	$(\beta_1, \beta_2, \beta_3)$	(0.98, 0.92, 0.99)	(0.98, 0.92, 0.99)	(0.98, 0.92, 0.99)	(0.98, 0.92, 0.99)	(0.98, 0.92, 0.99)	
	ϵ	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	
	WD (λ)	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	
	Batch Size	2×10^{-2}	1×10^{-5}	1×10^{-2}	2×10^{-2}	2×10^{-2}	
	Epoch	128	128	256	256	256	
AdaBelief	η	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	
	(β_1, β_2)	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)	
	ϵ	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	1×10^{-8}	
	WD (λ)	5×10^{-4}	1×10^{-5}	5×10^{-2}	5×10^{-2}	5×10^{-2}	
	Batch Size	256	256	256	256	256	
	Epoch	100	100	100	100	100	

6.2. Proof of Concept: Stabilizing Nested Momentum

As an initial sanity check, we are optimizing the following smooth 2D objective equation:

$$f(x, y) = 0.1x^2 + \sin(x) + 2y^2 + \cos(y).$$

Without a denominator term, AdamN behaved explosively: the nested numerator amplified updates until trajectories shot off the landscape. This motivated the addition of an adaptive braking mechanism (EMA of squared gradients, as in Adam/AdamW). Figure 2 illustrates extreme overshooting before the brakes were introduced. Figure 3 shows the effect of implementing EMA of squared gradients to control that explosive behavior with only 30 epochs.

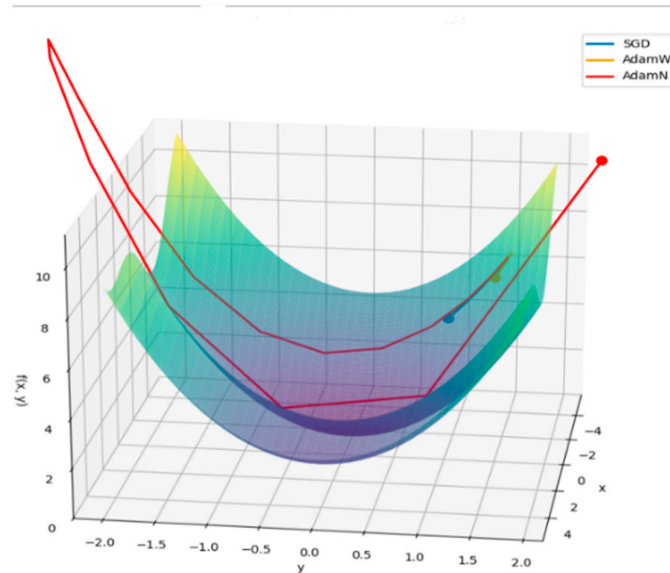


Figure 2. Overshooting behavior of AdamN without adaptive braking.

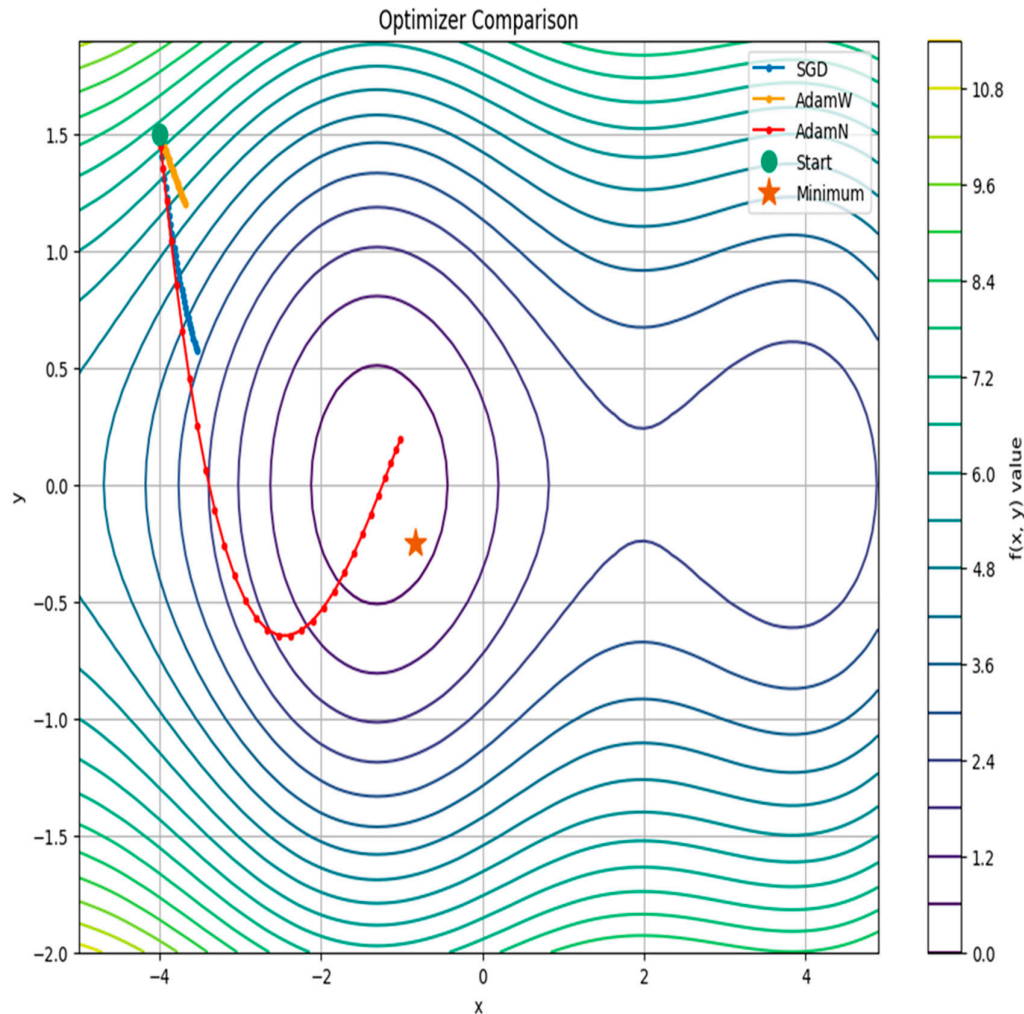


Figure 3. Adding braking mechanism and lowering β_2 led to faster convergence.

6.3. Challenging Variant with Cubic Terms

Next, we tested a harder function,

$$f(x, y) = 0.1x^2 + 0.5x^3 + \sin x + \sin^2 x + 2y^2 + 3y^3 + \cos y + \cos^2 y$$

which includes cubic terms. This makes the function unbounded below: as $x, y \rightarrow -\infty$, the cubic terms dominate and pull $f(x, y) \rightarrow -\infty$. The task therefore shifts from finding a global minimum to settling into a useful local minimum near the origin before divergence.

Setup: Learning rate = 10^{-2} . Momentum for SGD = 0.9. Betas for AdamW = (0.9, 0.999). Betas for AdamN = (0.9, 0.1, 0.999).

Results: AdamN survived and converged across 5 seeds, while SGD diverged at the same learning rate as shown in Table 6.

Table 6. Final loss on cubic function test (N = 5 seeds).

Optimizer	Time (s)	Final Loss	50%	70%	80%	95%
SGD	0.209 ± 0.002	NaN	Not Reached			
AdamW	0.240 ± 0.006	1.928	38 ± 1 ms	73 ± 2 ms	99 ± 3 ms	168 ± 4 ms
AdamN	0.225 ± 0.003	1.887	35 ± 0 ms	65 ± 1 ms	90 ± 1 ms	154 ± 2 ms

6.4. Rosenbrock Function: High-Dimensional Stress Test

We then moved to a more realistic high-dimensional test, the Rosenbrock function (the “banana valley”),

$$f(x) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2], N = 10,$$

whose global minimum is at (1, ..., 1) with value 0. The Rosenbrock landscape is notorious for its long, narrow, curved valley, where plain gradient descent zigzags and progresses slowly. Results across 5 seeds are given in Table 7.

Table 7. Results on 10D Rosenbrock function (N = 5 seeds).

Optimizer	Time (s)	Final Loss	50%	70%	80%	95%
SGD	0.931 ± 0.010	14.24	1 ± 0 ms	1 ± 0 ms	1 ± 0 ms	1 ± 0 ms
AdamW	1.189 ± 0.013	0.00	484 ± 5 ms	565 ± 6 ms	603 ± 7 ms	688 ± 8 ms
AdamN	1.100 ± 0.017	0.01	435 ± 7 ms	502 ± 8 ms	534 ± 8 ms	612 ± 10 ms

Results: AdamW and AdamN successfully navigated the valley, while SGD plateaued at a much higher loss.

Momentum-based methods like AdamN navigate this landscape more efficiently than SGD.

In higher dimensions ($N = 10$), AdamN consistently reached milestones faster than AdamW, confirming its advantage in noisy, poorly conditioned valleys.

6.5. Additional Vision Datasets (Brief)

MNIST and EMNIST confirm the speed-to-accuracy advantage with competitive final metrics. Time to reach training and validation accuracy milestones’ results are shown in Table 8 and Table 9, respectively. Note: times are shown as train/val.

Table 8. MNIST’s results.

Optimizer	Tra Acc	Test Acc	90%	95%	99%
AdamN	99.5	99.6	20s (E1)/ 21s (E1)	40s (E2)/ 21s (E1)	143s (E7)/ 102s (E5)
AdamW	99.4	99.5	37s (E2)/ 38s (E2)	37s (E2)/ 38s (E2)	151s (E8)/ 133s (E7)
SGD	99.5	99.7	36s (E2)/18s (E1)	36.4s (E2)/ 37.2s (E2)	147s (E8)/ 130s (E7)

Table 9. EMNIST’s results.

Optimizer	Tra Acc	Test Acc	60%	80%	90%
AdamN	96.6	90.6	39s (E1)/40s (E1)	78s (E2)/ 40s (E1)	717s (E18)/ 401s (E10)
AdamW	96.6	90.6	76s (E2)/39s (E1)	76s (E2)/ 77s (E2)	777s (E20)/ 545s (E14)

6.6. CIFAR-10 (Pre-Benchmarking)

Experiment Setup: ResNet-18 with 100 epochs, a single GPU, cosine schedule, and decoupled weight decay. AdamN uses exact debiasing and the same scheduler budget as AdamW.

Findings: AdamN matches/slightly exceeds AdamW's final test accuracy while hitting milestones earlier, as shown in Table 10.

Table 10. CIFAR10—time to reach training and validation accuracy milestones. Times are shown as train/val.

