DaMN: Deleting and Migrating Normalization Layers from Transformers Supplementary Material

Alexey Ryabykin[1] Irina Zhelavskaya[1] Egor Shvetsov[2] Alexey Rukhovich[1]
Nikita Okhotnikov[1] Artem Khrapov[1] Evgeny Burnaev[2]
Vladimir Mikhailovich Kryzhanovskiy[1]
[1]AI Foundation & Algorithm Lab [2]APPLIED-AI

Appendix A. Efficiency gains

To evaluate the computational benefits of the DaMN approach, we measured the inference time and energy consumption of Swin-B, ViT-B, ConvNeXt, MambaIR, SRFormer, and Restormer, across a range of GPU architectures: A100, H100, V100, RTX 2090 Ti, and RTX 4090. These measurements were conducted using the PYNVML Python library [1].

The experimental setup included 50 to 100 inference iterations per model (kept consistent across models), with each iteration repeated five times and a 10-second cooldown period between repetitions to ensure thermal stability. Compile mode in torch was not used. For classification tasks, we used a batch size of 512 and an input image size of 224. For super-resolution (SR) models, we applied an input size of 128 with a batch size of 8 for MambaIR and SRFormer, while Restormer was evaluated with an input size of 384.

Figures S1-S4 present the relative improvements in inference time and energy consumption of DyT and DaMN compared to the original models with normalization layers, computed as $1 - M/LN$, where M denotes inference time/energy consumption of the considered method (DyT or DaMN) and LN denotes the corresponding metric for the original model with layer normalization. These results are evaluated on the H100, A40, V100, RTX 2090 Ti, and RTX 4090 GPU (results for A100 are shown in Figure 1 in the main text of the paper). As shown, DaMN achieves the most significant gains across all GPU types by entirely removing normalization layers, resulting in the fastest and most energy-efficient configurations. Table S1 provides all results compiled together. Please note that in some cases, negative values appear in the performance metrics, indicating increased inference time and higher resource consumption relative to the baseline model with normalization layers (particularly for SRFormer in bf16 mode). A detailed investigation into mitigating or resolving these effects is beyond the scope of this paper.

[1] https://pypi.org/project/nvidia-ml-py/

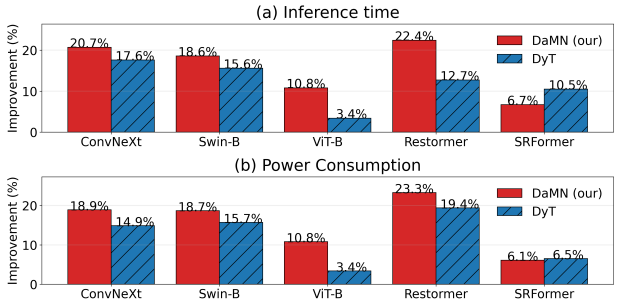

Fig. S1. a) Inference time and b) energy consumption improvements of DyT and DaMN compared to the original model with normalization layers, evaluated on an **H100 GPU**.

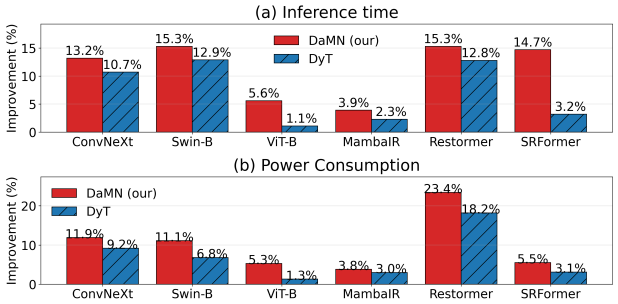

Fig. S2. a) Inference time and b) energy consumption improvements of DyT and DaMN compared to the original model with normalization layers, evaluated on an **V100 GPU**.

