

DaMN: Deleting and Migrating Normalization Layers from Transformers

Alexey Ryabykin¹ Irina Zhelavskaya¹ Egor Shvetsov² Alexey Rukhovich¹
 Nikita Okhotnikov¹ Artem Khrapov¹ Evgeny Burnaev²
 Vladimir Mikhailovich Kryzhanovskiy¹

¹AI Foundation & Algorithm Lab ²APPLIED-AI

Abstract

Normalization layers are essential for stabilizing and accelerating the training of transformer-based neural networks, yet they introduce significant memory traffic and computationally expensive reduction operations that degrade inference efficiency in terms of both latency and energy consumption. While previous approaches (e.g., DyT, SLAB) require training from scratch to replace or approximate normalization layers, we investigate whether it is possible to remove them from pretrained models while preserving model quality. To this end, we present DaMN, a method for post-training normalization removal. Evaluated on a range of tasks including image classification, super-resolution, and denoising, DaMN outperforms state-of-the-art removal and replacement strategies, it maintains high performance with less than 0.2 dB PSNR loss on super-resolution and denoising tasks. Computational gains in terms of inference time and energy consumption were carefully evaluated on a wide range of representative hardware, further confirming the practicality of our approach. Normalization layer removal leads to up to **18% faster inference** and **17% lower energy consumption**. The approach is framework-agnostic and deployment-friendly, requiring only a one-time calibration step and scheduled fine-tuning, with no architectural modifications, retraining from scratch, or custom kernels. These results highlight an additional opportunity for efficient deep learning deployment: use normalization during training to improve stability, and eliminate it during inference to enhance efficiency.¹

1. Introduction

Normalization layers (NL) are foundational in modern deep learning, stabilizing and accelerating training convergence [12, 13, 20]. Yet despite their algorithmic simplicity, NL incur significant dataflow inefficiencies, low arithmetic intensity, and high memory traffic that degrade latency and

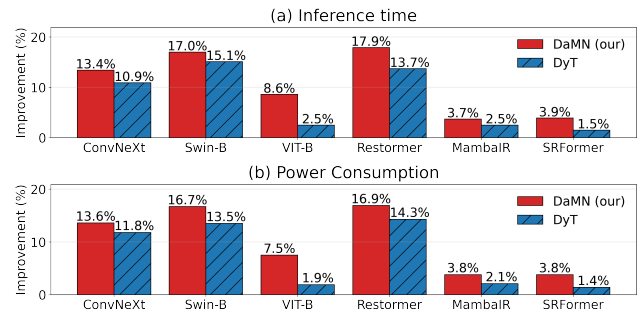


Figure 1. a) Inference time and b) energy consumption improvements of DyT [34] and DaMN compared to the original model with normalization layers (NL), evaluated on an A100 GPU with FP16 data type. DaMN enables the complete removal of all NL layers, resulting in the fastest and most energy-efficient model. Appendix A contains measurements across a range of GPU architectures and data types.

energy efficiency, especially on bandwidth and power-constrained hardware.

Figure 1 presents the relative improvements of DNN performance when NL is removed (red bars). For example, removing NL from the Restormer model [30] results in a significant decrease in inference time by 17.9%, and energy consumption by 16.9%. These strong performance benefits motivate us to find ways to effectively remove NL while avoiding accuracy degradation. Further analysis on other GPU types and additional details can be found in Appendix A.

A common mitigation strategy is folding NL into neighboring layers. This works well for static normalizations like BatchNorm [13], whose fixed statistics fuse cleanly into subsequent linear layers. However, this approach fails for data-dependent NL, which compute per-input statistics (e.g., mean and variance) and often involve nonlinear operations like square roots and divisions, hindering analytical fusion.

Another strategy involves substituting the NL with alternative operators. A notable example is the recently introduced technique DyT [34], which replaces LayerNorm

¹The implementation is available at [GitHub](#).

with a learnable scaled hyperbolic tangent activation function. DyT achieves performance comparable to the original models containing *NL*, while offering significant computational advantages, as illustrated by the blue bars in Figure 1. Nevertheless, this approach still requires retraining the models from scratch and does not yet match the computational efficiency achieved by completely removing *NL*s.

In this paper, we show that after training, *NL* can be entirely removed with minimal performance loss. Our method, DaMN (Deleting and Migrating Normalizations), uses fine-tuning to smoothly transition each *NL* into an identity mapping, redistributing its nonlinear compensation into adjacent trainable layers.

DaMN comprises three key components: (1) **Smooth Removal:** A scheduled fine-tuning process that gradually changes each *NL* into identity via a mixing coefficient, avoiding activation explosion and instability. (2) **Moment-Matched Affine initialization:** Each *NL* is replaced by an affine layer initialized to preserve input/output moments, preventing signal collapse. (3) **Sequential calibration:** Instead of calibrating all *NL* layers at once, DaMN updates them one by one, so that small mismatches do not get amplified in deeper layers.

