Normalization as Control: Batch Normalization Is the Deadbeat Corner of a Kalman-Tuned Family

Anonymous authors
Paper under double-blind review

Abstract

Batch normalization (BN) remains the standard normalizer in convolutional networks, and many variants replace its per-batch statistics with running estimates. Yet it is unclear where BN sits among such tracking normalizers, what should set a tracker's rate, and why principled trackers end up no better than BN. We answer in three steps. First, BN solves a small convex problem exactly on every batch; tracking that solution instead yields a family of normalizers that contains BN as its one-step, or deadbeat, member. Second, a Kalman-filter view sets the tracker's gain from the noise it assumes, and we prove that, once the mean is tracked, the tracker stays at a distance from BN that grows with that assumed noise: the noise model decides how far a tracker may leave BN. Third, experiments on ResNet-20 test this prediction. The standard noise model treats every activation as an independent sample, but the true independent units are the images, so it understates the noise 17- to 31-fold. The tracker then collapses onto BN and even finishes training below it, because its stored inference state is as noisy as a single batch; a one-pass recalibration removes the deficit. Changing only the noise model moves the same tracker away from BN in the predicted order, adding 6–11 points over plain test-batch statistics on corrupted data at batches of 8 or fewer.

1 Introduction

Normalization layers are now part of almost every deep network. Batch normalization (BN) made deep convolutional networks trainable at high learning rates (Ioffe & Szegedy, 2015); layer normalization and RMSNorm serve recurrent networks and Transformers (Ba et al., 2016; Zhang & Sennrich, 2019); group normalization serves detection and segmentation, where batches are small (Wu & He, 2018). Even networks designed without normalization replace it with carefully tuned substitutes (Brock et al., 2021; Zhu et al., 2025). Among these layers BN remains the standard choice for convolutional networks, and its success has produced both a range of explanations and a long line of variants (Huang et al., 2023).

Despite this success, BN has an unusual structure. At every training step it computes each channel's mean and variance from the current batch, normalizes with them, and discards them. This causes its best-known weaknesses: outputs depend on the rest of the batch, statistics are noisy when batches are small, and inference must switch to running averages that never normalized a training batch (Wu & Johnson, 2021; Summers & Dinneen, 2020). A long line of work therefore gives BN a memory, correcting batch statistics toward running averages (Ioffe, 2017), replacing them with running estimates (Chiley et al., 2019; Liao et al., 2016; Yan et al., 2020; Yong et al., 2020), or pooling recent iterations (Yao et al., 2021); at test time, re-estimating or mixing statistics on target data is the basis of adaptation (Li et al., 2017; Schneider et al., 2020). Their rates are set by hand, from the batch size (Yong et al., 2020), or by a one-step analysis at test time (Kimura, 2025), and BN stays the default wherever it works well.

Among the questions these variants leave open, the most basic is whether BN's exact per-batch computation is necessary at all. If it is a choice rather than a necessity, we would like to know what lies around it: where BN sits among normalizers that track their statistics,

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

what should set a tracker's rate, and why principled trackers end up no better than BN. Existing analyses take the per-batch computation as given, explaining BN through loss smoothing, scale invariance or batch noise (Santurkar et al., 2018; Arora et al., 2019; Kohler et al., 2019; Yang et al., 2019; Luo et al., 2019), and the variants are analyzed one at a time (Chiley et al., 2019; Ma & Klabjan, 2024), not as a family around BN.

This paper recasts normalization as control. BN's per-channel computation solves a small convex problem exactly: centering is the least-squares shift, and scaling minimizes a strictly convex function of the log-scale (Lemma 1). Instead of re-solving it on every batch, a controller can keep the shift μ and the log-scale s as states and move them toward each batch's optimum with gains α_μ and α, optionally smoothing with a weight ρ. The resulting family contains BN exactly: when the state is updated before it is used, the corner $(\alpha, \rho, \alpha_\mu) = (1, 1, 1)$ reproduces BN's training-mode forward and backward pass (Proposition 4). In control terms BN is the deadbeat member, which removes any error in one step, and choosing a normalizer becomes a matter of choosing gains.

The analysis draws on control theory, Kalman filtering and stochastic approximation, and proceeds in three steps. (i) The family's fixed point is BN's optimum and its log-domain rate is exactly $1 - \alpha$; with frozen weights and population statistics a deep stack converges at the single-layer rate up to a polynomial factor, while smoothing caps the rate at $\sqrt{1-\rho}$ (Section 4). (ii) Modelling the target log-scale as a random walk makes the scale update a scalar Kalman filter whose gain $1 - r/S$ weighs the innovation power S (the squared batch mismatch) against an assumed observation noise r; the standard $r_n = \frac{1}{4}\psi_1(n/2)$ counts all n reduced values as independent. (iii) Once the mean is tracked, this gain keeps the channel-RMS departure of the log-scale from BN's per-batch solution below $\sqrt{r/\beta}$ for any input (Theorem 8), and on a stationary stream whose true noise is κ times r it settles at $(2\kappa-2)/(2\kappa-1)$ (Proposition 9): the noise model sets how far a tracker may move from BN. Much of (i) and (ii) instantiates classical facts about exponential smoothing and Kalman filtering; what is new is the exact corner and the two statements of (iii), which tie a tracker's distance from BN to its noise model. These results define GFN-2, a member whose gain needs no per-task tuning (Section 5).

We test these predictions on ResNet-20, with the main comparisons and their decision rules fixed in advance (Section 6). GFN-2 costs 1.02× BN's compiled training time and ends 0.23 points below BN on CIFAR-10, a gap five seeds cannot resolve, and 0.91 below on CIFAR-100. It turns out that the noise model is the crux: activations within an image are strongly correlated, so the image is the independent sample and the true noise is 17–31 times r_n. The gain therefore overshoots by the amount Proposition 9 predicts, keeping GFN-2 near BN and its stored inference state as noisy as a single batch; recalibrating that state closes the gap to BN in retrained runs. Changing only the noise model moves the same tracker away from BN in the predicted order: at test batches of 8 or fewer, an image-level noise estimate adds 6–11 points over plain test-batch statistics, and heavy fixed smoothing adds more, since the estimate still misses part of the noise there.