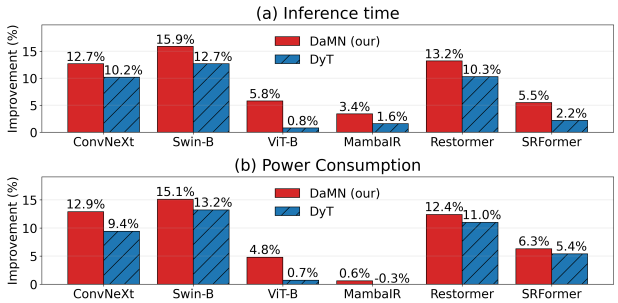

Fig. S3. a) Inference time and b) energy consumption improvements of DyT and DaMN compared to the original model with normalization layers, evaluated on an **RTX2080Ti GPU**.

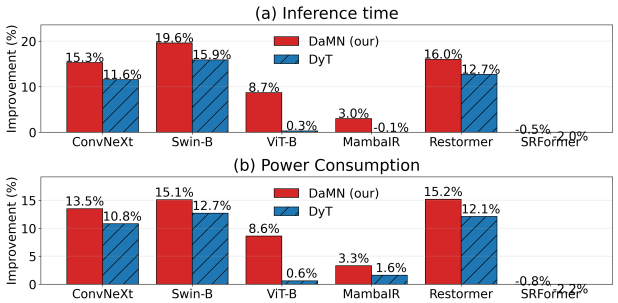

Fig. S4. a) Inference time and b) energy consumption improvements of DyT and DaMN compared to the original model with normalization layers, evaluated on an **RTX4090 GPU**.

Table S1. Relative change in inference Time, % and Power, % across hardware for different models and methods (higher is better; negative values indicate regressions). * denotes a different batch size used on RTX 2080 Ti due to its memory limits. Note that bf16 is not available on V100 and RTX 2080 Ti.