Our contribution: Despite practical demand for normalization-free deployment, removing LN from pre-trained models remains an unstudied challenge. We present DaMN, a careful removal procedure that avoids signal collapse through moment-matched initialization, sequential calibration, and smooth fine-tuning. By rigorously justifying the necessity of each step across image classification, super-resolution, and blind denoising, we demonstrate that DaMN achieves high performance and efficiency even surpassing models trained from scratch while maintaining minimal quality loss and immediate hardware savings. Our results open an additional opportunity in efficient deep learning deployment: **normalize during training for stability but eliminate normalization at inference for efficiency.** This decoupling unlocks hardware-friendly models without sacrificing accuracy, enabling faster, leaner, and more energy-efficient inference while preserving the benefits of normalization in the training phase.

2. Related Work

2.1. Definition and Types of Normalization Layers

A general form of normalization layers can be viewed as an approximate isotropization procedure [12]: $\phi(X) = \Sigma^{-1/2}(X - \mu \cdot \mathbf{1}^\top)$, where the statistics (e.g., mean μ and covariance Σ) and the dimensions over which they are computed are method-specific, different *NL* types are summarized in Appendix B.

2.2. Role of Normalization Layers

Although normalization layers are widely credited with enabling stable training, improved generalization, and enhanced robustness in deep networks [2, 20–22, 28], these benefits are predominantly observed during training. In this work, we focus on the post-training removal of normalization layers. Below, we provide a concise rationale for the necessity of normalization during training, sufficient to ground our subsequent analysis and consistent with established literature [4].

Normalization layers stabilize training by dynamically controlling the mean and variance of activations across layers. Consider a deep network with L layers, let μ_{in} and ν_{in} denote the mean and variance of the input signal of the first layer, respectively. Assume that each layer applies a fixed affine transformation with layer-wise shift parameter $\alpha \in \mathbb{R}$ and scale parameter $\beta \in \mathbb{R}$. Under this model, the effect compounds across layers, and the signal at the final layer has mean μ_{out} and variance ν_{out} given by: $\mu_{\text{out}} = \beta^L \mu_{\text{in}} + \alpha (\beta^L + \beta^{L-1} + \dots + \beta + 1)$, $\nu_{\text{out}} = \beta^{2L} \nu_{\text{in}}$. If $|\beta| > 1$, the network will suffer from a mean shift and exploding signals as depth increases: $\lim_{L \rightarrow \infty} \mu_{\text{out}} = \infty$, $\lim_{L \rightarrow \infty} \nu_{\text{out}} = \infty$. In the case where $|\beta| < 1$, the network will suffer from a mean shift and vanishing signals: $\lim_{L \rightarrow \infty} \mu_{\text{out}} = \frac{\alpha}{1-\beta}$, $\lim_{L \rightarrow \infty} \nu_{\text{out}} = 0$.

2.3. Strategies to avoid Normalization Layers

To mitigate these problems, four common strategies can be defined: 1) train models without *NL* with careful weight initialization schemes, 2) use alternative normalization strategies; 3) approximate *NL* with other functions, or 4) train with *NL* and remove it afterwards, which is precisely the focus of this work. We discuss these strategies below.

Training Without Normalization A recurring principles in these approaches are initialization and gradient flow control. For example, is to initialize the network close to the identity function by downscaling residual contributions at the start, effectively mimicking BatchNorm’s tendency to reduce residual branch scale in deep models [5]. Techniques such as Fixup [32], SkipInit [6], and ReZero [3] achieve this through carefully designed initialization schemes or architectural gating, preserving gradient flow from the outset and addressing the same convergence. Extending this idea, NFNets [5] combine weight standardization with per-layer adaptive gradient clipping to enable large-scale training while keeping activations and gradients well-controlled.

Alternative normalization strategies Another direction replaces LayerNorm with BatchNorm. Yao et al. [29] show that naive substitution causes instability due to unnormalized FFN blocks, and resolve this by inserting BN into FFNs, enabling stable and efficient training.

However, BN has inherent limitations in transformers. Wang et al. [26] attribute its weaker performance to Train-

ing–Inference Discrepancy (TID) and propose Regularized BN (RBN) to reduce the gap between batch and running statistics. SLAB [9] further combines LN and reparameterized BN via a dual-branch design with gradual transition.

Despite these advances, such methods require architectural changes and **training from scratch**, limiting their practicality when only pretrained models are available.

Approximations of Normalizations Another approach replaces NL with more efficient operators. DyT [34] substitutes LayerNorm with a learnable scaled activation $\tanh(\alpha x)$, enabling implicit normalization through nonlinearity. EvoNorm [17] integrates normalization and activation into a unified operator that adaptively controls signal statistics. FRN [24] applies per-channel normalization within each sample, followed by a learnable thresholded activation, avoiding batch dependence.

These methods remove reliance on batch- or feature-wise statistics, but have notable drawbacks: (1) they require **full retraining**, (2) exhibit reported failure cases (e.g., DyT failing to replace BN in some settings), and (3) may depend on custom operators not supported on all hardware, particularly for more complex designs like EvoNorm.

Post-Training Adaptation In this work, we advocate for this strategy: train a model with normalization layers (NL) and then remove them after training. This approach offers two key advantages: (1) it **avoids the computational cost of full retraining** when a high-quality pretrained model is available, and (2) it preserves the training benefits of NL , such as faster convergence and improved optimization stability.