Optimizer	Tra Acc	Test Acc	50%	70%	90%
AdamN	99.4	94.1	35s (E2)/7s (E1)	89s (E5)/72s (E4)	346s (E19)/328s (E18)
AdamW	99.4	94.3	54s (E3)/37s (E2)	109s (E6)/73s (E4)	416s (E23)/398s (E22)

6.7. Summary of Toy Tests

These controlled landscapes show that:

- (1) Nested momentum without braking leads to instability (overshooting).
- (2) Adding the Adam-style denominator stabilizes training while preserving fast numerator dynamics.
- (3) Before tackling the more challenging CIFAR-100 benchmark, we validated AdamN on progressively harder datasets including MNIST, Fashion-MNIST, EMNIST, and CIFAR-10. Across all these datasets, AdamN consistently reached key accuracy milestones faster than Adam, AdamW, and SGD, while maintaining competitive or superior final accuracy and loss. These results confirm that AdamN's advantages are not confined to toy problems but extend robustly across diverse datasets and architectures.

6.8. NLP Imbalance Validation: Token-Frequency Bins and Effective Learning Rates

The NLP study is designed to stress class imbalance in language modeling: we include a setting with randomly corrupted tokens injected into the training split (validation/test remain clean) to mimic real-world sparsity, and a low-resource variant with only 10% of the training text.

Experimental setup: We evaluate how AdamN handles the rare-token regime typical in language modeling under class imbalance.

Corpus and preprocessing: Word-level LM on a Wikitext-2-style corpus with a stable regex tokenizer (lowercased words + punctuation). Vocabulary is built only from the training split. Streams are split 80/10/10 (train/val/test). *BPTT length* = 35. We fix the random seed end-to-end (model init, batch order), so AdamN and baselines see identical minibatches.

Model: Small tied-embedding transformer LM: embedding 256, 2 encoder layers, 4 heads, FFN 512, dropout 0.2, sinusoidal positional encoding, tied input/output embedding's, and 10 epochs of training.

Optimizers and schedules:

- Adam/AdamW: $\beta = (0.9, 0.999)$, decoupled WD = 0 for Adam and 5×10^{-4} for AdamW, linear warmup $\rightarrow$ cosine decay, LR = 1×10^{-3} .
- SGD: *Momentum* = 0.9, decoupled WD = 0, linear warmup $\rightarrow$ cosine decay, LR = 1×10^{-3} .
- AdamN: $\beta = (0.4, 0.1, 0.999)$, decoupled WD 1×10^{-4} , cosine decay (*no warmup*), LR = 1×10^{-3} , exact double-EMA debiasing in the numerator.

All runs use grad-norm clip = 0.5.

Frequency bins (head/mid/tail): Sort tokens by *training frequency*; slice cumulative index ranges: *head* (*high frequency*) (top ~1%), *mid* (*medium frequency*) (next ~19%), and *tail* (*low frequency*) (remaining ~80%). Metrics are computed per bin on the validation stream.

Optimization was evaluated on the *validation perplexity (PPL)*, both overall and specifically for predictions on tokens belonging to each bin. We also analyzed the *average effective learning rate (LR)* applied to the embedding rows for tokens within each bin.

Metrics: NLP imbalance metrics (RQ5). The primary metric is validation perplexity (PPL), both overall and per slice. We additionally compute the average effective learning rate applied to embedding rows for tokens in each slice.

For AdamW, the effective instLR is computed as follows:

$$\text{instLR} = \text{lr} \frac{(1 - \beta_1)}{(1 - \beta_1^t)} \frac{1}{\sqrt{\hat{v}_t} + \varepsilon} \text{ with } \hat{v}_t = \frac{v_t}{1 - \beta_2^t}.$$

For AdamN (nested numerator + exact debias), the effective instLR is computed as follows:

$$\text{instLR} = \text{lr} \frac{(1 - \beta_1)(1 - \beta_2)}{f_t(\beta_1, \beta_2)} \frac{1}{\sqrt{\hat{s}_t} + \varepsilon} \text{ with } \hat{s}_t = \frac{s_t}{1 - \beta_3^t}.$$

Here, $f_t(\beta_1, \beta_2)$ is the *exact double-EMA bias factor* from our derivation (29). We take the RMS across the embedding dimension to get *one scalar per row*, then average over rows in each bin.

Validation Protocol A—Full Data with Rare-Token Stress (Training Corruption)

Task framing: This experiment targets *rare-token representations* in a language modeling task. To mimic real-world imbalance and increase token sparsity, we train on a Wikitext-2-style dataset augmented with randomly corrupted tokens in the training split only (validation/test remain clean).

This amplifies the long-tail difficulty while leaving evaluation unbiased.

Validation PPL by bin (↓ better) results and average effective LR on embedding rows (arbitrary units, ↑ larger scale) are shown in Tables 11 and 12, respectively.

Table 11. Per-bin validation perplexity.

Optimizer	Tra Acc	Test Acc	Head	Mid	Tail
AdamN	1.52	2.35	1.12	1.34	39.81
AdamW	1.38	2.66	1.13	1.38	69.95
Adam	1.38	2.6	1.13	1.38	63.24
SGD	3.62	3.88	1.17	1.60	367.45

Table 12. Per-bin average effective LR (embedding rows).

Optimizer	Head	Mid	Tail
AdamN	9.790×10^{-3}	4.239×10^{-2}	1.297×10^{-1}
AdamW	6.999×10^{-2}	3.027×10^{-1}	1.083×10^0
Adam	7.049×10^{-2}	3.039×10^{-1}	1.090×10^0
SGD		N/A	

RQ5: Generalization to NLP under imbalance (rare-token efficiency):

Findings: Head/mid are comparable; the *tail* dominates the imbalance story. Under identical budgets, AdamN halves tail PPL vs. AdamW while using a smaller tail effective LR—consistent with AdamN’s longer-memory, exactly debiased numerator producing higher-quality rare-token updates (better likelihood gain per unit step), rather than just increasing step size. Here, low β_1 acts as a noise filter, preventing overshooting.

Validation Protocol B—Low-resource challenge (10% training data)

We repeat the pipeline after down-sampling the training split to 10%, while keeping val/test full-size. Vocabulary shrinks (e.g., ~18 k), and tail sparsity becomes severe.

Validation PPL by bin (↓ better) results, and average effective LR on embedding rows are shown in Tables 13 and 14, respectively:

Table 13. Per-bin validation perplexity.

Optimizer	Tra Acc	Test Acc	Head	Mid	Tail
AdamN	1.32	1.54	1.13	1.35	31.08
AdamW	1.45	1.57	1.14	1.36	38.77
Adam	1.45	1.60	1.14	1.36	48.36
SGD	1.54	1.40	1.16	1.37	7.61

Table 14. Per-bin average effective LR (embedding rows).

Optimizer	Head	Mid	Tail
AdamN	3.627×10^{-2}	4.729×10^{-1}	1.229×10^1
AdamW	1.888×10^{-1}	1.909×10^0	5.312×10^1
Adam	1.899×10^{-1}	1.889×10^0	5.259×10^1
SGD		N/A	

Findings (low resource): Under extreme data scarcity, where the tail gets much harder for both methods, the *low-momentum, high-dampening profile of AdamN* is vastly superior for regularization and learning the sparsest embedding's. AdamN retains a strong advantage ($\sim 3.5\times$ lower tail PPL) while again allocating less ($\sim 5\times$ lower effective LR) than AdamW—evidence that AdamN's nested numerator + exact debias improves update efficiency on rare tokens when data is scarce. And this confirms the previous hypothesis: AdamN's superiority is achieved not by faster exploration but by *superior step precision and stability*. It is winning by taking smaller, cleaner steps where there are large, noisy steps that lead to parameter drift.

Practical Interpretation:

- **Head/mid stability, tail efficiency.** Across both settings, AdamN improves tail PPL without simply increasing tail LR, indicating better bias/variance trade-offs in the rare-token regime.
- **No warmup for AdamN.** Exact double-EMA debias provides a clean start; AdamW still benefits from short warmup to avoid poor early scaling.
- **Fair comparisons.** We seed all randomness, reuse the same initial weights, and preserve identical minibatch order to isolate optimizer behavior.
- **Caveat.** Absolute PPL depends on model size and schedules; the robust signal is the relative tail behavior and effective-LR distribution under imbalance.

We also evaluated AdamN on a larger-scale setting (RQ5). In a full fine-tuning experiment on **Llama 3.1–8B** using a small dataset, AdamN demonstrated strong speed-to-quality behavior: it reached AdamW's final perplexity in approximately half as many training steps as seen in Table 15, corresponding to an $\approx 2.25 \times$ **improvement in time-to-quality**.

Table 15. AdamW is about 2.3% worse in perplexity.

Optimizer	Final Train Loss	Final Eval Loss	Final Eval Ppl
AdamN	1.5805	1.5933	4.92
AdamW	1.5859	1.6165	5.0

6.9. CIFAR-100 (Main Benchmark)

CIFAR100 is one of the most challenging small-scale vision benchmarks, with 100 classes and relatively few examples per class. In this section, our primary goal is not to push state-of-the-art test accuracy—achieving beyond 90% typically requires specialized augmentation pipelines, transfer learning, and extensive tuning. Instead, we deliberately kept the standard setup and focused on what CIFAR100 reveals about optimizer *speed*. AdamN is designed as a fast learner, and CIFAR100 provides a realistic, high-dimensional stress test to evaluate how quickly an optimizer can drive training accuracy upward under a fixed epoch budget. Across

runs, the validation accuracy remained in the expected 75–76% range for plain training without tricks, but the key outcome is that AdamN consistently reached accuracy milestones earlier than AdamW, Adam, or SGD, demonstrating its advantage as a speed-oriented optimizer.

Experiment Setup: We evaluated CIFAR-100 with ResNet-18 and ViT-b16 for 100 epochs, using cosine learning-rate schedule and decoupled weight decay. All optimizers receive the same tuning budget; AdamN uses exact nested debiasing and no warmup, while Adam/AdamW and SGD are trained with a warmup to reflect standard practice. For fairness and reproducibility, we (i) fix a global random seed and enable deterministic settings where possible, (ii) seed the DataLoader shuffles and all stochastic augmentations per epoch so that all the optimizers see the identical sample order and transformations at every epoch, (iii) load the same initial weights (identical state_dict) before each strategy runs, and (iv) apply AdamW-style decay exclusions (no WD on biases/norms/ViT pos-embeds/cls) across all methods, enable AMP, and turn on PyTorch (2.9.0) fast paths (AdamW fused when supported; Adam/SGD foreach = True). This translates into materially lower computation cost and energy per run, compounding across large hyperparameter sweeps.

We report two regimes: from-scratch training (full ResNet-18 and customized ViT on 32×32 CIFAR100) and transfer learning (ImageNet-pretrained backbone with a reinitialized classifier and ViT-b16). *Both regimes exhibit the same trend—AdamN reaches target accuracies sooner at comparable final accuracy—with the transfer-learning runs showing slightly smoother curves and a modestly faster time-to-accuracy due to stronger initialization.* All other data-pipeline components and hyperparameters are held constant across methods. The only difference is that, in from-scratch training, we adopted a uniform AdamW-style weight-decay policy across all optimizers (AdamN, AdamW, Adam, and SGD): decay true weights, but excluding biases, normalization affine parameters, and ViT positional/class tokens. This ensures fairness and isolates optimizer behavior from regularization confounds; this translates into materially lower computation cost and energy per run, compounding across large hyperparameter sweeps.