Precision	Model	# NLs	Batch	Input	Method	A100		H100		V100		RTX 2080 Ti		RTX 4090		Mean over GPUs	
						Time, % ↑	Power, % ↑	Time, % ↑	Power, % ↑	Time, % ↑	Power, % ↑	Time, % ↑	Power, % ↑	Time, % ↑	Power, % ↑	Time, % ↑	Power, % ↑
bf16	swin_base_patch4_window7_224	53	512	224	tanh	15.0	14.5	7.7	8.8	n/a	n/a	n/a	n/a	16.3	14.2	13.0	12.5
bf16	swin_base_patch4_window7_224	53	512	224	identity	**18.3**	**16.6**	**9.0**	**10.5**	n/a	n/a	n/a	n/a	**20.0**	**16.5**	**15.8**	**14.5**
bf16	vit_base_patch16_224	22	512	224	tanh	1.4	1.8	4.0	4.1	n/a	n/a	n/a	n/a	0.1	2.2	1.8	2.7
bf16	vit_base_patch16_224	22	512	224	identity	**8.3**	**8.0**	**12.1**	**11.2**	n/a	n/a	n/a	n/a	**8.9**	**8.1**	**9.8**	**9.1**
bf16	convnext_base	36	512	224	tanh	12.4	10.3	12.9	14.9	n/a	n/a	n/a	n/a	11.5	11.1	12.3	12.1
bf16	convnext_base	36	512	224	identity	**13.9**	**13.5**	**18.6**	**18.1**	n/a	n/a	n/a	n/a	**15.3**	**13.9**	**15.9**	**15.2**
bf16	mambair	110	8	128	tanh	3.2	3.3	n/a	n/a	n/a	n/a	n/a	n/a	-0.2	1.3	1.5	2.3
bf16	mambair	110	8	128	identity	**4.1**	**5.1**	n/a	n/a	n/a	n/a	n/a	n/a	**1.7**	**3.3**	**2.9**	**4.2**
bf16	restormer	80	8	384	tanh	10.2	10.9	11.7	11.9	n/a	n/a	n/a	n/a	11.1	10.7	11.0	11.2
bf16	restormer	80	8	384	identity	**11.5**	**12.4**	**14.2**	**13.9**	n/a	n/a	n/a	n/a	**14.0**	**13.5**	**13.2**	**13.3**
bf16	srformer	74	8	128	tanh	-10.4	-6.4	-5.2	**2.1**	n/a	n/a	n/a	n/a	-1.4	-1.7	-5.7	**-2.0**
bf16	srformer	74	8	128	identity	**-8.2**	**-4.9**	**5.5**	-0.7	n/a	n/a	n/a	n/a	**-0.2**	**-0.4**	**-1.0**	**-2.0**
fp16	swin_base_patch4_window7_224	53	512	224	tanh	15.1	13.5	15.6	15.7	12.9	6.8	12.7	13.2	15.9	12.7	14.5	12.4
fp16	swin_base_patch4_window7_224	53	512	224	identity	**17.0**	**16.7**	**18.6**	**18.7**	**15.3**	**11.1**	**15.9**	**15.1**	**19.6**	**15.1**	**17.3**	**15.3**
fp16	vit_base_patch16_224	22	512	224	tanh	2.5	1.9	3.4	3.4	1.1	1.3	0.8	0.7	0.3	0.6	1.6	1.6
fp16	vit_base_patch16_224	22	512	224	identity	**8.6**	**7.5**	**10.8**	**10.8**	**5.6**	**5.3**	**5.8**	**4.8**	**8.7**	**8.6**	**7.9**	**7.4**
fp16	convnext_base	36	512	224	tanh	10.9	11.8	17.6	14.9	10.7	9.2	10.2	9.4	11.6	10.8	12.2	11.2
fp16	convnext_base	36	512	224	identity	**13.4**	**13.6**	**20.7**	**18.9**	**13.2**	**11.9**	**12.7**	**12.9**	**15.3**	**13.5**	**15.0**	**14.2**
fp16	mambair	110	8	128	tanh	1.8	3.8	n/a	n/a	2.3	3.0	1.6	-0.3	-0.1	1.6	1.4	2.0
fp16	mambair	110	8	128	identity	**3.6**	**5.5**	n/a	n/a	**3.9**	**3.8**	**3.4**	**0.6**	**3.0**	**3.3**	**3.5**	**3.3**
fp16	restormer	80	8	384	tanh	13.7	14.3	12.7	19.4	12.8	18.2	10.3	11.0	12.7	12.1	12.4	15.0
fp16	restormer	80	8	384	identity	**17.9**	**16.9**	**22.4**	**23.3**	**15.3**	**23.4**	**13.2**	**12.4**	**16.0**	**15.2**	**16.9**	**18.2**
fp16	srformer	74	8	128	tanh	1.5	1.8	**10.5**	**6.5**	3.2	3.1	2.2	5.4	-2.0	-2.2	3.1	2.9
fp16	srformer	74	8	128	identity	**3.9**	**3.9**	6.7	6.1	**14.7**	**5.5**	**5.5**	**6.3**	**-0.5**	**-0.8**	**6.1**	**4.2**
fp32	swin_base_patch4_window7_224	53	512, *128	224	tanh	0.7	-1.0	0.1	0.1	1.0	-1.2	0.3	0.1	-0.8	-0.3	0.3	-0.5
fp32	swin_base_patch4_window7_224	53	512, *128	224	identity	**1.8**	**1.3**	**2.2**	**2.2**	**2.9**	**-0.6**	**2.7**	**3.6**	**3.0**	**2.8**	**2.5**	**1.9**
fp32	vit_base_patch16_224	22	512	224	tanh	3.9	0.4	0.4	0.3	0.3	0.0	0.2	0.7	0.8	0.5	1.1	0.4
fp32	vit_base_patch16_224	22	512	224	identity	**4.7**	**0.9**	**1.6**	**1.5**	**1.3**	**0.6**	**1.5**	**4.8**	**2.8**	**2.6**	**2.4**	**2.1**
fp32	convnext_base	36	512, *128	224	tanh	0.4	**-0.9**	1.0	1.1	1.0	-0.8	1.2	0.0	-0.1	0.5	0.7	0.0
fp32	convnext_base	36	512, *128	224	identity	**1.2**	-1.1	**2.1**	**3.0**	**2.6**	**0.1**	**3.1**	**2.6**	**3.6**	**3.5**	**2.5**	**1.6**
fp32	mambair	110	8	128	tanh	2.4	1.7	n/a	n/a	1.7	2.9	1.2	1.5	-0.2	1.9	1.3	2.0
fp32	mambair	110	8	128	identity	**3.7**	**2.7**	n/a	n/a	**3.5**	**4.2**	**3.1**	**3.0**	**2.7**	**3.3**	**3.3**	**3.3**
fp32	restormer	80	8, *4	384	tanh	3.3	12.6	5.5	5.7	4.5	10.7	4.2	4.3	5.8	4.9	4.7	7.6
fp32	restormer	80	8, *4	384	identity	**5.0**	**14.6**	**9.1**	**8.9**	**5.5**	**21.2**	**6.9**	**6.7**	**9.0**	**7.6**	**7.1**	**11.8**
fp32	srformer	74	8	128	tanh	7.1	0.9	**-2.4**	**-10.5**	0.9	-1.5	0.3	-0.4	0.9	-0.4	1.4	-2.2
fp32	srformer	74	8	128	identity	**7.4**	**2.4**	-2.8	-10.9	**2.7**	**-0.1**	**1.7**	**1.5**	0.0	**2.1**	**1.5**	**-0.6**
			Mean tanh (Time, % / Power, %,)			5.3	5.3	6.4	6.6	4.4	4.4	3.8	3.8	4.6	4.5	4.9	5.1
			Mean identity (Time, % / Power, %,)			**7.6**	**7.5**	**10.1**	**9.0**	**7.1**	**7.4**	**6.3**	**6.2**	**7.9**	**7.3**	**7.9**	**7.6**