A recent study [10] exemplifies this paradigm by demonstrating post-training removal of LayerNorm from a pretrained GPT-2 model. The authors show that sequentially removing LayerNorm modules and fine-tuning after each step allows the model to recover its original performance. However, this sequential fine-tuning process is computationally expensive. In our initial experiments on models such as Swin Transformer for image classification, even this sequential approach failed, when replacing certain normalization layers with identity mappings led to an immediate loss explosion, halting training entirely. While partial removal may be feasible in some cases, success appears highly sensitive to the removal order and still requires extensive fine-tuning.

Due to the high computational cost and sensitivity to removal order, we exclude the sequential approach from our baselines. Instead, we adopt a simpler “Naive” baseline: replacing NL with identity mappings followed by fine-tuning.

2.4. Baselines

We conclude this related work section with a brief overview of our baselines. We select at least one representative method from each of the three categories described above.

A complete list of the approaches evaluated is provided in Table 1.

Table 1. Normalization-free approaches used as baselines and ours.

Method	Core Idea
<i>Full-train approaches</i>	
Identity	Replace NL with identity and train from scratch
DyT [34]	Approximate NL with a learnable scaled tanh: $\tanh(\alpha x)$
SLAB [9]	Smooth transition to BN-like form
SkipInit [6]	Initializes residual branches with near-zero weights
Fixup [32]	Scales residual branches and biases via initialization
<i>Post-train approaches</i>	
Naive	Replace NL with identity and fine-tune
DaMN (Ours)	Initialize using statistics and smoothly transition to identity

3. Method

In this section, we first briefly outline the key steps of our procedure, followed by a detailed description and rationale for each.

Key Steps: (1) Sequentially initialize each normalization layer with an affine surrogate by matching the first and second moments of its input and output using a calibration set (20-30% of the training set in our experiments); (2) Smoothly remove the original normalization by blending it with the surrogate via a coefficient α , which is gradually annealed from 1 to 0 using a cosine schedule during fine-tuning; (3) After α reaches 0, fuse the trained affine parameters into adjacent linear layers.

The overall DaMN pipeline, which integrates these components, is illustrated in Figure 2. To support the necessity of these components, we refer to our ablation experiments in Section 4.1.

(1) Smooth Removal via α -Scheduling. Direct removal of a normalization layer $N(\cdot)$ from a trained model typically leads to severe accuracy degradation due to abrupt distribution shifts in residual streams. We address this by introducing a *smooth blending* of normalized and unnormalized paths, parameterized by $\alpha \in [0, 1]$:

$$y(\alpha) = (1 - \alpha)x + \alpha N(x). \quad (1)$$

At $\alpha = 1$ the output is fully normalized; at $\alpha = 0$ the layer behaves as an identity. We anneal α to zero over a short fine-tuning phase, ensuring a gradual distribution shift to which downstream layers can adapt. The schedule must satisfy:

- **Smoothness:** Avoid discontinuities, which may cause sudden activation shifts.
- **Monotonic decay:** Ensure consistent progression toward removal.
- **Finite horizon:** Reach $\alpha = 0$ within a target number of epochs or calibration steps.

In this paper, we explored a cosine schedule to achieve the properties mentioned above:

$$\alpha_t = \begin{cases} \frac{1}{2} (1 + \cos \frac{\pi t}{T}), & \text{if } 0 \leq t \leq T, \\ 0, & \text{if } t > T. \end{cases} \quad (2)$$

where t is the current step and T is the total decay duration. Although other scheduling strategies may be evaluated, we found the cosine schedule sufficient for our purposes and did not pursue further investigation in that direction.

(2) Moment-Matched Affine Surrogate for Normalization Removal. When blending away a normalization layer with a schedule $\alpha \rightarrow 0$, the normalized path $N(x)$ and the identity path x can have very different per-feature means and scales. Direct mixing of x and $N(x)$ causes transient bias/variance shifts that propagate through residual streams. We replace normalization with a *affine surrogate* that is moment-matched to $N(x)$ and remains learnable while α decreases.

For an input activation $x \in \mathbb{R}^d$ and a normalization $N(x)$ (LayerNorm or RMSNorm family), we define

$$y(\alpha) = (1 - \alpha) (\gamma' \odot x + \beta') + \alpha N(x), \quad (3)$$

where the parameter $\alpha \in [0, 1]$ is given by schedule 2, and the parameters $\gamma', \beta' \in \mathbb{R}^d$ are initially calibrated to match the first two moments of $N(x)$. Precisely, we take

$$\gamma' = \frac{s_N}{s_x}, \quad \beta' = m_N - \gamma' \odot m_x, \quad (4)$$

where s_x, m_x, s_N and m_N are per-feature means and standard deviations calculated for a small calibration set $\mathcal{D}_{\text{cal}}$:

$$m_x = \mathbb{E}_{x \sim \mathcal{D}_{\text{cal}}}[x], \quad s_x = \sqrt{\mathbb{E}[(x - m_x)^2] + \varepsilon}, \quad (5)$$

$$m_N = \mathbb{E}[N(x)], \quad s_N = \sqrt{\mathbb{E}[(N(x) - m_N)^2] + \varepsilon}. \quad (6)$$

For collecting variance we found important to use Welford’s algorithm [27] to avoid numerical error.