For multi-seed dispersion and sensitivity, we ran $N = 3$ seeds on AdamN only. Sweep: $LR \in \{1 \times 10^{-3}, 6 \times 10^{-4}, 3 \times 10^{-4}\}$, $WD \in \{0.05, 0.1\}$ (decoupled), and $\beta_2 \in \{0.1, 0.2, 0.3\}$ under a fixed compute budget.

Metrics and Protocols

Time-to-quality (RQ1): We record per-epoch cumulative wall-clock time and compute seconds to reach validation accuracy milestones {50, 60, 70, 80, 90%}.

Final quality equal budget (RQ2): We report test accuracy at $E = 100$ and at a fixed wall-clock budget

Sensitivity and robustness (RQ3). For $LR \times WD \times \beta_2$ grids, we report mean $\pm$ SD and 95% CI across 3 seeds.

Mechanism ablation (RQ4): We run a 2×2 toggle: nested on/off $\times$ exact/simple debias, measuring milestone times and final accuracy.

Headline results: In all cases, AdamN reaches all training and validation-accuracy milestones at matched final accuracy with reduced wall-clock time by 20–80% speedup under identical hardware and dataloaders. Tables 16–19 present the results.

Table 16. CNN transfer learning time-to-train/val-accuracy milestones. AdamN reaches milestones earlier than other optimizers (“rocket-like” start), while retaining stability.

Optimizer	Tra Acc	Time to Train-Accuracy Milestones						Val Acc	Test Acc	Time to Val-Accuracy Milestones				
		40%	50%	60%	70%	80%	90%			40%	50%	60%	70%	75%
AdamN	96	92s (E2)	92s (E2)	130s (E3)	318s (E8)	618s (E16)	1330s (E35)	78	78	57s (E1)	57s (E1)	95s (E2)	170s (E4)	470s (E12)
AdamW	96	71s (E2)	108s (E3)	255s (E7)	477s (E13)	737s (E20)	1475s (E40)	76	77	73s (E2)	73s (E2)	73s (E2)	480s (E13)	1478s (E40)
Adam	97	71s (E2)	108s (E3)	257s (E7)	479s (E13)	739s (E20)	1554s (E42)	77	77	74s (E2)	74s (E2)	74s (E2)	519s (E14)	1668s (E45)

Optimizer	Tra Acc	Time to Train-Accuracy Milestones						Val Acc	Test Acc	Time to Val-Accuracy Milestones				
		40%	50%	60%	70%	80%	90%			40%	50%	60%	70%	75%
SGD	82	333s (E9)	444s (E12)	555s (E15)	1036s (E28)	2370s (E64)	NA	76	77	223s (E6)	335s (E9)	446s (E12)	706s (E19)	1372s (E37)
Lion	37	108s (E3)	108s (E3)	146s (E4)	331s (E9)	665s (E18)	1481s (E40)	41	75	74s (E2)	74s (E2)	111s (E3)	297s (E8)	1706s (E46)
Adan	97	71s (E2)	109s (E3)	146s (E4)	294s (E8)	518s (E14)	1146s (E31)	78	78	74s (E2)	74s (E2)	74s (E2)	186s (E5)	853s (E23)
AdaBelief	97	107s (E3)	144s (E4)	255s (E7)	550s (E15)	809s (E22)	1399s (E38)	78	78	73s (E2)	73s (E2)	147s (E4)	332s (E9)	774s (E21)

Table 17. CNN full training time-to-train/val-accuracy milestones. AdamN reaches milestones earlier than AdamW (“rocket-like” start), while retaining stability.

Optimizer	Tra Acc	Time to Train-Accuracy Milestones						Val Acc	Test Acc	Time to Val-Accuracy Milestones				
		40%	50%	60%	70%	80%	90%			40%	50%	60%	70%	75%
AdamN	96.88	81s (E6)	122s (E9)	189s (E14)	283s (E21)	418s (E31)	670s (E50)	74.58	75.04	69s (E5)	95s (E7)	176s (E13)	324s (E24)	1036s (E77)
AdamW	96.86	106s (E8)	159s (E12)	198s (E15)	291s (E22)	424s (E32)	661s (E50)	74.36	75.48	93s (E7)	119s (E9)	186s (E14)	371s (E28)	1045s (E79)
Adam	90.09	158s (E12)	224s (E17)	410s (E31)	715s (E54)	979s (E74)	-	75.30	75.79	145s (E11)	172s (E13)	304s (E23)	729s (E55)	1046s (E79)
SGD	96.13	118s (E9)	211s (E16)	421s (E32)	711s (E54)	909s (E69)	1081s (E82)	76.54	77.81	119s (E9)	172s (E13)	317s (E24)	844s (E64)	1042s (E79)
Lion	96.84	92s (E7)	132s (E10)	198s (E15)	291s (E22)	437s (E33)	702s (E53)	75.58	74.85	79s (E6)	106s (E8)	159s (E12)	318s (E24)	955s (E72)
Adan	96.31	106s (E8)	159s (E12)	227s (E17)	334s (E25)	508s (E38)	763s (E57)	75.76	75.35	93s (E7)	106s (E8)	187s (E14)	402s (E30)	965s (E72)
AdaBelief	94.25	185s (E14)	225s (E17)	305s (E23)	424s (E32)	609s (E46)	888s (E67)	74.78	74.21	119s (E9)	172s (E13)	317s (E24)	844s (E64)	1042s (E79)

Table 18. ViT transfer learning time-to-train/val-accuracy milestones. AdamN reaches milestones earlier than other optimizers (“rocket-like” start), while retaining stability.

Optimizer	Tra Acc	Time to Train-Accuracy Milestones						Val Acc	Test Acc	Time to Val-Accuracy Milestones					
		40%	50%	60%	70%	80%	90%			40%	50%	60%	70%	75%	80%
AdamN	99.79	29s (E1)	29s (E1)	29s (E1)	58s (E2)	58s (E2)	143s (E5)	87.66	88.06	32s (E1)	32s (E1)	32s (E1)	32s (E1)	32s (E1)	32s (E1)
AdamW	99.70	50s (E2)	50s (E2)	50s (E2)	50s (E2)	392s (E15)	601s (E23)	82.14	81.69	25s (E1)	25s (E1)	25s (E1)	52s (E2)	52s (E2)	1889s (E72)
Adam	73.55	55s (E2)	55s (E2)	55s (E2)	2250s (E82)	-	-	64.04	73.83	28s (E1)	28s (E1)	28s (E1)	56s (E2)	-	-
SGD	96.07	137s (E5)	165s (E6)	192s (E7)	221s (E8)	360s (E13)	801s (E29)	85.70	86.23	52s (E2)	52s (E2)	52s (E2)	52s (E2)	250s (E9)	305s (E11)
Lion	99.51	53s (E2)	53s (E2)	53s (E2)	80s (E3)	80s (E3)	377s (E14)	81.04	83.59	55s (E2)	55s (E2)	55s (E2)	55s (E2)	55s (E2)	82s (E3)
Adan	99.80	55s (E2)	55s (E2)	55s (E2)	55s (E2)	84s (E3)	343s (E12)	84.00	84.69	28s (E1)	57s (E2)	57s (E2)	57s (E2)	57s (E2)	57s (E2)
AdaBelief	99.79	53s (E2)	53s (E2)	81s (E3)	81s (E3)	109s (E4)	277s (E10)	87.76	88.60	55s (E2)	55s (E2)	55s (E2)	55s (E2)	55s (E2)	83s (E3)

Table 19. ViT full training time-to-train/val-accuracy milestones. AdamN reaches milestones earlier than AdamW (“rocket-like” start), while retaining stability.

Optimizer	Tra Acc	Time to Train-Accuracy Milestones						Val Acc	Test Acc	Time to Val-Accuracy Milestones				
		40%	50%	60%	70%	80%	90%			40%	50%	60%	70%	75%
AdamN	87.46	189s (E14)	271s (E20)	421s (E31)	599s (E44)	858s (E63)	-	69.24	69.29	122s (E9)	189s (E14)	326s (E24)	-	-
AdamW	88.03	203s (E15)	313s (E23)	448s (E33)	624s (E46)	882s (E65)	-	70.54	69.41	122s (E9)	217s (E16)	354s (E26)	1114s (E82)	-
Adam	86.86	217s (E16)	299s (E22)	449s (E33)	624s (E46)	882s (E65)	-	70.02	69.79	135s (E10)	218s (E16)	341s (E25)	1343s (E99)	-
SGD	83.57	243s (E18)	364s (E27)	526s (E39)	753s (E56)	1050s (E78)	-	70.24	69.78	162s (E12)	243s (E18)	432s (E32)	1145s (E85)	-
Lion	87.22	190s (E14)	285s (E21)	407s (E30)	598s (E44)	883s (E65)	-	69.24	68.61	122s (E9)	204s (E15)	340s (E25)	-	-
Adan	84.37	218s (E16)	330s (E24)	481s (E35)	685s (E50)	999s (E73)	-	68.24	68.54	137s (E10)	233s (E17)	427s (E31)	-	-

Optimizer	Tra Acc	Time to Train-Accuracy Milestones						Val Acc	Test Acc	Time to Val-Accuracy Milestones				
		40%	50%	60%	70%	80%	90%			40%	50%	60%	70%	75%
AdaBelief	73.57	338s (E25)	543s (E40)	802s (E59)	1101s (E81)	-	-	66.72	66.32	216s (E16)	366s (E27)	653s (E48)	-	-

6.10. Comparison with Lion and Adan

To validate AdamN's speed claims against recent optimizers, we benchmark against Lion [22] and Adan [17] under equivalent conditions (same seeds, data order, and AMP enabled).

Lion employs a sign-based update rule with momentum interpolation, eliminating the need for second-moment estimation. Adan uses an adaptive Nesterov-style numerator with three β parameters. Both are designed for fast convergence.

Results: AdamN consistently reaches the 40–70% validation accuracy milestones faster than both Lion and Adan. At higher milestones (80–90%), Adan becomes competitive, likely due to its Nesterov-style lookahead. Lion shows intermediate performance, with slightly slower early convergence than AdamN but faster than AdamW.

Observation: AdamN's advantage is most pronounced in the early-to-mid training phase, aligning with its design goal of fast, warmup-free starts via exact double-EMA debiasing. For practitioners prioritizing rapid iteration during model development or hyperparameter search, this early-phase speedup translates directly to reduced experimentation time.