Ultimately, the control view explains BN more than it replaces it: BN's exact per-batch solve is one corner of an analyzable family. Specifically, the main takeaway is that where a tracker sits between BN and a long average is decided by its noise model, with the update rule playing little role: under the analytic noise a principled tracker is BN in disguise, and its deficit is an inference artifact; under a measured noise it averages as far as its estimate of the noise allows.

2 The objective batch normalization solves

We start from the observation on which the rest of the paper builds: BN's per-channel computation is the exact solution of a small convex problem. Fix one channel and let $\boldsymbol{z} \in \mathbb{R}^n$ collect its pre-activations over a batch. For a convolutional layer, $n = B \cdot H \cdot W$ counts every position reduced over, not only the B examples. Write $m = \frac{1}{n}\sum_i z_i$ and $v = \frac{1}{n}\sum_i (z_i - m)^2$. In training mode BN outputs $\mathrm{BN}(\boldsymbol{z}) = (\boldsymbol{z} - m)/\sqrt{v + \epsilon}$, followed by a learned affine map.

Lemma 1 (The per-channel objective). The shift $a^\star = \arg\min_a \|\boldsymbol{z} - a\mathbf{1}\|^2$ equals m. For $v + \epsilon > 0$ the function $F(s) = e^{2s}(v + \epsilon) - 2s$ is strictly convex with unique minimizer $s^\star = -\frac{1}{2}\log(v + \epsilon)$, and $(\boldsymbol{z} - a^\star)e^{s^\star} = \mathrm{BN}(\boldsymbol{z})$.

Proof. The shift is the orthogonal projection onto $\mathbf{1}^\perp$. For the scale, it suffices to find the stationary point: $F'(s) = 2e^{2s}(v + \epsilon) - 2$ vanishes exactly at $e^{2s^\star}(v + \epsilon) = 1$, and $F''(s) = 4e^{2s}(v + \epsilon) > 0$. □

Recall that $x - \log x$ is minimized at $x = 1$. Since $F(s) = x - \log x + \log(v+\epsilon)$ with $x = e^{2s}(v+\epsilon)$, the second moment of the rescaled channel, the objective essentially requires the centered channel to have unit second moment, which BN attains in closed form. Equivalently, for $\epsilon = 0$, F is the Gaussian negative log-likelihood in the log-precision (with $\epsilon > 0$, a variance-floored version of it), so $s^\star$ is the maximum-likelihood estimate, as it is for GN or LN over their own reduction sets; what distinguishes BN is only that its reduction set is the batch, which is what makes tracking across batches possible.

3 Normalization as control

BN solves the problem of Lemma 1 from scratch on every batch. Naturally, one could instead keep the solution as a state and correct it when a new batch arrives. We keep a per-channel state (μ_t, s_t) and correct it with the current batch statistics (m_t, v_t) as follows:

$$\mu_t = \mu_{t-1} + \alpha_\mu\,(m_t - \mu_{t-1}), \tag{1}$$

$$q_t = (1 - \rho)\,q_{t-1} + \rho\,e^{2s_{t-1}}\big(v_t + (m_t - \mu_t)^2 + \epsilon\big), \tag{2}$$

$$s_t = s_{t-1} - \tfrac{\alpha}{2}\log q_t, \tag{3}$$

$$\boldsymbol{y}_t = \gamma \odot (\boldsymbol{z}_t - \mu_t)\,e^{s_t} + \boldsymbol{b}. \tag{4}$$

In words, (1) moves the shift a fraction α_μ of the way to the batch mean; (2) measures the second moment of the batch normalized with the old scale, smoothed over time with weight ρ; (3) moves the log-scale against this mismatch with gain α; and (4) emits with the updated state (update-then-output). Recall that the optimum of Lemma 1 is where this second moment equals one, so $\frac{1}{2}\log q_t$ can be interpreted as the log-domain distance between the old scale and the batch optimum. Three properties follow (proofs in Appendix A).

Proposition 2 (Fixed point). For stationary statistics (m, v) the unique fixed point of (1)–(3) with $\alpha, \alpha_\mu, \rho \in (0, 1]$ is $\mu^\star = m$, $s^\star = -\frac{1}{2}\log(v + \epsilon)$, $q^\star = 1$, the solution of Lemma 1.

Intuitively, the controller can come to rest only at BN's solution, since each equation pins down one state; the gains decide how it gets there, not where it goes.

Proposition 3 (Exact log-domain rate). Let $\rho = 1$, $\mu_t \equiv m$, and $e_t = s_t - s^\star$. Then $e_t = (1 - \alpha)e_{t-1}$ exactly, for errors of any size; the controller converges for every $\alpha \in (0, 2)$.

Concretely, each step multiplies the log-scale error by $1 - \alpha$, however large the error is: with the mean tracked, $q_t = e^{2e_{t-1}}$, and the logarithm in (3) turns the multiplicative scale error into an additive one.

Proposition 4 (BN is the deadbeat corner). At $(\alpha, \rho, \alpha_\mu) = (1, 1, 1)$ the output (4) equals $\gamma \odot \mathrm{BN}(\boldsymbol{z}_t) + \boldsymbol{b}$ for every previous state. With the BN-style backward pass of Section 5, the gradients with respect to $\boldsymbol{z}_t$, γ and $\boldsymbol{b}$ also equal BN's. If the order is reversed (emit with (μ_{t-1}, s_{t-1}), then update), the corner is BN with statistics that lag by one batch.