Importantly, at the final stage (γ', β') are fused into adjacent layers.

(3) Sequential Calibration. Since deep models typically consist of multiple layers arranged in sequence, the removal of one normalization layer $N(\cdot)_{(l)}$ can alter the mean, standard deviation, and overall distribution of the subsequent layers $N(\cdot)_{(l+1)}, N(\cdot)_{(l+2)}$, etc.. As a result, surrogate parameters (γ', β') for these affected layers must be recomputed based on the updated information from the removed $N(\cdot)_{(l)}$. To maintain consistency and accuracy, this recomputation is performed sequentially, after removing $N(\cdot)_{(l)}$, the updated information is propagated forward

through the network up to l layer to compute the parameters for $N(\cdot)_{(l+1)}$. This procedure is outlined in Algorithm 1 in Appendix C.

Our experiments in Section 4.1 demonstrate that sequential calibration of surrogate parameters leads to stable fine-tuning and results; in contrast, omitting this process results in a dramatic increase in model loss and quality drop.

4. Experimental Evaluation & Ablation Study

4.1. Ablation Study

To investigate the role of each component of the DaMN method, we conduct experiments evaluating models performance when these components are absent or incorrectly computed.

4.1.1. Impact of Affine Surrogate Transforms

To validate the effectiveness of Affine Surrogate Transforms in mitigating the negative impact of removing normalization layers NL , we conduct an ablation study by progressively removing several NL from Swin-B, both with and without applying the transforms. We observe that removing even the first normalization layer leads to a dramatic degradation in model performance, and this loss cannot be fully recovered through fine-tuning alone.

Figure 3 illustrates this effect for the first NL and the first downsampling NL in Swin-B (similar behavior is observed for other NL s in Swin-B and also in other models). In this experiment, we do not perform fine-tuning in order to clearly highlight the impact of Affine Surrogate Transforms. The parameter α represents a scaling coefficient applied to a normalization layer: $\alpha = 1$ corresponds to the original model with the full normalization layer, while $\alpha = 0$ corresponds to its complete removal. As can be seen from Figure 3, in the absence of Affine Surrogate Transforms (blue line), the model’s performance collapses to nearly 0% as $\alpha \rightarrow 0$. In contrast, when the transforms are applied (red line in Figure 3), the model retains nearly the same level of performance, even without fine-tuning.

The blue line in Figure 5, representing naive removal (without a surrogate and with $\alpha = 0$), offers additional insight into the importance of applying affine surrogate transforms with a proper estimation of the affine parameters. It shows explosive growth in variance that leads to saturation of the FP32 data type, indicated by NaN values marked with blue triangles – this signals that feature maps amplitudes are being amplified too rapidly.

We also observed that the performance of the calibrated model is highly sensitive to the accuracy of the estimated statistics. To illustrate the impact of accurate statistical estimation, we conduct a controlled perturbation experiment on the affine parameters of the surrogate normalization layers in MambaIR at scale $\times 2$. Figure 4 demonstrates that

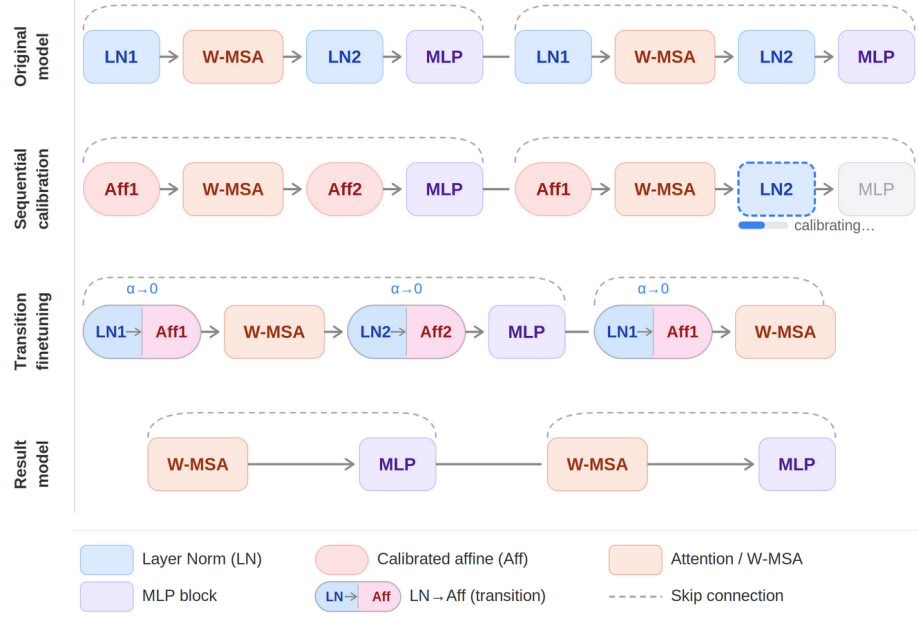


Figure 2. DaMN pipeline consists of three main steps: (1) Sequentially initialize each normalization layer with an affine surrogate by matching the first and second moments; (2) Smoothly remove the original normalization with fine-tuning; (3) Fuse the trained affine parameters into adjacent linear layers.