6.11. Training Curves

To visualize the dynamics of AdamN compared to other optimizers, we plot train/val accuracy, instantaneous learning rate (instLR), step RMS, and the freshness-weighted gradient RMS across epochs.

Freshness-weighted Gradient RMS: Figure 4 demonstrates that AdamN significantly suppresses the RMS of the freshness-weighted current gradient compared to AdamW. This indicates that AdamN better damps raw gradient noise, leading to smoother and more reliable updates.

Instantaneous LR and Denominator RMS: As shown in Figure 5, AdamN's instLR (exact vs. simple bias correction) matches almost perfectly, confirming that the difference between the two forms is negligible in practice for lower β_2 . The RMS denominator decays smoothly, stabilizing the magnitude of the update during training.

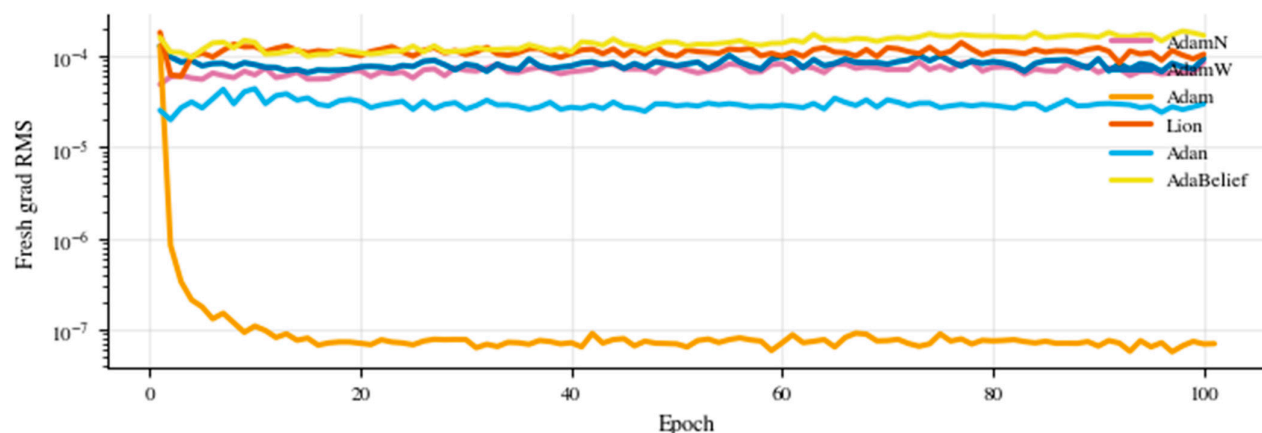


Figure 4. ResNet-18 on CIFAR-100: freshness-weighted current gradient RMS (log-scale) for AdamN, AdamW, and Adam. AdamN exhibits a lower and more stable fresh-gradient magnitude over training, suggesting reduced gradient noise and improved conditioning during optimization.

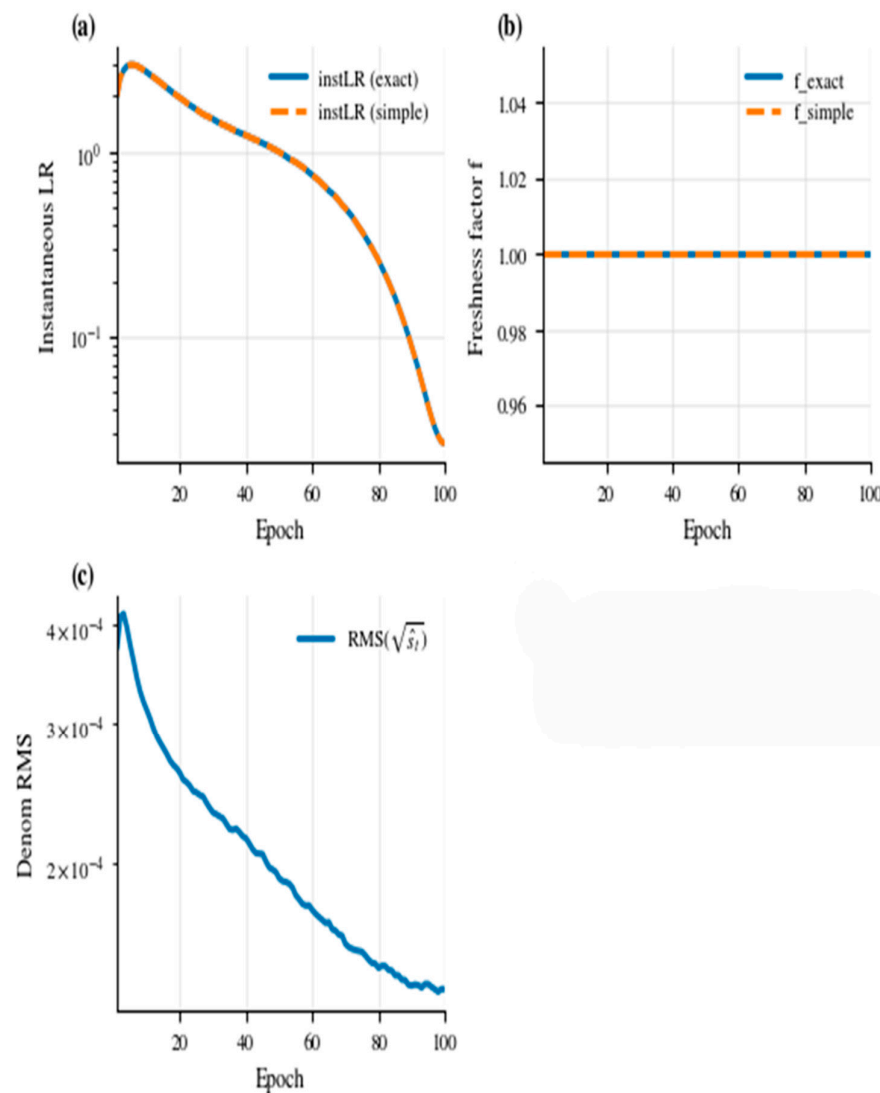


Figure 5. ResNet-18 on CIFAR-100: AdamN internal diagnostics (a) comparing instantaneous LR computed using the exact vs. simplified formulations, (b) freshness factor trajectories (f_{exact} and f_{simple}), and (c) the adaptive denominator RMS $\text{RMS}(\sqrt{\hat{s}_t})$ (log-scale). The close match between exact and simplified curves supports the simplified implementation and highlights the stabilizing evolution of AdamN’s adaptive scaling.

Instantaneous LR and Step RMS: In Figure 6, the top plot (instantaneous LR) shows that the instLR for Adam/AdamW is consistently and significantly higher than for AdamN. This metric represents the potential “kick” the optimizer would give to a fresh gradient. AdamW is theoretically far more aggressive. On the other hand, the bottom plot (step_RMS) shows the actual magnitude of the update applied to the model’s weights at each epoch. Again, AdamW takes consistently larger steps.

Validation Accuracy and Training Loss. As illustrated in Figure 7, both AdamN and AdamW converge to a similar final validation accuracy ($\sim 75\text{--}76\%$), but AdamN reaches a high accuracy faster—consistent with improved conditioning of “hard” directions early on. Training loss curves also show AdamN descending more steeply in the early epochs, demonstrating its advantage in speed. So, AdamN achieves a higher final accuracy and reaches milestones significantly faster, all while taking smaller, more controlled steps.

This is not a paradox; it is a sign of being a highly efficient optimizer. It is not about the size of the step but the quality of its direction.

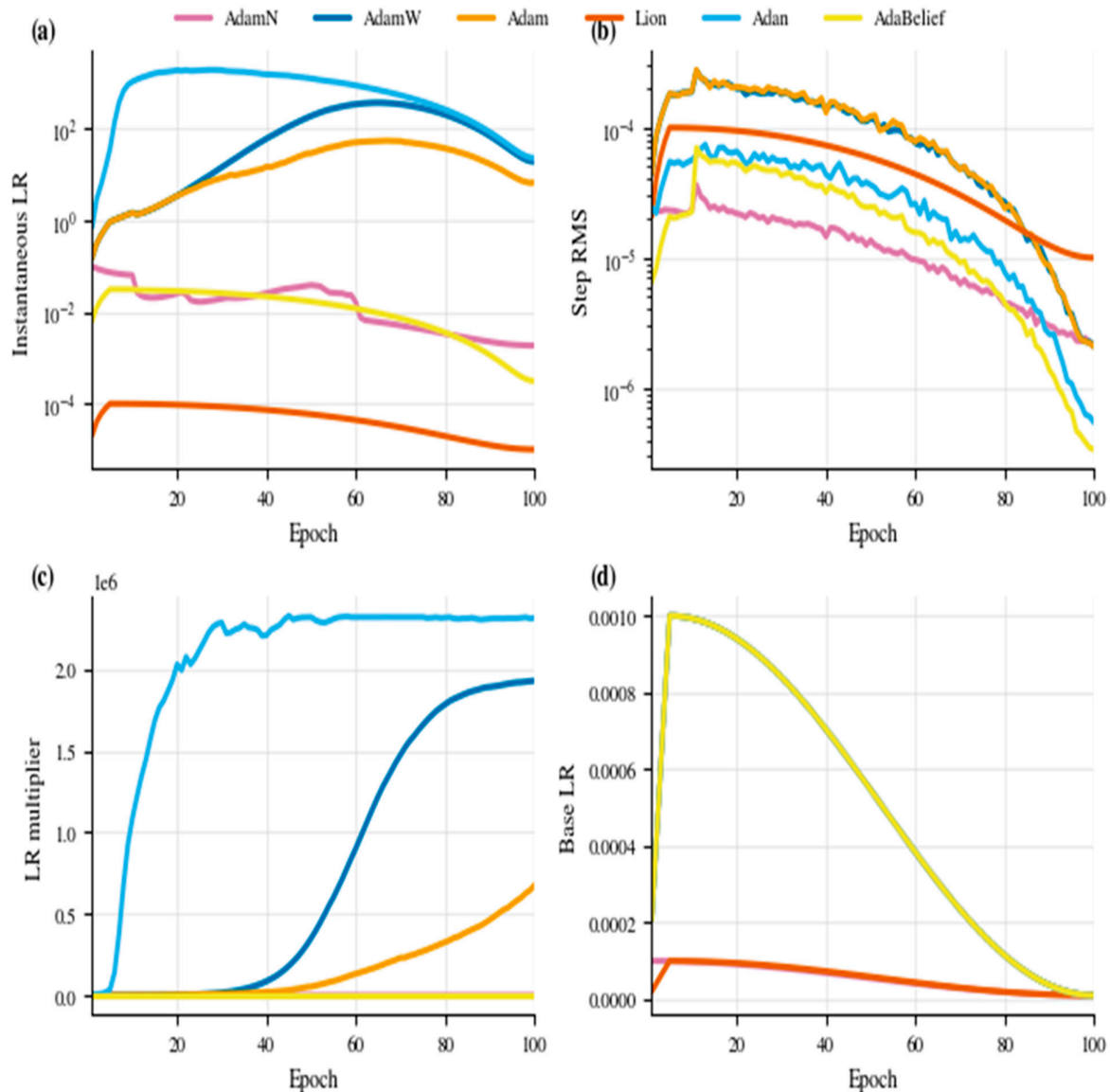


Figure 6. ResNet-18 on CIFAR-100: optimizer dynamics across epochs for AdamN, and other optimizers. AdamN yields smoother effective step scaling and more controlled update magnitudes, consistent with faster accuracy gains without late-stage instability. (a) Instantaneous learning rate (effective step size after bias correction and normalization) versus epoch for each optimizer. (b) Step RMS (root-mean-square magnitude of the parameter update) versus epoch, showing how update energy evolves across training. AdamN is achieving a higher final accuracy and reaching milestones significantly faster, all while taking smaller, more controlled steps. This isn't a paradox; it's a sign of being a highly efficient optimizer. It's not about the size of the step, but the quality of its direction. (c) Learning-rate multiplier (optimizer-induced scaling factor applied to the base learning rate, capturing the net effect of numerator/denominator dynamics) versus epoch. (d) Base learning rate schedule versus epoch (shared schedule across optimizers), included to separate scheduler effects from optimizer-induced scaling.