Put differently, the corner forgets its past completely, and in code GFN-2 at this corner matches torch.nn.BatchNorm2d to within 3×10^{-12} in float64 (Appendix E.5). A controller that removes any error in one step is called deadbeat (Åström & Wittenmark, 1997), so BN is the deadbeat controller for the objective of Lemma 1; strictly speaking, the loop only estimates statistics and is therefore an observer; we use the word "controller" loosely, and analyze no feedback on the weights. The identity concerns training only: at inference every member applies its stored state, which at the corner holds the last training batch's statistics rather than BN's running average. However, fixed gains chosen by hand can fail: 13 of the 36 settings in Figure 3 diverge, all with $\alpha \le 0.3$, whereas the adaptive gains of Section 6.2 pass through that region without diverging.

4 Convergence, spectrum, and the gain

The section proceeds in three moves, first bounding how fast a deep stack settles, then showing what smoothing costs, and finally deriving the gain from a noise model.

Deep stacks. Each layer's statistics depend on the states below it, so a deep stack might settle slowly. Fortunately, the acyclic graph keeps this coupling benign.

Theorem 5 (Cascade convergence). Assume that (A1) weights are frozen and statistics are population statistics; (A2) each channel's variance stays above some $v_{\min} > 0$; and (A3) each layer's target, the fixed point of Proposition 2 for the statistics it receives, is Lipschitz in the states of earlier layers. (i) For one layer with $\rho = 1$, the mean error decays at rate $|1 - \alpha_\mu|$ and the log-scale error at $\max\{|1-\alpha|, (1-\alpha_\mu)^2\}$, up to a constant and, when the two rates are equal, a factor t. (ii) If, with its target held fixed, each layer's update is a λ-contraction on a neighborhood of the fixed point that the iterates do not leave, the distance $E_\ell(t)$ of layer ℓ's states from the joint fixed point of an L-layer stack satisfies $E_\ell(t) \le p_\ell(t)\lambda^t$ for a polynomial p_ℓ of degree at most $\ell - 1$.

Each assumption has one job: (A1) isolates the controller from learning; (A2), which depends on the data, keeps the logarithms finite; (A3), which depends on the activations, limits how far an upstream error travels. The upshot is that a layer's rate depends only on its own gains, and depth costs a polynomial prefactor but never the rate, because the acyclic graph makes each layer a contraction driven by a vanishing input from below (Appendix A).

Proposition 6 (Smoothing caps the rate). For $\rho < 1$ and a tracked mean, the local Jacobian J of (2)–(3) in $(e_t, \log q_t)$ has $\det J = 1 - \rho$, so for every α its spectral radius is at least $\sqrt{1-\rho}$, with equality when the eigenvalues have equal modulus.

That is, the two modes' memories multiply to $1 - \rho$ whatever α is (Appendix A), so in the noise-free limit smoothing the second moment costs speed that no gain recovers; the variance it is meant to reduce can be reduced through the gain itself, as a single-state ($\rho = 1$) filter does (Muth, 1960).

The gain as a Kalman filter. So far the gains were free. However, the right gain depends on how fast the target moves and how noisy each batch's estimate of it is, and a filtering model makes both explicit. We model the optimal log-scale as a random walk, the standard surrogate for slow drift, observed through BN's batch solution:

$$s_t^\star = s_{t-1}^\star + w_t, \qquad \hat{s}_t = -\tfrac{1}{2}\log(v_t + \epsilon) = s_t^\star + \xi_t, \tag{5}$$

where the drift w_t has unknown variance σ_w^2 and ξ_t is the observation noise. The innovation is the log-energy mismatch before any update, $\nu_t = \frac{1}{2}\log\big(e^{2s_{t-1}}(v_t + (m_t - \mu_{t-1})^2 + \epsilon)\big)$. When the mean is tracked, $\nu_t = s_{t-1} - \hat{s}_t$, so with $\rho = 1$, (3) reads $s_t = s_{t-1} - \alpha\nu_t$, which can be interpreted as a scalar Kalman measurement update with gain α.

Proposition 7 (Kalman gain with analytic observation noise). If the n reduced values are i.i.d. Gaussian with known mean and ϵ is ignored, $\mathrm{Var}(\xi_t) = r_n = \frac{1}{4}\psi_1(n/2)$, where ψ_1 is the trigamma function, and $r_n \to 1/(2n)$ for large n. Treating ξ_t as Gaussian with this variance, the steady-state Kalman gain is $\alpha = 1 - r_n/S$ with $S = \mathbb{E}[\nu_t^2]$. As $\sigma_w^2 \to \infty$, $\alpha \to 1$: BN is the Kalman filter for unbounded drift. As $\sigma_w^2 \to 0$, $\alpha \to 0$.

Read this way, the gain weighs drift against noise: a fast-moving target calls for trusting each batch ($\alpha \to 1$, BN), a still one for averaging over many ($\alpha \to 0$). Since r_n depends only on n (Bartlett & Kendall, 1946), the gain needs no tuning once S is measured, as in innovation-based adaptive filtering (Mehra, 1970). Unfortunately, the i.i.d. model is an idealization: Section 6.2 finds the true noise r_{true} far above r_n, so $1 - r_n/S$ exceeds the model-optimal $1 - r_{\text{true}}/S$. The adaptive law replaces S by a running average of the channel-mean squared innovation, defined as:

$$\hat{S}_t = (1-\beta)\hat{S}_{t-1} + \beta\,\bar{\nu}_t^2, \qquad \alpha_t = \mathrm{clip}\big(1 - r_n/\hat{S}_t,\ \alpha_{\min},\ 1\big), \qquad \bar{\nu}_t^2 = \tfrac{1}{C}\textstyle\sum_c \nu_{t,c}^2, \tag{6}$$

where β is the averaging rate and C the number of channels, so each layer has one gain. Effectively, each layer compares its recent mismatches with the noise it expects: mismatches

well above r_n mean the target moves, and the gain goes to one. Because α_t uses the current innovation and is shared across channels, (6) is a Kalman-motivated rule rather than the Kalman filter of (5); the results below concern (6) itself.

How far can the tuned controller move from BN? We will now show that the distance is capped by the noise the rule assumes: large innovations drive the gain to one, while small innovations leave little to correct, so their product is bounded.