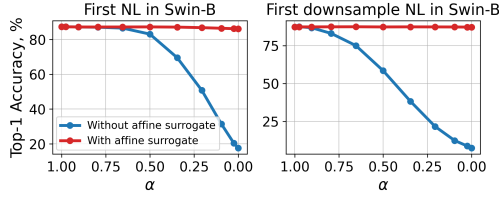


Figure 3. Impact of affine surrogate initialization on performance during normalization removal (α from 1 to 0). We adjust α for the first or first downsampling normalization layers in Swin-B, keeping other layers fixed. Red shows surrogate application, blue shows no surrogate. No fine-tuning is performed. Moment-matched surrogates enable smooth transitions with minimal performance drop, consistent across other layers and models.

even slight perturbations in γ' and β' can lead to significant performance degradation, emphasizing the importance of numerically stable and precise moment estimation during calibration.

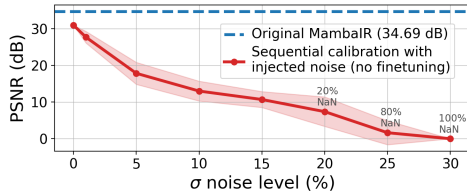


Figure 4. Effect of noisy affine parameters on MambaIR performance at scale $\times 2$. Gaussian noise is injected into γ' and β' to simulate estimation errors, σ denotes relative to γ' and β' additive noise with zero mean. Even small perturbations cause notable accuracy drops, underscoring the need for stable statistical estimation during calibration. Noise injection details are in Appendix E.

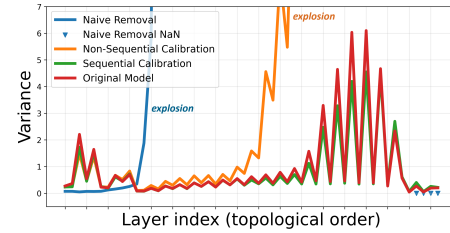


Figure 5. Comparison of output variances across different normalization removal strategies for the Swin-B architecture: naive removal without calibration, non-sequential calibration, and sequential calibration. Sequential calibration allows maintaining consistency throughout the network.

ing the calibration process. Moreover, we observed that if the affine parameters are estimated incorrectly, which corresponds a high noise level, we get NaN during the forward propagation (even using fp32 data type). A probability of catching NaN for a high noise level is plotted above the dots. For example, 80% of attempts fail for 25% noise level. To mitigate such numerical instability during the online estimation of moments, we employ the Welford algorithm [27] for computing means and variances.

4.1.2. Sequential calibration

To validate the importance of sequential calibration in stabilizing the model when removing *NL*, we compare three strategies on the Swin-B model: (1) *naive NL removal* without any calibration, (2) *non-sequential calibration*, where all normalization layers are removed and affine surrogates are calibrated independently in one step, and (3) *sequen-*

tial calibration, where surrogates are inserted and calibrated layer by layer in a topological order.

With the non-sequential calibration, the statistics of intermediate activations rapidly drift from those of the original model. This drift accumulates across depth, leading to significant increases in variance in downstream layers (orange curve in Figure 5). In contrast, sequential calibration ensures that each surrogate compensates for both its own normalization removal and accumulated distortions from earlier layers. As a result, the variance of layer outputs stays close to that of the original model across all layers (green curve in Figure 5), in contrast to both naive removal and non-sequential calibration, where variance diverges quickly and some layers produce unstable (NaN) activations.

We also find that without sequential calibration, the fine-tuning with smooth removal also behaves less robust and gives worse quality (see Appendix D).

4.1.3. Smooth Removal

To evaluate the effectiveness of the Smooth Removal mechanism in the DaMN framework, we conduct an ablation study by removing normalization layers with and without applying the smoothing strategy. Smooth removal is designed to prevent large derivative jumps that could lead to activation explosions, allowing downstream layers to adapt progressively. This ensures a more stable training process, while the monotonic and finite properties of the removal process guarantee a controlled and efficient transition toward complete normalization layer removal.

We observe that removing normalization layers using a smoothing strategy consistently results in better performance compared to abrupt removal. As illustrated in Table 2, replacing 24 out of 53 normalization layers in the Swin-B model with identity layers in a smooth manner leads to only a 0.2% drop in top-1 accuracy, while immediate replacement, or so-called naive removal, completely destroys the model. Specifically, the smooth transition approach helps avoid instability during training, leading to improved convergence and the reduced occurrence of loss spikes. Moreover, the performance degradation is significantly minimized compared to the abrupt removal method. This happens across all tried models and tasks. A detailed comparison between naive removal and DaMN is presented in Tables 4, 5, and 6 for various tasks and models.

Table 2. Top-1 Accuracy (%) on ImageNet for Swin-B model. Smooth removal (except the sequential calibration and affine surrogate transforms) vs. naive removal.

Removed LayerNorms	Using only smoothing		Naive removal	
	Top-1 Acc. (%)	Epochs	Top-1 Acc. (%)	Epochs
0 (Baseline)	83.418	300	83.418	300
24 of 53	83.246	450	NaN	< 1

An important point to discuss is how convergence de-

Table 3. Influence of smoothing durations on training convergence for different models.