Appendix B. Normalization layers types

A general form of normalization layers can be viewed as an approximate isotropization procedure [6]:

$$\phi(X) = \Sigma^{-1/2}(X - \mu \cdot \mathbf{1}^\top), \tag{1}$$

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 Table S2. The key distinction among popular normalization methods lies in how mean μ and covariance Σ are computed and the axes over which they are are computed. Batch Normalization [7] computes statistics over the batch and spatial/token dimensions for each channel, a design that excels in CNNs with large batches, yielding faster convergence and higher learning rate tolerance. Layer Normalization (LN) [1] instead normalizes each token's features independently within each sample, making it robust to small batches and essential for stable Transformer training [3, 11]. RMSNorm [18] further simplifies LN by skipping mean subtraction and normalizing solely by the root mean square, preserving scale invariance with reduced computation. In practice, BN dominates convolutional architectures, LN is standard in Transformers, and RMSNorm has gained traction in recent large language models (e.g., LLaMA [14], Mistral [8], Qwen [2]).

Table S2. A general form of normalization layers can be seen as $\phi(X) = \Sigma^{-1/2}(X - \mu \cdot \mathbf{1}^\top)$ with method-specific statistics defined below.

Normalization	**Axes**	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$	**Notes**
BatchNorm [7]	Batch + spatial	per channel	$\sqrt{\text{Var}(X) + \epsilon}$ per channel	Fixed mean and variance
InstanceNorm[15]	Spatial, per sample & channel	over spatial dim.	$\sqrt{\text{Var}(X) + \epsilon}$ over spatial dim.	On-the-fly per sample
LayerNorm [1]	Feature dim.	over last dim.	$\sqrt{\text{Var}(X) + \epsilon}$ over same	On-the-fly per sample
GroupNorm [17]	Grouped channels + spatial	over group + spatial	$\sqrt{\text{Var}(X) + \epsilon}$ over same	On-the-fly per sample
RMSNorm [18]	Feature dim.	no centering	$\sqrt{\text{mean}(X^2) + \epsilon}$	On-the-fly per sample

Appendix C. Sequential Calibration and Removal of Normalization Layers: Algorithm

Accurate and numerically stable estimates of mean and variance are crucial for fitting the affine surrogates in our calibration procedure. For all computer vision experiments (image restoration and image classification) we used Welford's algorithm [16] from the beginning and it consistently turned out to be the most reliable choice. For LLMs we initially switched to a simpler estimator that collects the mean and mean of squares for each feature,

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad q = \frac{1}{n}\sum_{i=1}^{n} x_i^2, \qquad v = q - \mu^2, \tag{2}$$

and accumulating originally was in `bf16`. This estimator behaved very poorly in the LLM setting: such variance computation suffers from catastrophic cancellation when q and μ^2 are close, so rounding noise in the collected statistics leads to large errors in v and, consequently, unstable affine surrogates and activation explosions. The noise injection procedure we replicated by introducing the additive noise in the statistics (which you can find in the Section 4.1) and it showed significance of accurate statistic collection.