Theorem 8 (Proximity to the deadbeat solve). Run (1)–(3) with $\rho = 1$, $\alpha_\mu = \alpha_t$ and the gain (6). Let $(\mu_t^{\mathrm{BN}}, s_t^{\mathrm{BN}}) = (m_t, \hat{s}_t)$ be BN's solution on batch t and $\delta_t = m_t - \mu_{t-1}$. Then for every step and any input sequence:

(a) $\mu_t - \mu_t^{\mathrm{BN}} = -(1-\alpha_t)\,\delta_t$ and $s_t - s_t^{\mathrm{BN}} = (1-\alpha_t)(s_{t-1} - \hat{s}_t) - \frac{\alpha_t}{2}\log\big(1 + (1-\alpha_t)^2\delta_t^2/(v_t+\epsilon)\big)$, so every departure from BN carries the factor $1-\alpha_t$;

(b) $(1-\alpha_t)\,\bar{\nu}_t \le \min\{(1-\alpha_{\min})\bar{\nu}_t,\ r_n/(\beta\bar{\nu}_t)\} \le \sqrt{r_n/\beta}$, where $\bar{\nu}_t$ is the channel-RMS innovation; when the mean is tracked ($\delta_t = 0$), $(1-\alpha_t)\bar{\nu}_t$ is exactly the channel-RMS log-scale departure from BN, which is therefore at most $\sqrt{r_n/\beta}$;

(c) if $\hat{S}_{t-1} \ge r_n$, which holds at the first step ($\hat{S}_0 = r_n$) and whenever the previous unclipped gain was nonnegative, the bound tightens to $\sqrt{r_n}\,/\,\big(2\sqrt{\beta(1-\beta)}\big)$.

What (b) says, in plain terms, is that the factor $1-\alpha_t$ and the innovation cannot both be large, so the departure is capped by a constant set by the assumed noise and the averaging rate β, and (c) adds that the first step is safer still. Since (a) and (b) use only the gain rule, they hold verbatim with any per-step noise r_t in place of r_n, such as the calibrated one of Section 6.2, while (c) then needs $\hat{S}_{t-1} \ge r_t$: how far the tracker may leave BN is decided by its noise model.

With $\beta = 0.05$ the constants are $4.5\sqrt{r_n}$ and $2.3\sqrt{r_n}$; the recorded training gains (at least 0.55 at the last batch of every epoch) keep $\hat{S}$ well above r_n, so in ResNet-20 at $B = 128$ ($n \ge 8192$) the channel-RMS log-scale departure is at most 0.018. The surprising part is that, under the noise measured in Section 6.2, BN's own batch solution fluctuates by 0.03–0.04 at $n = 8192$, so the rule removes only part of the noise it should filter. There are, of course, limitations: the cap assumes a tracked mean (Appendix A bounds the mean-mismatch term), and it says nothing about accuracy directly.

Where the gain settles. On a stationary stream we can also say where the gain comes to rest when the assumed noise is wrong.

Proposition 9 (Gain under a misspecified noise). Run (6) with a tracked mean on a stationary stream whose observation noise is $\kappa \ge 1$ times the noise r the rule assumes, and replace $\hat{S}_t$ by its mean. Then the gain settles at $\alpha^\star = \max\{\alpha_{\min}, (2\kappa-2)/(2\kappa-1)\}$; in terms of the fraction $f = 1/\kappa$ of the true noise that the rule assumes, $\alpha^\star = (2-2f)/(2-f)$ above the floor.

A stationary target, in principle, calls for the smallest gain, which the rule reaches only if its noise is nearly exact: near $f = 1$ each 10% of missed noise adds about 0.2 to the gain, and a κ-fold underestimate holds it near $1 - 1/(2\kappa)$ (proof in Appendix A; simulated in Appendix B).

5 GFN-2: a member motivated by the analysis

We now collect the choices the analysis supports into one member of the family. GFN-2 (gain-filtered normalization; the suffix counts its two tracked states, μ and s) uses $\rho = 1$ (Proposition 6), update-then-output order (Proposition 4) and the gain (6) with the analytic r_n (Proposition 7). Two further choices are heuristic: the mean shares the scale gain ($\alpha_\mu = \alpha_t$), and the backward pass reuses BN's gradient formula at the controller's state, a surrogate when $\alpha_t < 1$ that is within 1.2% of the fixed-gain derivative in random tests

Table 1: Final-epoch test accuracy (%, mean ± s.d. over five seeds) and training time per epoch relative to BN (compiled / eager); per-seed values in Appendix E.

Method	CIFAR-10	CIFAR-100	time vs. BN
BN	92.09 ± 0.16	67.39 ± 0.43	1.00 / 1.00
GFN-2	91.87 ± 0.34	66.48 ± 0.44	1.02 / 1.31
GN	90.25 ± 0.18	61.09 ± 0.66	—
LN	90.49 ± 0.23	64.23 ± 0.34	—

at $\alpha = 0.9$ (10% for a stop-gradient diagonal) but less accurate at low gains and for some upstream gradients (Appendix A.8). A second member, GFN-2c (calibrated), differs only in the noise that (6) assumes, which it estimates from the images in each batch (Section 6.2); Algorithm 1 in Appendix C gives one step of either for one layer.

The backward pass is $\partial \boldsymbol{z} = e^s\big(\boldsymbol{g} - \overline{\boldsymbol{g}} - \hat{\boldsymbol{z}}\,\overline{\boldsymbol{g} \odot \hat{\boldsymbol{z}}}\big)$, where $\boldsymbol{g}$ is the gradient with respect to $\hat{\boldsymbol{z}} = (\boldsymbol{z} - \mu)e^s$ and bars are per-channel means. That is, BN's formula at the controller's state removes the gradient along the constant and along $\hat{\boldsymbol{z}}$, the directions the normalization constraint fixes, exactly at the corner and approximately elsewhere. Defaults are $\beta = \alpha_{\min} = 0.05$ and $\epsilon = 10^{-5}$, fixed for every experiment; with the analytic noise at $B = 128$, a 3×3 grid around them moves test-time accuracy by at most 0.003 points (Appendix E).