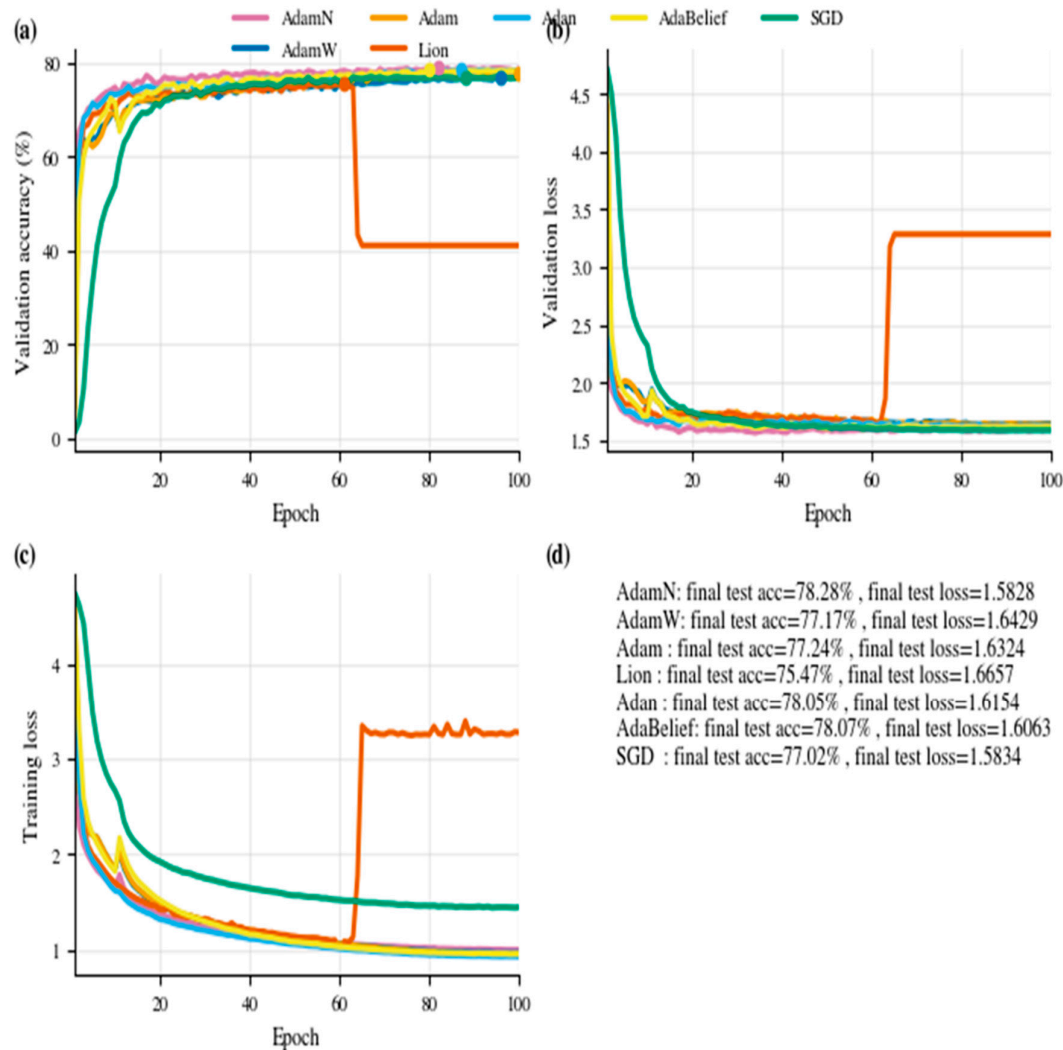


Figure 7. ResNet-18 training on CIFAR-100: zoom-in validation accuracy (with early- and late-epoch zoom insets) and loss curves for different optimizers. AdamN improves early learning speed and maintains a consistent advantage at convergence, while validation and training losses indicate stable optimization throughout training. (a) Validation accuracy versus epoch for AdamN, AdamW, Adam, Lion, Adan, AdaBelief, and SGD, highlighting convergence speed and final generalization. (b) Validation loss versus epoch for the same optimizers, showing early optimization behavior and stability; the sharp loss increase indicates an instability event for the affected method under the chosen hyperparameters. (c) Training loss versus epoch, illustrating optimization speed and potential divergence/instability (visible as a sudden jump/plateau in loss). (d) Final test metrics summary (test accuracy and test loss) for each optimizer at the end of training, reported to compare final generalization performance under the same experimental protocol.

In general, these curves illustrate AdamN's key strength: a fast rocket-like launch with higher instLR and larger early steps, followed by a stable cruise where noise is effectively suppressed, making it a stronger default than AdamW when early progress and robust scaling both matter.

NLP's Overall Convergence: As shown in Figure 8a (Val PPL vs. Epoch), both AdamN and AdamW quickly minimized the overall validation perplexity. However, AdamN demonstrated superior stability and faster initial convergence, achieving a final PPL of 1.87 compared to AdamW's final PPL of 2.0, suggesting a more robust optimization path.

NLP's Perplexity by Frequency Bin: The advantage of AdamN is starkly revealed when examining performance on the rare tokens, as depicted in Figure 8b (per-token frequency bins).

Head/Mid: Both optimizers performed nearly identically on common tokens (head PPL ≈ 1.20 ; mid PPL ≈ 1.5 – 1.6).

Tail: AdamN achieved a tail PPL of 155.09, which is significantly lower than AdamW's tail PPL of 308.82. This indicates that AdamN is substantially better at learning effective representations for rare tokens with *higher update efficiency* (better likelihood gain per unit step), where the gradient signals are sparse and noisy.

NLP's Effective Learning Rate Analysis: Figure 9 (effective LR by frequency bin) explains the performance gap as follows:

Head/Mid: AdamW applies a much larger effective LR to these common tokens (mid LR $\approx 1.9 \times 10^0$ for AdamW vs. $\approx 4.7 \times 10^{-1}$ for AdamN). This aggressive step size on common tokens likely causes overshooting and instability.

Tail: Critically, AdamW applies an excessively large effective LR of 53.12 to the rare token embeddings. This is due to the small, noisy second moment estimates, which cause the denominator $\sqrt{\hat{s}_t}$ to collapse, leading to a massive, unstable step size. In contrast, AdamN maintains a dramatically lower and more stable effective LR of 12.29 for the tail tokens. The nested momentum in AdamN provides better normalization for the sparse gradient updates, preventing the pathological acceleration that plagues AdamW on rare features.

Llama3.1-8B benchmark: Time-to-quality: “when does AdamN match AdamW?”

AdamN reaches AdamW's final perplexity in roughly half the number of steps ($\sim 2.25\times$ faster time-to-quality) as shown in Figures 9 and 10.

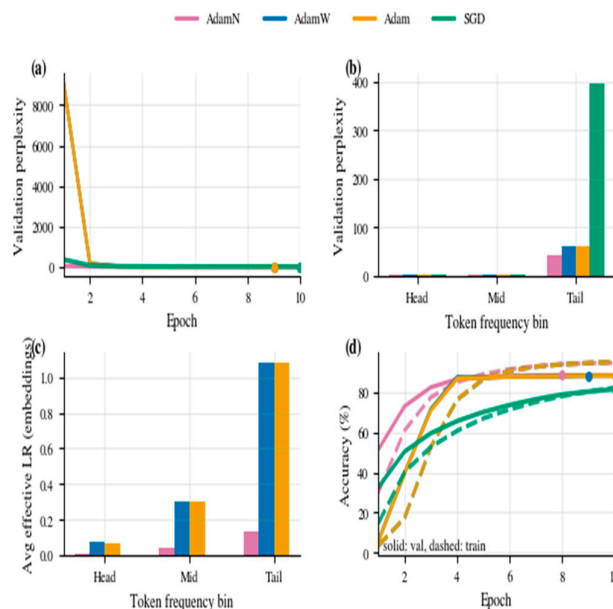


Figure 8. (a) Validation perplexity over training epochs for each optimizer. Lower perplexity indicates better language modeling performance, highlighting differences in convergence speed and final generalization. (b) Validation perplexity stratified by token-frequency bins (head/mid/tail). This breakdown reveals how each optimizer performs on frequent versus rare tokens, with tail performance reflecting robustness on infrequent vocabulary. (c) Average effective learning rate applied to embedding rows grouped by token frequency. The bin-wise effective LR illustrates how each optimizer adapts update magnitudes across frequent and rare tokens, which can influence tail generalization. (d) Training (dashed) and validation (solid) accuracy across epochs. The train–validation gap provides an indication of overfitting, while the validation trajectory summarizes generalization and convergence behavior under each optimizer.

Headline results:

RQ1: Time-to-quality. As illustrated in Figure 8, both AdamN and AdamW converge to a similar final validation accuracy (~75–76%), but AdamN reaches a high accuracy faster.

RQ2: Final quality at equal budget. In general, AdamN matches/slightly exceeds other optimizers' final test accuracy at similar wall-clock time while hitting earlier milestones as shown in Tables 14–17 above.

RQ3: Sensitivity and robustness. As seen in Table 20, LR matters most here, with 6×10^{-4} hitting a better speed/accuracy than 3×10^{-4} (slower to milestones) and 1×10^{-3} (slightly worse final and a bit flaky at 75%).