Model	Diverged	Converged
Swin-B	0–10 epochs	100 epochs
ViT-B	0–5 epochs	100 epochs
GPT-2	0–600 iterations	800–1000 iterations

pends on the smoothing duration – that is, the number of iterations during which the parameter α decreases from 1 to 0. If the duration is very short, the behavior resembles that of a naive removal strategy. On the other hand, if the duration is not increased enough, the model may diverge, as the transition from NL to the identity occurs too rapidly. However, if α is relaxed over a sufficiently long period, the model tends to converge to a high-quality solution. As illustrated in Table 3, the smoothing duration cannot be too short: the first column shows durations that lead to divergence, while the second column presents durations that are sufficient for convergence.

4.2. Main Results

To evaluate the effectiveness of DaMN, we conduct experiments using Transformer architectures across a wide range of tasks and domains. In each experiment, we remove the standard Layer Normalization or RMSNorm from the original models using the DaMN framework and compare it with NL removal/replacement methods described in Table 1. Both the modified and original versions of the models are trained and tested following their official open-source repositories.

For DyT, Fixup, and SkipInit, we adopt the implementations provided in their official repositories without modification. The SLAB method [9] introduces two components: Simplified Linear Attention (SLA) and Progressive Batch Normalization (PBN). Since our goal is to evaluate normalization removal rather than attention redesign, we retain only the Progressive Batch Normalization component and remove the SLA module from the architecture. We note that the original SLAB paper does not report ablation results for PBN alone, therefore our implementation isolates the contribution of this component for a fair comparison with other normalization-related approaches.

Image Super-Resolution We evaluate DaMN on three representative transformer models for image super-resolution: SwinIR [15], MambaIR [8], and SRFormer [33]. All models were trained using their original repositories with DF2K dataset [16]. Reproduced metrics closely match the papers. About 40% of training data (without augmentation) is used for sequential calibration. PSNR/SSIM results on Set5, Set14, BSD100, Urban100, and Manga109 ($\times 2$) are reported in Table 4; $\times 3/\times 4$ are in Appendix G. We compare: Original (reproduction), Identity (trained without NL), naive NL removal, DyT, and DaMN. All removal methods start from the reproduced model unless noted. We

Table 4. PSNR/SSIM at scale $\times 2$ for three transformer-based models on standard SR benchmarks. The best results are highlighted in **red**, the second-best in **blue** in this table and also in the tables below.

Model	Method	Set5		Set14		BSD100		Urban100		Manga109	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
SwinIR	Original	38.40	0.9618	34.44	0.9243	32.55	0.9030	33.81	0.9427	39.92	0.9797
	Identity	38.24	0.9590	33.95	0.9210	31.90	0.8900	33.00	0.9300	39.00	0.9732
	Naive Removal	37.80	0.9523	33.52	0.9171	31.45	0.8826	32.75	0.9228	38.86	0.9671
	DyT	38.34	0.9603	34.49	0.9252	32.45	0.9025	33.74	0.9419	39.76	0.9777
	SLAB	38.27	0.9597	34.30	0.9230	32.37	0.9000	33.60	0.9410	39.64	0.9720
	DaMN (ours)	38.30	0.9605	34.41	0.9240	32.41	0.9019	33.68	0.9415	39.70	0.9736
MambaIR	Original	38.54	0.9624	34.69	0.9263	32.55	0.9046	34.12	0.9443	40.26	0.9804
	Identity	Training diverged right from the start									
	Naive Removal	Training diverged right from the start									
	DyT	38.43	0.9622	34.35	0.9238	32.50	0.9037	33.69	0.9414	39.88	0.9799
	SLAB	38.41*	0.9622*	34.31*	0.9240*	32.48*	0.9040*	33.63*	0.9409*	39.80*	0.9794*
	DaMN (ours)	38.49	0.9625	34.52	0.9253	32.55	0.9046	33.93	0.9433	40.12	0.9801
SRFormer	Original	38.58	0.9630	34.79	0.9270	32.60	0.9050	34.35	0.9461	40.24	0.9806
	Identity	38.41*	0.9620*	34.46*	0.9255*	32.48*	0.9035*	33.68*	0.9418*	39.88*	0.9799*
	Naive Removal	Training diverged right from the start									
	DyT	38.44	0.9622	34.46	0.9250	32.49	0.9037	33.74	0.9423	39.84	0.9796
	SLAB	38.53	0.9614	34.78	0.9258	32.54	0.9047	34.27	0.9451	40.22	0.9801
	DaMN (ours)	38.42	0.9622	34.41	0.9249	32.55	0.9040	34.13	0.9452	40.10	0.9802

report performance of our reproduction of the model (denoted by Original), training from scratch without *NL* (denoted by Identity), naive removal of *NL*, DyT, and the proposed DaMN method. We start all *NL* removal procedures from the reproduced model, unless stated otherwise.