6 Experiments

The experiments put the analysis to the test. We first verify that GFN-2 matches BN, then trace the measurements that keep it so close, and finally probe whether the gain helps where the theory predicts it should.

Setup. We train a CIFAR ResNet-20 (He et al., 2016) whose 19 normalization layers are all BN, GroupNorm (GN, $\min(32, C)$ groups; Wu & He, 2018), LayerNorm over (C, H, W) (LN; Ba et al., 2016), or GFN-2, on CIFAR-10 and CIFAR-100 (Krizhevsky, 2009), with SGD (momentum 0.9), batch size 128, learning rate 0.03 with cosine decay over 200 epochs, weight decay 5×10^{-4}, crop-and-flip augmentation, torch.compile and five seeds. The learning rate came from a short pilot on GFN-2 and was reused for all methods, so GN and LN serve as references under a shared recipe rather than separately tuned competitors; each protocol and its decision rule were fixed before the run (Appendix D).

6.1 Accuracy and cost

We first compare accuracy and cost. At the final epoch GFN-2 is 0.23 points below BN on CIFAR-10 (paired 95% confidence interval $[-0.57, +0.11]$, lower in 4 of 5 seeds), a gap five seeds cannot resolve, and 0.91 points below on CIFAR-100 ($[-1.67, -0.15]$), beyond the parity margin we fixed in advance. Both methods stay 1.4–1.8 (CIFAR-10) and 2.3–6.3 (CIFAR-100) points above LN and GN. In the compiled loop used for all accuracy numbers GFN-2 takes 1.02× BN's time per epoch (1.31× eager); in an isolated forward–backward benchmark, where data loading plays no role, it takes 1.16–1.18× compiled and 2.2–3.2× eager (Table 7). Section 6.2 links the shortfall to a noisy stored inference state.

6.2 Why GFN-2 trails BN, and how to close the gap

Why does a tuned filter stay so close to BN, and why does it end training below it? We answer by measurement, then by intervention.

The analytic noise constant is too small. During training GFN-2's gain stays between 0.55 and 0.999 (Figure 2). To see why, we measure the observation noise on retrained models (Appendix D). With every normalization state frozen, so that only the sampled images vary, the variance of the batch solution $\hat{s}$ across independent batches exceeds r_n in every layer of every model, by a median factor κ of 17–31 (Figure 1a). It turns out that the image,

Table 2: Mechanism tests on retrained ResNet-20 (final epoch; CIFAR-100 mean of two seeds, CIFAR-10 one seed; per-seed values in Appendix E.3). Gains: range over layers and per-epoch records. Recalibrated: the stored state is replaced by training-set averages of batch statistics, as precise BN does.

		CIFAR-100			CIFAR-10	
Method	gains in training	train	stored	recalibrated	stored	recalibrated
BN	1 (deadbeat)	90.40	67.61	67.51	92.34	92.38
GFN-2 (analytic r_n)	0.59–1.00	90.28	66.95	67.93	91.58	92.13
GFN-2c (image-level $\hat{r}$)	0.05–0.62	91.54	67.27	67.51	91.80	91.80

not the position, is the independent sample: neighboring activations are strongly correlated (lag-1 correlation 0.58), the batch mean is similarly inflated, and heavy tails explain less than a tenth of the excess. Proposition 9 turns this into a prediction: in controller traces of the original models on a stationary stream, where adapting upstream layers renormalize each batch and the median κ is 9 (Appendix E.2), it predicts the median gain of every layer to within 0.017 at $B = 32$ with no fitting, and the innovations have lag-1 autocorrelation -0.46 to -0.48, the $-\alpha/2$ of white noise filtered with gain α, so by Mehra's whiteness test the gain is far too high for that stream (Figure 1b). During training the weights also move the target, a drift we do not measure; whatever its size, a κ-fold underestimate of the noise biases the gain toward one.

Calibrating the noise. The measurement also suggests the fix. For a batch of B images, let e_{ic} be image i's mean squared deviation from the batch mean in channel c. The batch variance is the average of the e_{ic}, so, treating the B energies as independent and linearizing the logarithm, its log has variance approximately:

$$\hat{r} = \frac{1}{C} \sum_{c} \frac{\mathrm{Var}_i(e_{ic})}{4B\,\bar{e}_c^2}, \tag{7}$$

where $\bar{e}_c$ is the mean over images. In plain terms, the noise is read off from how much the images in a batch disagree about each channel's energy, which captures within-image correlation and differences between images at once. Both steps of this plug-in estimate become exact only as B grows, since the energies share the batch mean and the logarithm of an average of few energies spreads more than its linearization (Appendix A.7). At $B = 32$ and 128 it matches the frozen-state noise to within 20% in every layer of every retrained model (Figure 1a, lines); Section 6.3 examines smaller B. GFN-2c replaces r_n in (6) by $\max\{r_n, \hat{R}_t\}$, where $\hat{R}_t$ averages $\hat{r}$ at the same rate β; it costs one more reduction and no tuning. Its gains fall to 0.05–0.62 during training, far from BN's corner, and training completes with the same surrogate backward, although fixed gains that low diverge (Figure 3); since adaptive gains start near one and the surrogate is least accurate at low gains (Appendix A.8), we report this as an observation.