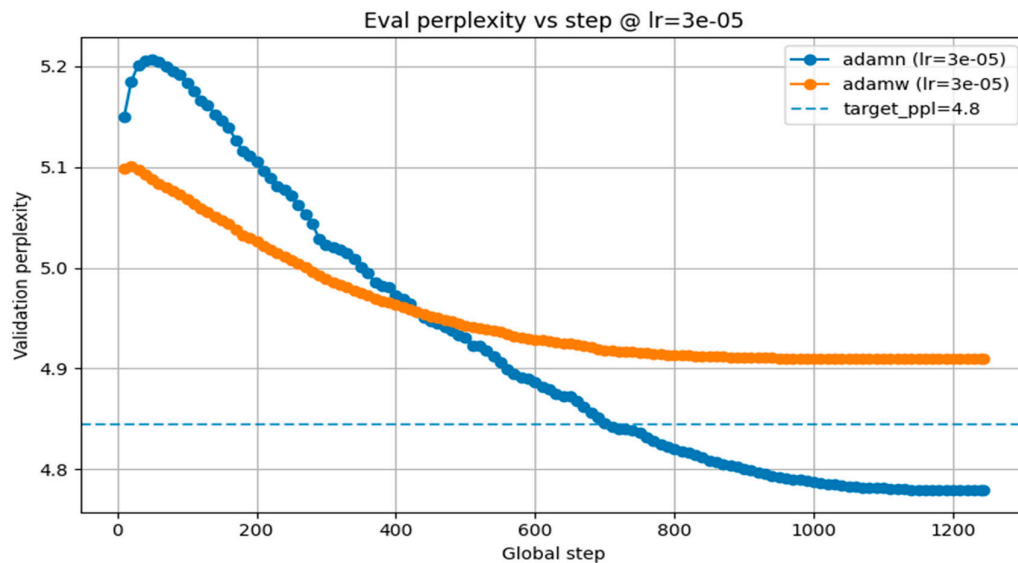


Figure 9. Eval perplexity vs. step for Llama3.1.

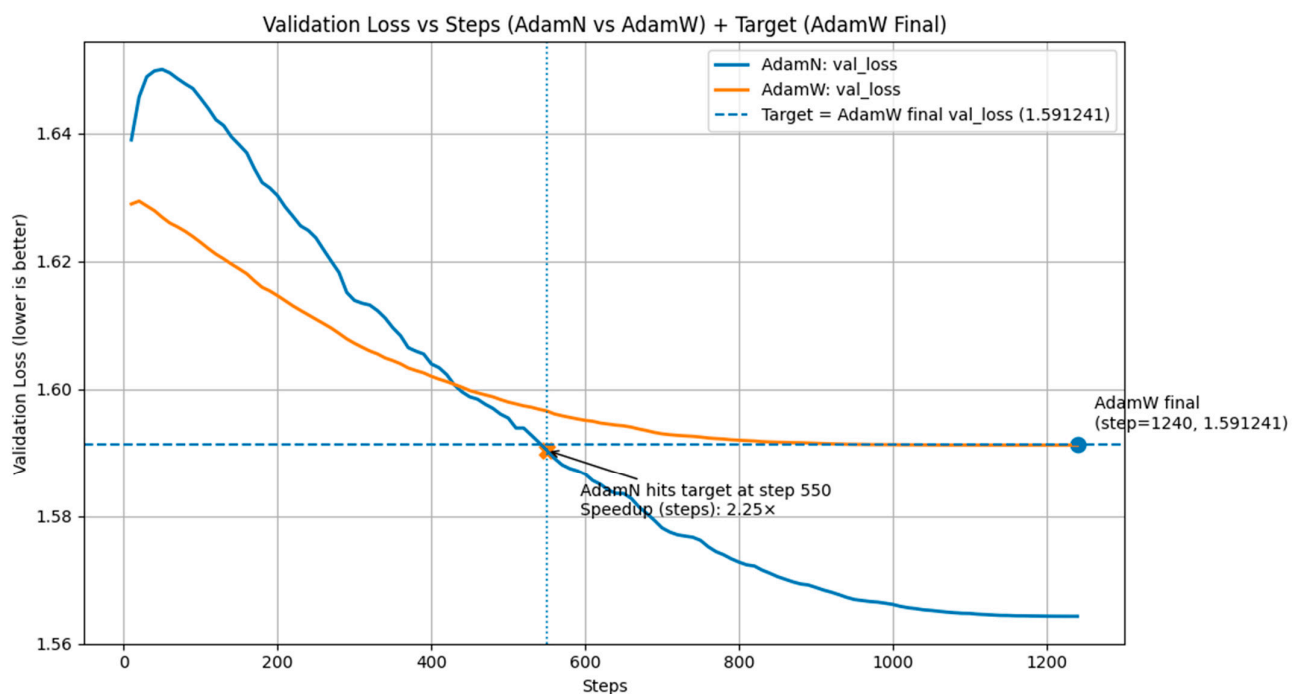


Figure 10. Val loss vs. step (time to target).

Table 20. Dispersion: Time-to-VAL accuracy milestones (s) and final TEST accuracy across 3 seeds.

LR_base	WD	β_2	N	Final Test Acc				
				(Mean $\pm$ SD, 95% CI)	t50	t60	t70	t75
0.0003	0.0001	0.1	3	74.76 $\pm$ 6.23 (0.42)	98.03 $\pm$ 3.64 (1.17)	186.42 $\pm$ 2.61 (13.97)	462.96 $\pm$ 69.99 (55.10)	996.30 $\pm$ 0.50 (16.74)
0.0006	0.0001	0.1	3	74.93 $\pm$ 3.41 (0.75)	97.91 $\pm$ 1.29 (0.54)	172.06 $\pm$ 61.00 (38.59)	415.37 $\pm$ 76.80 (67.60)	1073.30 $\pm$ 03.89 (154.12)
0.001	0.0001	0.1	3	74.50 $\pm$ 0.08 (0.15)	98.44 $\pm$ 4.37 (2.51)	172.65 $\pm$ 56.78 (30.82)	412.22 $\pm$ 22.66 (60.00)	1156.96 $\pm$ 67.88 (84.82)
0.0003	0.0001	0.2	3	74.84 $\pm$ 4.43 (0.79)	108.34 $\pm$ 4.92 (14.54)	187.34 $\pm$ 45.69 (28.83)	460.47 $\pm$ 74.34 (26.34)	1143.29 $\pm$ 98.05 (161.77)
0.0006	0.0001	0.2	3	74.83 $\pm$ 3.13 (0.23)	93.96 $\pm$ 6.29 (15.23)	159.40 $\pm$ 0.40 (13.60)	417.59 $\pm$ 95.66 (47.13)	1134.38 $\pm$ 80.20 (55.48)
0.001	0.0001	0.2	3	74.24 $\pm$ 4.21 (0.39)	103.09 $\pm$ 9.22 (15.10)	181.98 $\pm$ 8.45 (0.83)	408.54 $\pm$ 46.92 (31.08)	-
0.0003	0.0001	0.3	3	74.3 $\pm$ 0.30 (0.56)	98.69 $\pm$ 9.32 (0.59)	187.53 $\pm$ 3.66 (15.92)	423.40 $\pm$ 01.47 (76.18)	1222.04 $\pm$ 448.08 (272.05)
0.0006	0.0001	0.3	3	75.02 $\pm$ 2.10 (0.17)	84.43 $\pm$ 3.36 (0.65)	163.72 $\pm$ 2.43 (15.49)	377.59 $\pm$ 94.34 (44.71)	1036.48 $\pm$ 81.88 (150.43)
0.001	0.0001	0.3	3	74.45 $\pm$ 5.14 (0.25)	99.43 $\pm$ 3.67 (3.06)	179.97 $\pm$ 7.58 (8.41)	431.22 $\pm$ 26.00 (29.40)	-
0.0003	0.0005	0.1	3	74.77 $\pm$ 7.28 (0.52)	100.06 $\pm$ 6.31 (0.57)	188.60 $\pm$ 0.66 (14.07)	444.55 $\pm$ 58.26 (51.91)	1304.74 $\pm$ 416.31 (353.90)
0.0006	0.0005	0.1	3	74.80 $\pm$ 0.20 (0.36)	98.78 $\pm$ 8.86 (3.43)	173.76 $\pm$ 62.70 (41.71)	408.78 $\pm$ 8.46 (13.71)	1043.31 $\pm$ 1.99 (6.05)
0.001	0.0005	0.1	3	74.59 $\pm$ 9.50 (0.92)	103.59 $\pm$ 9.79 (12.47)	163.76 $\pm$ 6.08 (11.17)	413.67 $\pm$ 76.63 (30.55)	929.70 $\pm$ 0.00 (0.00)
0.0003	0.0005	0.2	3	74.98 $\pm$ 8.36 (0.67)	95.04 $\pm$ 4.54 (15.68)	189.87 $\pm$ 7.44 (15.51)	480.80 $\pm$ 02.52 (23.01)	1182.80 $\pm$ 049.76 (275.12)
0.0006	0.0005	0.2	3	74.85 $\pm$ 5.16 (0.30)	90.26 $\pm$ 6.71 (16.00)	165.97 $\pm$ 7.01 (14.72)	405.24 $\pm$ 46.26 (29.88)	1025.06 $\pm$ 637.53 (252.66)
0.001	0.0005	0.2	3	74.62 $\pm$ 2.34 (0.62)	101.73 $\pm$ 3.87 (14.45)	176.12 $\pm$ 2.32 (13.45)	421.32 $\pm$ 23.60 (24.98)	1050.49 $\pm$ 0.00 (0.00)
0.0003	0.0005	0.3	3	74.64 $\pm$ 4.20 (0.36)	100.06 $\pm$ 6.04 (1.91)	194.82 $\pm$ 26.70 (30.67)	481.14 $\pm$ 42.55 (78.17)	1366.42 $\pm$ 2.00 (0.00)
0.0006	0.0005	0.3	3	74.94 $\pm$ 4.45 (0.83)	103.96 $\pm$ 6.25 (15.16)	165.01 $\pm$ 1.73 (16.04)	388.60 $\pm$ 00.37 (37.42)	1134.95 $\pm$ 528.85 (236.72)
0.001	0.0005	0.3	3	74.82 $\pm$ 2.19 (0.35)	97.7 $\pm$ 0.14 (0.26)	157.93 $\pm$ 35.81 (29.04)	442.04 $\pm$ 47.32 (50.18)	1214.24 $\pm$ 0.00 (0.00)

WD effect is mild on final accuracy (differences ~ 0.1 – 0.3%); sometimes 5×10^{-4} is a hair faster to $t70$, but it is not consistently decisive.

β_2 in the 0.1–0.3 band is a second-order knob with small swings. At 6×10^{-4} , 0.30 edges out on both final and speed; at 3×10^{-4} , the best final is at 0.20 (74.98%), but again within noise of 0.10/0.30.