SwinIR (74 *NLs*). Training without *NL* causes minor degradation (-0.2 to -0.9 dB). Naive removal performs worse (-0.6 to -1.1 dB). DyT matches or slightly exceeds baseline (~ -0.1 dB on average). DaMN achieves comparable or better results than DyT/SLAB with only 30% fine-tuning, fully removing *NLs* (-0.1 to -0.2 dB).

MambaIR (110 *NLs*). Training without *NL* or naive removal leads to gradient explosion and divergence despite extensive tuning, confirming the importance of normalization. DaMN removes *NLs* with 30% training budget and α -scheduling (80% of training), achieving near-original performance and sometimes outperforming DyT (e.g., 33.93 vs 33.69 dB on Urban100). SLAB is unstable and diverges by the end of training.

SRFormer (74 *NLs*). Training with identity *NL* initially works (~ -0.5 dB) but later collapses; no stable configuration was found. DaMN with 30% retraining and α -scheduling (80%) matches baseline performance. SLAB approaches baseline but requires training from scratch.

Blind Image Denoising We perform our evaluation with real-world blind image denoising using the Restormer [30] model, trained on the SIDD dataset [1] and tested on SIDD benchmark images. The model was reproduced using the

Table 5. Blind denoising results on SIDD dataset. * denotes that further finetuning of the method has failed.

Method	SIDD	
	PSNR (dB)	SSIM
Original	40.02	0.9603
Identity	38.26*	0.9513*
DyT	39.98	0.9601
DaMN	39.96	0.9601

original Restormer repository² to match the reported results. Nearly 40% of the training dataset (without augmentations) was used for calibrating statistics during surrogate initialization. Fine-tuning with DaMN required 30% of the epochs from the original training procedure, with alpha scheduling applied over 80% of the fine-tuning process. DaMN and DyT results are close to the original reproduction results.

It is worth noting the behavior with identity. Identity from scratch performed well during the first 7000 iterations achieving the loss values comparable to reproduction: $5.78e-3$ for identity in comparison to $5.82e-3$. However, after this point, the loss rapidly became NaN, indicating a failure in training, and the model became clearly stuck. The results obtained during the first 4000 iterations are summarized in Table 5.

Image Classification We evaluate DaMN on the

²<https://github.com/swz30/Restormer>

Table 6. ImageNet Top-1 Accuracy (%) for Different Normalization Removal Strategies. * denotes that further finetuning of the method has failed. † denotes the affine surrogate transforms and the smooth removal, except the sequential calibration.

Method	Swin-B	ViT-B	ConvNeXt
Original	83.4	83.0	83.1
Identity	50.3*	53.5*	42.2*
Naive Removal	Diverged	Diverged	Diverged
SkipInit [6]	72.3	74.1	73.1
Fixup [31]	77.5	77.2	73.7
DyT [34]	81.3	82.5	82.3
SLAB[9]	81.6	82.5	82.5
DaMN (ours)	87.2	82.6	82.7

ImageNet-1K [23] classification task using three representative transformer-based models: Swin-B [18], ViT-B [7], and ConvNeXt-B [19]. The experimental setup and all the technical details can be found in Appendix H.

Our method achieves results that are comparable to or better than DyT, SLAB. But all methods incur high performance degradation, up to 2% drop, from the reproduced original models. Retain some LN is one of the ways to reduce this gap.

Other full-train approaches, requiring retraining from scratch, such as SkipInit and Fixup improved upon the Identity initialization (72–77% accuracy), yet they still fell short of the original models’ performance (83%).

LLM optimization We further evaluate our approach on language models. We use the popular and modification-friendly Modded-NanoGPT [14] to train a 128M-parameter model on the FineWeb dataset due to its simplicity and low computational demands.

Applying our method, we successfully remove 30 out of 45 normalization layers, retaining them only in the final two decoder layers with further removal leading to model divergence. We observe a slight improvement in validation loss, from 3.29 (original) to 3.22 (DaMN), compared to 3.29 → 3.27 for DyT. We speculate that this performance gain may be due to the original model not having fully converged. Additionally, we remove half of normalization layers of Qwen2.5-1.5B finetuned on OpenMathInstruct-2 [25], getting minor performance drop 29.6 → 28.8 (Pass@1) on MATH500 benchmark [11]. This demonstrates that the approach is applicable to models finetuned for downstream tasks.

While these results are preliminary and require further investigation, they suggest a promising direction for reducing the number of *NL* in large language models without sacrificing performance. We provide a full description of the methodology and experimental details in Appendix I.

4.3. Efficiency gains

For efficiency gains we evaluate latency and energy consumption among various GPUs and models. In this section, we review this results briefly, for more detailed analysis of performance gains across a range of GPUs (A100, A40, H100, V100 and RTX 2080Ti) and data types (BF16, FP16 and FP32) please see Appendix A. Figure 1 presents the relative improvements in inference time and energy consumption of DyT and DaMN compared to the original models with normalization layers. These results are evaluated on the A100 GPU. As we can see, **DyT shows valuable improvements** (blue bars). For instance, replacing the normalization layers with DyT for the Restormer model [30] gives 13.7% and 14.3%, respectively. Also, we note that DaMN achieves the most significant gains by entirely removing normalization layers. **DaMN outperforms DyT** by 1-6% in terms of inference time and energy consumption (red bars), resulting in faster and more energy-efficient configurations.