The stored inference state inherits the overshoot. With gains near one the stored state is essentially a one-batch estimate: its variance is $\alpha/(2-\alpha)$ times the per-batch noise, about 0.8 at $\alpha = 0.9$ against 0.05 for BN's running average, and the last batch of an epoch has only 80 images. Accordingly, GFN-2's test accuracy fluctuates 3.4–3.5× more than BN's over the last 20 epochs (Figure 1c). One forward pass over the training set, with the training augmentation, replaces the stored state by averaged batch statistics, as precise BN does (Wu & Johnson, 2021). This recalibration raises GFN-2 by 0.98 points on CIFAR-100 and 0.55 on CIFAR-10, closing its gap to BN in these runs, and leaves BN unchanged (Table 2); GFN-2c, whose lower gain already smooths the stored state, gains only 0.24 and 0.00. That GFN-2c does not also overtake BN fits what a filter optimizes, namely the error of the statistic rather than generalization. Two candidate reasons, which we do not separate, are BN's regularizing batch noise (Luo et al., 2019), consistent with GFN-2c's 1.1–1.3 points higher training accuracy on CIFAR-100, and the surrogate backward at low gains. In short, in these runs GFN-2's deficit is an inference artifact, and the remedy is the one BN already uses: a slow inference average kept apart from the training statistics.

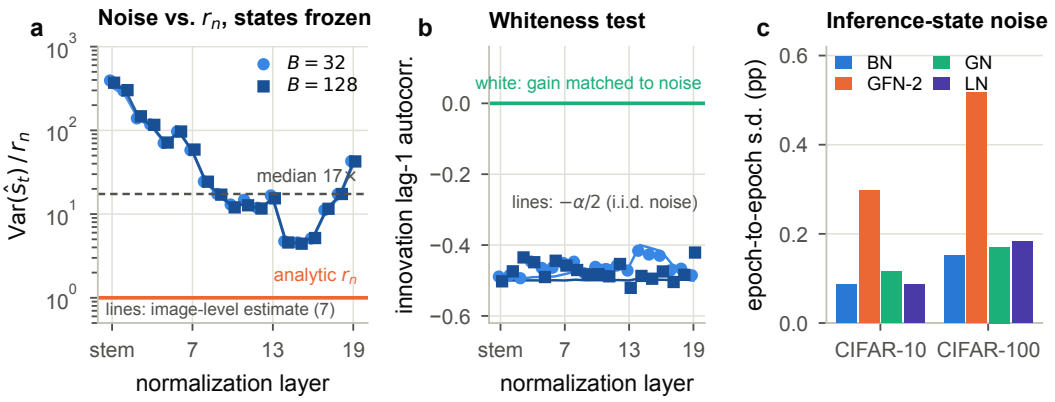

Figure 1: Diagnosing the gain. (a) Variance of $\hat{s}$ across independent batches with every normalization state frozen (retrained CIFAR-10 GFN-2), relative to r_n, per layer; lines: the image-level estimate (7). (b) Lag-1 autocorrelation of the innovations in controller traces of the original model (markers) against $-\alpha/2$ for white noise filtered with the recorded gain (lines); a matched gain gives zero. (c) Epoch-to-epoch s.d. of test accuracy over the last 20 epochs (five seeds).

6.3 Test-time adaptation and small batches

So far the gain has had little room to act; we now test it under distribution shift and at small batches. On CIFAR-10-C (Hendrycks & Dietterich, 2019) (19 corruptions, 5 severities, no gradients or labels), each (corruption, severity) cell is a stationary stream that starts from the trained state, and each method runs with stored and with test-batch statistics: BN in training mode (Nado et al., 2020; Schneider et al., 2020), GFN-2 running its controller, and GFN-2 at its deadbeat corner, $\alpha \equiv 1$.

The gain from adaptation is the gain from test statistics. Switching to test-batch statistics recovers about ten points for BN and GFN-2 alike (Table 8), and GN, which never stores statistics, lands within 0.27 points of both. Adaptive GFN-2 is indistinguishable from BN with batch statistics (-0.04 points), and replacing its gain by $\alpha \equiv 1$ changes accuracy by at most 0.18 points in any of the 475 cells, consistent with the 0.018 cap of Theorem 8 at $n \geq 8192$: the benefit comes from test statistics; the analytic controller adds essentially nothing.

The noise model orders the gains at small batches. By Proposition 9, a stationary stream calls for the lowest gain, and the rule approaches it only as far as its noise model allows. We test this within one retrained GFN-2 model, changing only the gain (Figure 2b, Table 5): the analytic rule, the image-level rule of Section 6.2, and, as a reference for heavy smoothing rather than a method, the fixed $\alpha = 0.1$ that was best at $B = 128$ on the original models. The gains line up in the predicted order: over the deadbeat corner at $B = 8$, 4 and 2, the analytic rule adds 0.3, 0.7 and 1.5 points (0.4, 0.8 and 1.8 over the five original seeds), the image-level rule 5.9, 8.3 and 10.9, and fixed smoothing 6.6, 11.7 and 19.1. At $B \geq 32$, where (7) matches the measured noise, the image-level rule reaches the fixed gain's accuracy with no tuning (77.34 vs. 77.30% at $B = 32$, 77.46 vs. 77.53% at $B = 128$). At smaller B it stops short, because (7) then misses part of the noise: in a synthetic model with the measured correlation it recovers 93–101% of the true noise at $B = 128$ but 66–80% at $B = 8$ and 36–46% at $B = 2$ (Appendix A.7), which by Proposition 9 would hold the gain at 0.3–0.8 instead of near its floor. Put simply, how much a tracker gains at small batches follows how much noise it assumes. This comparison has its limits, of course: stationary streams reward heavy averaging and do not exercise the rule's response to drift, and per-sample normalizers such as GN remain stronger at the smallest batches (Figure 2a); we focus on what governs a batch-statistics tracker rather than on which normalizer to deploy.