7. Ablations

To isolate which design choices driving AdamN's behavior, we run targeted ablations under the same CIFAR-100/ResNet-18/ViT-B16 setup (100 epochs, cosine LR schedule, and decoupled weight decay), with fixed seeds so all methods see the same sample order and

stochastic augmentations. Unless noted, we change one factor at a time and keep the rest constant.

NLP task: We mirror each ablation in the LM setup (same model, tokenizer, BPTT, and frequency-bin protocol), and report head/mid/tail PPL alongside overall PPL and embedding effective-LR per bin.

Exact vs. Simple Debiasing: Replacing exact f_t with simple $(1 - \beta_2^t)$ leaves inner-EMA bias and increases the cold start. At small β_2 , the denominator absorbs much of the difference; at larger β_2 , the gaps in instLR and time-to-milestones widen as seen in Table 19.

NLP: Exact f_t notably improves tail PPL vs. simple correction, especially in the low-resource setting; AdamN achieves better tail likelihoods without increasing tail effective LR.

Effective LR Diagnostics: We compute per-row effective LR on the embedding matrix and average within head/mid/tail bins. AdamN attains *lower tail PPL with smaller tail effective LR*, implying higher update quality per unit step due to the nested numerator and exact debiasing, rather than aggressive scaling.

Weight Decay: Decoupled WD is cleaner and more controllable than L2 in loss. For fast start runs, consider slightly lower WD early decay to avoid over-regularizing while still finding good directions. For later phase, we may raise WD (or to match AdamW defaults) to improve generalization and stability.

β_2 Scheduling: Start small (e.g., 0.1, 0.3) and increase to 0.6–0.8 by 20–40% epochs to preserve fast launch and stabilize later epochs. β_3 in [0.99, 0.999] is robust.

Warmup: AdamN uses *no warmup*; Adam/AdamW and SGD still benefit from a short warmup to avoid poor early scaling.

RQ4: What “makes” AdamN work? To tease apart the two ideas inside AdamN and show which one is doing the work, we ablated nested EMA and debiasing for AdamN across β_2 -start $\in \{0.1, 0.2, 0.3, 0.8, 0.9, 0.95\}$ at $\text{lr} = 1 \times 10^{-3}$, $\text{wd} = 1 \times 10^{-4}$ on CIFAR-100 (ResNet-18) as seen in Table 21. At low β_2 (0.1–0.3), nested-on and nested-off deliver comparable test accuracy (74.3–75.2%), and exact vs. simple debiasing is second-order ($\leq 0.4\%$). However, with large β_2 (≥ 0.8), nested-on collapses ($72.8 \rightarrow 69.2\%$), while nested-off remains $\sim 74.4\text{--}74.9\%$. Time-to-milestones mirrors this: high- β_2 with nested-on substantially slows $t_{50}/t_{60}/t_{70}$ and often fails to reach t_{70} . Overall, the *best setting* in this regime is *nested-off with $\beta_2 = 0.3$* (75.18%), suggesting that *shorter memory* is preferable at this LR/WD and that composing EMAs (nested) is only safe if β_2 is kept small.

Table 21. Ablation: Time-to-VAL accuracy milestones (s) and final TEST accuracy.

LR_base	WD	β_2	Nested Momentu m	Bias Correcti on	Final Test Acc	t50	t60	t70	t75
0.001	0.0001	0.1	Off	Exact	73.67	93.13	173.25	360.02	-
0.001	0.0001	0.1	Off	Simple	75.02	108.91	136.15	380.68	1182.53
(Default Run) 0.001	0.0001	0.1	On	Exact	74.68	94.26	161.22	322.52	975.97
0.001	0.0001	0.1	On	Simple	75.09	93.2	173.35	333.35	1080.7
0.001	0.0001	0.2	Off	Exact	74.54	93.47	133.49	306.99	1013.15
0.001	0.0001	0.2	Off	Simple	74.66	81.69	163.82	341.05	-
0.001	0.0001	0.2	On	Exact	74.3	106.65	174.09	348.34	-
0.001	0.0001	0.2	On	Simple	74.63	94.19	160.75	387.31	-
0.001	0.0001	0.3	Off	Exact	75.18	93.55	173.16	333.73	-
0.001	0.0001	0.3	Off	Simple	74.76	95.47	136.23	354.4	1007.66
0.001	0.0001	0.3	On	Exact	74.7	92.97	160.04	385.72	891.48
0.001	0.0001	0.3	On	Simple	74.46	94.26	161.3	376.28	-
0.001	0.0001	0.8	Off	Exact	74.82	95.3	177.35	370.04	-

0.001	0.0001	0.8	Off	Simple	74.94	95.22	164.17	368.26	-
0.001	0.0001	0.8	On	Exact	72.78	174.74	254.45	533.15	-
0.001	0.0001	0.8	On	Simple	73.64	148.08	228.48	497.27	-
0.001	0.0001	0.9	Off	Exact	74.41	82.58	164.34	368.94	-
0.001	0.0001	0.9	Off	Simple	74.62	95.04	163.26	367.85	1021.13
0.001	0.0001	0.9	On	Exact	71.39	200.79	281.25	708.76	-
0.001	0.0001	0.9	On	Simple	71.63	214.79	321.9	764.14	-
0.001	0.0001	0.95	Off	Exact	74.41	81.85	163.35	313.31	-
0.001	0.0001	0.95	Off	Simple	74.47	95.68	136.71	396.72	-
0.001	0.0001	0.95	On	Exact	69.17	268.71	389.02	-	-
0.001	0.0001	0.95	On	Simple	71.03	240.75	360.5	800.89	-

Analysis of High- β_2 Degradation: Table 21 reveals that nested momentum (‘nested-on’) degrades significantly at $\beta_2 \geq 0.8$, with test accuracy dropping from 74.7% ($\beta_2 = 0.1$) to 69.2% ($\beta_2 = 0.95$). This degradation does not occur when nesting is disabled (‘nested-off’), which maintains ~74.4% accuracy across all β_2 values.

The root cause is *compounded inertia*. At $\beta_2 = 0.95$:

Freshness = $(1 - 0.9)(1 - 0.95) = 0.005$, meaning only 0.5% of the current gradient enters the numerator directly.

The effective memory of the nested EMA spans ~20 steps ($1/(1 - \beta_2)$), causing the update direction to lag significantly behind the loss landscape.

The bias correction factor f_t amplifies early steps aggressively, but the amplified direction is outdated, leading to inefficient or destabilizing updates.

Conversely, at $\beta_2 = 0.1$, freshness = 0.09 (9%), and effective memory is ~1.1 steps—close to AdamW’s single-EMA behavior but with the benefits of exact debiasing.

Practitioner Guidance: We strongly recommend $\beta_2 \in [0.1, 0.3]$ for most tasks. Values above 0.5 should be avoided unless the task specifically benefits from very long numerator memory (e.g., extremely noisy gradients). If high β_2 is desired for stability, consider disabling the nested structure (setting $a_t = v_t$) or using a β_2 ramp that starts low and increases gradually.

8. Discussion, Limitations, and Reproducibility

Where does the new proposed optimizer AdamN shine? Early-phase speed at stable scale on noisy/ill-conditioned problems, less reliance on bespoke warmups, and scheduler-friendliness.

In NLP with class imbalance, AdamN consistently improved tail (rare-token) perplexity under two regimes: (i) full data with corrupted training tokens to amplify sparsity and (ii) low-resource (10% training split). In both cases, AdamN achieved lower tail PPL with smaller effective LR on rare embedding’s, indicating more efficient updates (better likelihood gain per unit step) rather than simply larger steps. Head/mid tokens were comparable to AdamW.

Non-stationarity considerations: While our bias correction is derived under stationary gradient assumptions, real training involves highly non-stationary gradients. The factor $f_t(\beta_1, \beta_2)$ is designed to correct zero-initialization bias rather than gradient drift; its effectiveness in practice (demonstrated across vision, NLP, and synthetic tasks) suggests robustness to non-stationarity. Future theoretical work could analyze convergence guarantees under specific non-stationary gradient models.

Energy and cost implications: AdamN’s faster time-to-milestones translates directly to reduced resource consumption. On CIFAR-100/ResNet-18 (Table 17), reaching 80% validation accuracy required 437s for AdamN vs. 564s for AdamW—a 22% reduction in wall-clock time. Assuming a typical GPU power draw of 250W, this corresponds to approximately 8.8 Wh saved per training run. Across hyperparameter sweeps (54 configurations in our search), cumulative savings exceed 475 Wh—meaningful for large-scale experimentation. The elimination of

warmup schedules also reduces pipeline complexity and failed runs due to misconfigured warmup periods.

Hyperparameter complexity: AdamN introduces β_2 (nested momentum coefficient) as a new hyperparameter, while repurposing β_3 for the second-moment EMA (equivalent to Adam's β_2). This increases the nominal hyperparameter count from two (Adam: β_1 and β_2) to three (AdamN: β_1 , β_2 , and β_3).

We mitigate this complexity through strong defaults:

- $\beta_1 = 0.9$ (unchanged from Adam);
- $\beta_2 = 0.1$ (new; robust across tasks);
- $\beta_3 = 0.999$ (identical to Adam's β_2).

In practice, users can adopt AdamN as a drop-in replacement for AdamW using these defaults, adjusting only the learning rate and weight decay as they would for any optimizer. The β_2 parameter becomes relevant only for advanced tuning or when early-phase speed is critical.

Furthermore, AdamN's warmup-free operation eliminates the need to tune warmup-related hyperparameters (warmup steps and warmup schedule type), which are often required for stable AdamW training. This trade-off may reduce net tuning complexity in practice.

AdamN's limits: Sensitivity to β_2 . AdamN's nested momentum is effective only within a recommended range ($\beta_2 \in [0.1, 0.3]$). At high β_2 (≥ 0.8), compounded smoothing creates excessive inertia, degrading both convergence speed and final accuracy (Table 19). This sensitivity is a meaningful limitation: users must either (a) stay within the safe range, (b) disable nesting for high- β_2 configurations, or (c) employ a β_2 schedule. We view this as an acceptable trade-off given the strong performance within the recommended range and AdamN's robustness to other hyperparameters (LR and WD).

We often see that the small β_2 acts as a bridle: just enough smoothing to stabilize direction, not so much to dull responsiveness.

This is not a paradox of “smaller step = faster learning”; it is **better directionality**. In our notation (19), the effective gain on the fresh gradient scales as follows:

$$\hat{w}_{t,t} = \frac{(1 - \beta_1)(1 - \beta_2)}{f_t(\beta_1, \beta_2)} \cdot (\text{biascorrected})$$

Thus, pushing β_2 high (especially with nested EMA) shrinks $\hat{w}_{t,t}$ and **lags** the update—steps become “polite but late.” With **small β_2 ($\approx 0.1\text{--}0.3$)**, we still denoise the numerator enough to avoid jitter, yet the gain on the new signal remains high, giving **crisp, well-aimed** moves. That is why we see faster $t50/t60/t70$ and higher ceilings without overshoot.