5. Conclusion

This paper has proved the efficiency of the new approach (DaMN): normalization during training for convergence, and elimination of normalization at inference to improve efficiency. It is shown that the computational cost of normalization layers in transformer-based DNNs is extremely high. Removing these layers reduces inference time by up to **18%**, and energy consumption by up to **17%**, on NVIDIA GPUs. This makes DaMN a valuable tool for cloud computing scenarios. Furthermore, we expect that the benefits on neural network accelerators for embedded devices, such as smartphones, will be even greater.

6. Limitations

While DaMN demonstrates that normalization layers can be removed from pretrained transformer models with minimal performance degradation, several limitations remain. First, the method requires a calibration stage and a fine-tuning phase, which introduces additional computational cost compared to direct deployment of the pretrained model. Second, the effectiveness of the affine surrogate relies on accurate estimation of activation statistics; inaccurate moment estimation may lead to instability during removal, particularly in very deep networks. Third, our experiments indicate that partial normalization removal may benefit in terms of accuracy preservation.

Acknowledgment. The work was supported by the grant for research centers in the field of AI provided by the Ministry of Economic Development of the Russian Federation in accordance with the agreement 000000C313925P4F0002 and the agreement №139-10-2025-033.

References

- [1] Abdelrahman Abdelhamed, Stephen Lin, and Michael S Brown. A high-quality denoising dataset for smartphone cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1692–1700, 2018. 7
- [2] Muhammad Awais, Md Tauhid Bin Iqbal, and Sung-Ho Bae. Revisiting internal covariate shift for batch normalization. *IEEE Transactions on Neural Networks and Learning Systems*, 32(11):5082–5092, 2020. 2
- [3] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W. Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth, 2020. 2
- [4] Garrett Bingham and Risto Miikkulainen. Autoinit: Analytic signal-preserving weight initialization for neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6823–6833, 2023. 2
- [5] Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization, 2021. 2
- [6] Soham De and Samuel L. Smith. Batch normalization biases residual blocks towards the identity function in deep networks, 2020. 2, 3, 8
- [7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 8
- [8] Hang Guo, Jinmin Li, Tao Dai, Zhihao Ouyang, Xudong Ren, and Shu-Tao Xia. Mambair: A simple baseline for image restoration with state-space model, 2024. 6
- [9] Jialong Guo, Xinghao Chen, Yehui Tang, and Yunhe Wang. Slab: Efficient transformers with simplified linear attention and progressive re-parameterized batch normalization, 2024. 3, 6, 8
- [10] Stefan Heimersheim. You can remove gpt2’s layernorm by fine-tuning, 2024. 3
- [11] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 8
- [12] Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization towards efficient whitening. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4874–4883, 2019. 1, 2
- [13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. 1
- [14] Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fern-bear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the nanogpt baseline, 2024. 8
- [15] Jingyun Liang, Jiezhong Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer, 2021. 6
- [16] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 136–144, 2017. 6
- [17] Hanxiao Liu, Andrew Brock, Karen Simonyan, and Quoc V. Le. Evolving normalization-activation layers, 2020. 3
- [18] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021. 8
- [19] Zhuang Liu, Hanzhi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022. 8
- [20] Ekdeep S Lubana, Robert Dick, and Hidenori Tanaka. Beyond batchnorm: Towards a unified understanding of normalization in deep learning. *Advances in Neural Information Processing Systems*, 34:4778–4791, 2021. 1, 2
- [21] Kaifeng Lyu, Zhiyuan Li, and Sanjeev Arora. Understanding the generalization benefit of normalization layers: Sharpness reduction. *Advances in Neural Information Processing Systems*, 35:34689–34708, 2022.
- [22] Hanyang Peng, Yue Yu, and Shiqi Yu. Re-thinking the effectiveness of batch normalization and beyond. *IEEE transactions on pattern analysis and machine intelligence*, 46(1): 465–478, 2023. 2
- [23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. 8
- [24] Saurabh Singh and Shankar Krishnan. Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks, 2020. 3
- [25] Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data, 2024. 8
- [26] Jiayi Wang, Ji Wu, and Lei Huang. Understanding the failure of batch normalization for transformers in nlp, 2022. 2
- [27] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962. 4, 5
- [28] Yuxin Wu and Justin Johnson. Rethinking” batch” in batch-norm. *arXiv preprint arXiv:2105.07576*, 2021. 2
- [29] Zhuliang Yao, Yue Cao, Yutong Lin, Ze Liu, Zheng Zhang, and Han Hu. Leveraging batch normalization for vision transformers. In *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 413–422, 2021. 2

- [30] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Restormer: Efficient transformer for high-resolution image restoration. In *CVPR*, 2022. 1, 7, 8
- [31] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 12369–12380, 2019. 8
- [32] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization, 2019. 2, 3
- [33] Yupeng Zhou, Zhen Li, Chun-Le Guo, Li Liu, Ming-Ming Cheng, and Qibin Hou. Srformerv2: Taking a closer look at permuted self-attention for image super-resolution, 2024. 6
- [34] Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization, 2025. 1, 3, 8