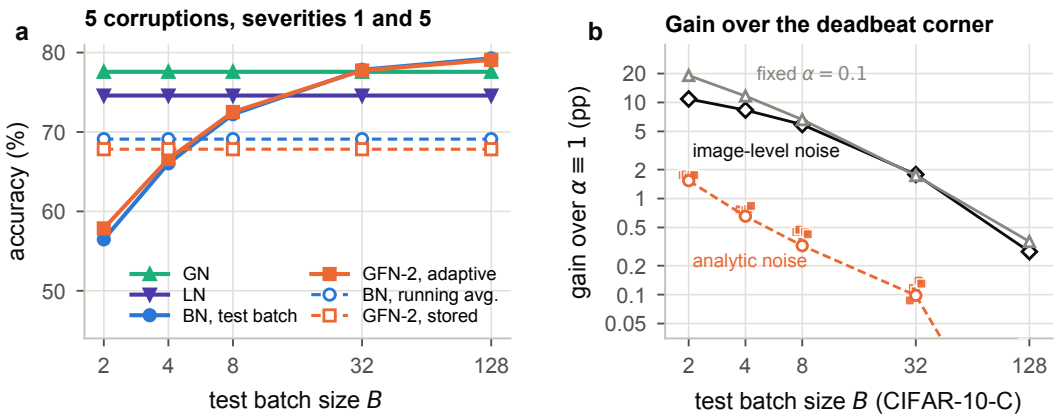

Figure 2: Test-time normalization on CIFAR-10-C (brightness, fog, impulse noise, shot noise, zoom blur; severities 1 and 5; each cell a stationary stream). (a) Original models, five seeds: test-batch statistics (solid blue, orange) collapse at small B; per-sample normalizers (green, purple) and stored statistics (dashed) do not depend on B. (b) Gain over the deadbeat corner ($\alpha \equiv 1$) within one retrained GFN-2 model, changing only the gain; squares: analytic gain on the five original seeds. B also sets the updates per pass, so trends in B are exploratory.

7 Related work

Normalizers with memory. Online Normalization updates running statistics after use and controls the backward pass (Chiley et al., 2019); Momentum BN sets its rate from batch size and training stage (Yong et al., 2020); Cross-Iteration BN pools iterations (Yao et al., 2021); Diminishing BN proves convergence (Ma & Klabjan, 2024); and Kalman Normalization filters across layers (Wang et al., 2018). GFN-2 instead updates log-scale before use and sets its gain from innovation noise (Table 11), building on deadbeat control (Åström & Wittenmark, 1997), Kalman filtering (Kalman, 1960), noise identification (Mehra, 1970) and stochastic approximation (Robbins & Monro, 1951; Borkar, 1997).

Test-time adaptation. Target-data BN statistics (Li et al., 2017; Nado et al., 2020; Schneider et al., 2020) underlie TENT and CoTTA (Wang et al., 2021; 2022). Other methods mix source and test statistics (You et al., 2021; Mirza et al., 2022; Lim et al., 2023), average them over a stream (Hu et al., 2021; Hong et al., 2023; Kang et al., 2024; Yuan et al., 2023; Su et al., 2024), derive an MSE-optimal weight (Kimura, 2025), or address correlated streams and small batches (Gong et al., 2022; Tomar et al., 2024; Niu et al., 2023).

8 Limitations and conclusion

Limitations. Our evidence comes from ResNet-20 on CIFAR-10, CIFAR-100 and CIFAR-10-C with stationary test streams; the mechanism tests use one or two retrained seeds and the small-batch comparison one model; we compare with BN, GN and LN under a learning rate chosen on GFN-2. The theory assumes frozen weights (Theorem 5), random-walk drift and Gaussian noise (Proposition 7), a tracked mean (Theorem 8) and a stationary stream (Proposition 9); (7) is biased low at small B, and the backward pass is a surrogate away from the corner.

Conclusion. BN is the deadbeat corner of a family of trackers whose Kalman-tuned member provably remains at a distance from BN bounded by the noise it assumes. Since the true independent units are images, the analytic noise is too small and the tracker collapses onto BN; a measured noise moves it away in the predicted order. The distance a tracker should keep from BN is set by the noise; the choice of update rule matters only after that.

References

Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. Theoretical analysis of auto rate-tuning by Batch Normalization. In International Conference on Learning Representations (ICLR), 2019.

Karl J. Åström and Björn Wittenmark. Computer-Controlled Systems: Theory and Design. Prentice Hall, 3rd edition, 1997.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. arXiv preprint arXiv:1607.06450, 2016. URL https://arxiv.org/abs/1607.06450.

M. S. Bartlett and D. G. Kendall. The statistical analysis of variance-heterogeneity and the logarithmic transformation. Supplement to the Journal of the Royal Statistical Society, 8(1):128–138, 1946. doi: 10.2307/2983618.

Vivek S. Borkar. Stochastic approximation with two time scales. Systems & Control Letters, 29(5):291–294, 1997. doi: 10.1016/S0167-6911(97)90015-3.

Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In Proceedings of the 38th International Conference on Machine Learning (ICML), volume 139 of Proceedings of Machine Learning Research, pp. 1059–1071. PMLR, 2021.

Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Koster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James. Online Normalization for training neural networks. In Advances in Neural Information Processing Systems (NeurIPS), volume 32, 2019.

Taesik Gong, Jongheon Jeong, Taewon Kim, Yewon Kim, Jinwoo Shin, and Sung-Ju Lee. NOTE: Robust continual test-time adaptation against temporal correlation. In Advances in Neural Information Processing Systems (NeurIPS), volume 35, pp. 27253–27266, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, 2016.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In International Conference on Learning Representations (ICLR), 2019.

Junyuan Hong, Lingjuan Lyu, Jiayu Zhou, and Michael Spranger. MECTA: Memory-economic continual test-time model adaptation. In International Conference on Learning Representations (ICLR), 2023.

Xuefeng Hu, Gokhan Uzunbas, Sirius Chen, Rui Wang, Ashish Shah, Ram Nevatia, and Ser-Nam Lim. MixNorm: Test-time adaptation through online normalization estimation. arXiv preprint arXiv:2110.11478, 2021.

Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training DNNs: Methodology, analysis and application. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(8):10173–10196, 2023. doi: 10.1109/TPAMI.2023.3250241.