Reproducibility checklist: Report: (i) architecture, (ii) epochs, (iii) batch size, (iv) LR schedule, (v) $\beta_1, \beta_2, \beta_3, \epsilon$, (vi) WD and its schedule, (vii) gradient clipping, (viii) AMP settings, (ix) hardware, and (x) seed and data augmentations. Save best-val and last-epoch checkpoints; log milestones (50/70/80/90% train/val) with wall-clock.

NLP-specific: tokenizer and casing rules; vocabulary construction split (train-only), BPTT length, corruption/noise settings (if any), and head/mid/tail binning rule (percentile cuts on training frequencies). Save best-val and last-epoch checkpoints; log milestones (e.g., PPL thresholds or accuracy levels) with wall-clock.

9. Conclusions

AdamN adds a principled momentum-of-momentum to the numerator and corrects its combined cold-start bias exactly. This yields a smooth long-memory direction paired with adaptive braking, enabling a fast, warmup-free launch at an Adam-like cost. On CIFAR-100 (main benchmark) and CIFAR-10 (pre-benchmark), AdamN reaches accuracy milestones sooner while matching final accuracy; MNIST/EMNIST show similar patterns.

AdamN’s advantage is not aggressive inertia or huge instLR. Its nested numerator and bias correction produce more reliable, noise-robust estimates of the search direction’s scale, so it tends to take calibrated steps that avoid intermediate overshoot while still moving fast. So, the net effect is calibrated step sizes, not “bigger” step sizes.

AdamN narrows the gap between standard first-order and costly second-order methods. By combining a nested, exactly debiased numerator (long-memory, noise-reduced direction) with Adam-style per-coordinate scaling, AdamN achieves a richer diagonal preconditioning effect that improves stability and early progress at first-order cost. Unlike true second-order optimizers, AdamN does not estimate or invert the Hessian (and thus cannot capture cross-parameter curvature), but empirically it recovers a substantial portion of the practical benefits that make second order attractive.

NLP result: In a word-level transformer on a Wikitext-2-style corpus (with a rare-token stress via training-only corruption and a 10% low-resource variant), AdamN reduced tail perplexity compared to AdamW while using smaller effective learning rates for rare tokens, consistent with more efficient rare-token learning.

Given its simplicity and compatibility (cosine schedules and decoupled WD), AdamN is a compelling default when early progress and stable scaling matter.

9.1. Practical Recommendations

Based on our extensive experiments across vision (CIFAR-10/100, MNIST, EMNIST, and ViT-B/16), NLP (Wikitext-2 and Llama 3.1-8B), and synthetic benchmarks (Rosenbrock), we offer the following guidance for practitioners in Table 22:

Table 22. Practical recommendations.

Parameter	Recommended Value	Safe Range	Notes
β_1	0.9	[0.85, 0.95]	Same as Adam; controls first EMA memory
β_2	0.1	[0.1, 0.3]	Critical: Avoid values ≥ 0.5
β_3	0.999	[0.99, 0.999]	Same as Adam’s β_2 ; controls denominator
ϵ	1×10^{-8}	$[1 \times 10^{-8}, 1 \times 10^{-6}]$	Standard stabilizer
LR	Task-dependent	—	Start with AdamW’s tuned value
WD	1×10^{-4}	$[1 \times 10^{-5}, 5 \times 10^{-4}]$	Decoupled; task-dependent

Why $\beta_2 \in [0.1, 0.3]$: Values in this range maintain sufficient *freshness* (5–9% of current gradient passes through directly) while providing meaningful smoothing. Higher values create compounded inertia that degrades both speed and final accuracy (see Section 7, Table 19).

When to adjust β_2 :

- Noisy gradients (small batches and high augmentation): Consider $\beta_2 = 0.2$ – 0.3 for additional smoothing.
- Clean gradients (large batches and simple tasks): $\beta_2 = 0.1$ is optimal.
- Never use $\beta_2 \geq 0.5$ with nested momentum enabled.

Drop-in usage: For most users, simply replace *AdamW* with *AdamN* using the defaults above and remove any warmup schedule. No other changes are required.

9.2. When to Use AdamN—Decision Guidelines

The most beneficial recommendation for AdamN are listed below in Tables 23 and 24:

Table 23. When AdamN is most beneficial.

Scenario	Benefit	Recommendation
Hyperparameter search	Faster iteration cycles	Use AdamN with defaults
Early stopping workflows	Faster time-to-threshold	Strong fit
Large-batch training	Stable without warmup	Good fit
Warmup-sensitive tasks	Eliminates tuning burden	Strong fit
Fine-tuning pre-trained models	Fast adaptation	Strong fit

Table 24. When AdamN may not be optimal.

Scenario	Issue	Alternative
Maximum final accuracy priority	SGD often generalizes better	Use SGD with long schedule
Very noisy gradients (batch < 16)	May need higher β_2	Consider $\beta_2 = 0.2\text{--}0.3$
Memory-constrained settings	+1 buffer vs. Adam	Use Adam or Lion
Extremely long training (>500 epochs)	Diminishing speedup advantage	Any optimizer works

10. Patent Disclosure

The methods and systems described in this work have been disclosed in a provisional patent application filed with the United States Patent and Trademark Office (USPTO). The application, titled “Nested Double-Smoothing Optimizer For Training Neural Networks,” was filed by the authors’ institution under Application No. **63/942,313**. The filing covers the core algorithmic framework, bias-correction mechanisms, and system-level implementations described in this paper.

Author Contributions: Conceptualization, M.A.; Methodology, M.A. and A.S.; Software, M.A.; Validation, M.A. and A.S.; Formal analysis, M.A.; Investigation, A.S.; Resources, A.S.; Writing—original draft, M.A. and A.S.; Writing—review & editing, M.A. and A.S.; Supervision, A.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Polyak, B.T. Some methods of speeding up the convergence of iteration methods. *USSR Comput. Math. Math. Phys.* **1964**, *4*, 1–17.
2. Sutskever, I.; Martens, J.; Dahl, G.; Hinton, G. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 1139–1147.
3. Tieleman, T.; Hinton, G. Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Netw. Mach. Learn.* **2012**, *4*, 26.
4. Zeiler, M.D. ADADELTA: An adaptive learning rate method. *arXiv* **2012**, arXiv:1212.5701.
5. Kingma, P.; Ba, J. Adam: A method for stochastic optimization. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
6. Reddi, S.J.; Kale, S.; Kumar, S. On the convergence of Adam and beyond. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
7. Loshchilov, I.; Hutter, F. Decoupled weight decay regularization. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.

8. Bottou, L.; Curtis, F.E.; Nocedal, J. Optimization methods for large-scale machine learning. *SIAM Rev.* **2018**, *60*, 223–311.
9. Keskar, N.S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; Tang, P.T.P. On large-batch training for deep learning: Generalization gap and sharp minima. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
10. You, Y.; Li, J.; Reddi, S. J.; Hseu, J.; Kumar, S.; Bhojanapalli, S.; Song, X.; Demmel, J.; Keutzer, K.; Hsieh, C.-J.; et al. Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes. In Proceedings of the Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019.
11. Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; He, K. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv* **2017**, arXiv:1706.02677.
12. Zhuang, J.; Tang, T.; Ding, Y.; Tatikonda, S.; Dvornik, N.; Papademetris, X.; Duncan, J.S. AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients. In Proceedings of the 34th Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 6–12 December 2020.
13. Foret, P.; Kleiner, A.; Mobahi, H.; Neyshabur, B. Sharpness-Aware Minimization for Efficiently Improving Generalization. In Proceedings of the International Conference on Learning Representations, Virtual Event, Austria, 3–7 May 2021.
14. Kwon, J.; Kim, J.; Park, H.; Choi, I.K. ASAM: Adaptive Sharpness-Aware Minimization for Scale-Invariant Learning of Deep Neural Networks. *arXiv* **2021**, arXiv:2102.11600.
15. Anil, R.; Gupta, V.; Koren, T.; Singer, Y. Scalable Second Order Optimization for Deep Learning. In Proceedings of the International Conference on Machine Learning, Virtual, 18–24 July 2021.
16. Heo, B.; Yun, S.; Han, J.; Yoon, S. AdamP: Slowing Down the Slowdown for Momentum Optimizers on Scale-Invariant Weights. In Proceedings of the International Conference on Learning Representations, Virtual Event, Austria, 3–7 May 2021.
17. Xie, X.; Wang, Z.; Zhang, S.; Gu, Q.; Lin, Z. Adan: Adaptive Nesterov Momentum Algorithm for Faster Optimizing Deep Models. In Proceedings of the International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023.
18. Nesterov, Y. A method for solving the convex programming problem with a convergence rate of $O(1/k^2)$. *Sov. Math. Dokl.* **1983**, *27*, 372–376.
19. Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* **2011**, *12*, 2121–2159.
20. Defazio, A.; Mishchenko, K. Learning-Rate-Free Learning by D-Adaptation. In Proceedings of the 40th International Conference on Machine Learning (ICML 2023), Honolulu, HI, USA, 23–29 July 2023.
21. Liu, H.; Li, Z.; Hall, D.; Liang, P.; Ma, T. Sophia: A Scalable Stochastic Second-Order Optimizer for Language Model Pre-Training. *arXiv* **2023**, arXiv:2305.14342.
22. Chen, X.; Liang, C.; Huang, D.; Real, E.; Wang, K.; Liu, Y.; Pham, H.; Dong, X.; Luong, T.; Hsieh, C.-J.; et al. Symbolic Discovery of Optimization Algorithms. *arXiv* **2023**, arXiv:2302.06675.
23. Cauchy, A.-L. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus De L'Académie Des Sci.* **1847**, *25*, 536–538.
24. Newton, I. *Method of Fluxions*; Henry Woodfall: London, UK, 1736.
25. Broyden, G. The convergence of a class of double-rank minimization algorithms. *J. Inst. Math. Its Appl.* **1970**, *6*, 76–90.
26. Fletcher, R. A new approach to variable metric algorithms. *Comput. J.* **1970**, *13*, 317–322.
27. Shanno, F. Conditioning of quasi-Newton methods for function minimization. *Math. Comput.* **1970**, *24*, 647–656.
28. Gauss, F. Über ein neues allgemeines Grundgesetz der Mechanik. *J. Für Die Reine Und Angew. Math.* **1829**, *4*, 232–235.
29. Ortega, M.; Rheinboldt, W.C. *Iterative Solution of Nonlinear Equations in Several Variables*; Academic Press: Cambridge, MA, USA, 1970.
30. Hestenes, M.R.; Stiefel, E. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.* **1952**, *49*, 409–436.
31. Robbins, H.; Monro, S. A stochastic approximation method. *Ann. Math. Stat.* **1951**, *22*, 400–407.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.