Sergey Ioffe. Batch Renormalization: Towards reducing minibatch dependence in batch-normalized models. In Advances in Neural Information Processing Systems (NIPS), volume 30, 2017.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning (ICML), volume 37 of Proceedings of Machine Learning Research, pp. 448–456. PMLR, 2015.

R. E. Kalman. A new approach to linear filtering and prediction problems. Journal of Basic Engineering, 82(1):35–45, 1960. doi: 10.1115/1.3662552.

Juwon Kang, Nayeong Kim, Jungseul Ok, and Suha Kwak. MemBN: Robust test-time adaptation via batch norm with statistics memory. In European Conference on Computer Vision (ECCV), 2024.

Masanari Kimura. Higher-order asymptotics of test-time adaptation for Batch Normalization statistics. arXiv preprint arXiv:2505.16257, 2025. URL https://arxiv.org/abs/2505.16257.

Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. Exponential convergence rates for Batch Normalization: The power of length-direction decoupling in non-convex optimization. In Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS), volume 89 of Proceedings of Machine Learning Research, pp. 806–815. PMLR, 2019.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting Batch Normalization for practical domain adaptation. In International Conference on Learning Representations (ICLR), Workshop Track, 2017.

Qianli Liao, Kenji Kawaguchi, and Tomaso Poggio. Streaming Normalization: Towards simpler and more biologically-plausible normalizations for online and recurrent learning. arXiv preprint arXiv:1610.06160, 2016. URL https://arxiv.org/abs/1610.06160.

Hyesu Lim, Byeonggeun Kim, Jaegul Choo, and Sungha Choi. TTN: A domain-shift aware Batch Normalization in test-time adaptation. In International Conference on Learning Representations (ICLR), 2023.

Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in Batch Normalization. In International Conference on Learning Representations (ICLR), 2019.

Yintai Ma and Diego Klabjan. Diminishing batch normalization. IEEE Transactions on Neural Networks and Learning Systems, 35(5), 2024.

Raman K. Mehra. On the identification of variances and adaptive Kalman filtering. IEEE Transactions on Automatic Control, 15(2):175–184, 1970. doi: 10.1109/TAC.1970.1099422.

M. Jehanzeb Mirza, Jakub Micorek, Horst Possegger, and Horst Bischof. The norm must go on: Dynamic unsupervised domain adaptation by normalization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 14765–14775, 2022.

John F. Muth. Optimal properties of exponentially weighted forecasts. Journal of the American Statistical Association, 55(290):299–306, 1960.

Zachary Nado, Shreyas Padhy, D. Sculley, Alexander D'Amour, Balaji Lakshminarayanan, and Jasper Snoek. Evaluating prediction-time Batch Normalization for robustness under covariate shift. arXiv preprint arXiv:2006.10963, 2020. URL https://arxiv.org/abs/2006.10963.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Zhiquan Wen, Yaofo Chen, Peilin Zhao, and Mingkui Tan. Towards stable test-time adaptation in dynamic wild world. In International Conference on Learning Representations (ICLR), 2023.

Herbert Robbins and Sutton Monro. A stochastic approximation method. The Annals of Mathematical Statistics, 22(3):400–407, 1951. doi: 10.1214/aoms/1177729586.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does Batch Normalization help optimization? In Advances in Neural Information Processing Systems (NeurIPS), volume 31, 2018.

Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Improving robustness against common corruptions by covariate shift adaptation. In Advances in Neural Information Processing Systems (NeurIPS), volume 33, pp. 11539–11551, 2020.

Zixian Su, Jingwei Guo, Kai Yao, Xi Yang, Qiufeng Wang, and Kaizhu Huang. Unraveling batch normalization for realistic test-time adaptation. In AAAI Conference on Artificial Intelligence, volume 38, 2024.

Cecilia Summers and Michael J. Dinneen. Four things everyone should know to improve Batch Normalization. In International Conference on Learning Representations (ICLR), 2020.

Devavrat Tomar, Guillaume Vray, Jean-Philippe Thiran, and Behzad Bozorgtabar. Unmixing test-time normalization statistics: Combatting label temporal correlation. In International Conference on Learning Representations (ICLR), 2024.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In International Conference on Learning Representations (ICLR), 2021.

Guangrun Wang, Jiefeng Peng, Ping Luo, Xinjiang Wang, and Liang Lin. Kalman Normalization: Normalizing internal representations across network layers. In Advances in Neural Information Processing Systems (NeurIPS), volume 31, 2018.

Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual test-time domain adaptation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7201–7211, 2022.

Yuxin Wu and Kaiming He. Group Normalization. In Proceedings of the European Conference on Computer Vision (ECCV), pp. 3–19, 2018.

Yuxin Wu and Justin Johnson. Rethinking "batch" in BatchNorm. arXiv preprint arXiv:2105.07576, 2021. URL https://arxiv.org/abs/2105.07576.

Junjie Yan, Ruosi Wan, Xiangyu Zhang, Wei Zhang, Yichen Wei, and Jian Sun. Towards stabilizing batch statistics in backward propagation of Batch Normalization. In International Conference on Learning Representations (ICLR), 2020.

Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. A mean field theory of Batch Normalization. In International Conference on Learning Representations (ICLR), 2019.

Zhuliang Yao, Yue Cao, Shuxin Zheng, Gao Huang, and Stephen Lin. Cross-iteration Batch Normalization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 12331–12340, 2021.

Hongwei Yong, Jianqiang Huang, Deyu Meng, Xiansheng Hua, and Lei Zhang. Momentum Batch Normalization for deep learning with small batch size. In Proceedings of the European Conference on Computer Vision (ECCV), pp. 224–240, 2020.

Fuming You, Jingjing Li, and Zhou Zhao. Test-time Batch statistics calibration for covariate shift. arXiv preprint arXiv:2110.04065, 2021. URL https://arxiv.org/abs/2110.04065.

Longhui Yuan, Binhui Xie, and Shuang Li. Robust test-time adaptation in dynamic scenarios. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 15922–15932, 2023.

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

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 14901–14911, 2025.