Welford's algorithm directly addresses numerical instability and we apply this per feature in a vectorized, batched form (see Algorithm 1). After observing severe instability with the two-moment estimator on LLMs, we switched GPT-2 calibration to Welford, which yielded stable statistics and allowed us to safely delete about 70% of normalization layers. For Qwen2.5, we kept the two-moment estimator but promoted both $\sum x_i$ and $\sum x_i^2$ to `fp64`, which significantly reduced numerical issues at the cost of slightly higher calibration overhead.

Algorithm 1 Sequential Calibration and Removal of Normalization Layers

Require: Model f, topologically ordered normalization layers $\mathcal{L}$, calibration data $\mathcal{D}$

1: Initialize f in evaluation mode; freeze all parameters

2: **for each** layer $\ell \in \mathcal{L}$ in forward (topological) order **do**

3: **Calibrate** ℓ**:** Stream $\mathcal{D}$; for each mini-batch B of size m: Welford algorithm [16] to compute per-feature batch mean/var for *inputs* and *LN outputs*:

$$\delta_z \leftarrow \bar{z}_B - \mu_z, \quad n_z^+ \leftarrow n_z + m$$
$$\mu_z^+ \leftarrow \mu_z + \delta_z \frac{m}{n_z^+}$$
$$M_{2,z}^+ \leftarrow M_{2,z} + m\, s_{z,B}^2 + \frac{n_z m}{n_z^+}\,\delta_z^2$$
$$(n_z, \mu_z, M_{2,z}) \leftarrow (n_z^+,\, \mu_z^+,\, M_{2,z}^+)$$

After streaming all batches:

$$m_{x,\ell} \leftarrow \mu_x, \quad s_{x,\ell} \leftarrow \sqrt{M_{2,x}/n_x},$$
$$m_{N,\ell} \leftarrow \mu_N, \quad s_{N,\ell} \leftarrow \sqrt{M_{2,N}/n_N}$$

4: **Fit affine surrogate (per feature):**

$$\gamma'_\ell = \frac{s_{N,\ell}}{s_{x,\ell}}, \qquad \beta'_\ell = m_{N,\ell} - \gamma'_\ell \odot m_{x,\ell}$$

5: **Replace** ℓ**:** Remove the normalization and insert

$$y = \gamma'_\ell \odot x + \beta'_\ell$$

6: **Proceed:** Continue with the modified model so downstream stats reflect the replacement

7: **return** Calibrated affine surrogates γ and β.

Appendix D. Sequential calibration: Experiments

In this section, we quantify the importance of the sequential calibration procedure described in Algorithm 1 (Appendix C) for image restoration models. We consider two representative $\times 4$ super-resolution backbones, MambaIR and SRFormer, and evaluate DaMN variants on the Set14 benchmark. All models are calibrated and fine-tuned using the same data and schedules as in the main SR experiments. We compare our full DaMN with sequential calibration to two ablations: (1) a "no scaling" variant, in which we only shift activations without matching their moments to the outputs of the original normalization layers, and (2) a "non-sequential" variant, where all surrogate affine layers are fitted in parallel on the original network and applied in a single step, without updating downstream statistics after each replacement.

Table S3 reports PSNR on Set14 before and after DaMN fine-tuning. For both architectures, the "no scaling" variant is highly unstable and immediately diverges, yielding NaN values even after fine-tuning. The non-sequential calibration is also fragile: for MambaIR it diverges as well, and for SRFormer it produces a substantially worse initialization (27.16 dB vs. 28.61 dB before fine-tuning) and slightly lower performance after fine-tuning (29.04 dB vs. 29.09 dB) compared to our sequential scheme. In contrast, the full sequential calibration yields stable surrogates for both models and preserves reasonable PSNR before fine-tuning (26.40 dB for MambaIR and 28.61 dB for SRFormer), which smooth removal then recovers to near-baseline performance.

Table S3. Ablation on sequential calibration in DaMN. We report a restoration metric PSNR↑ on dataset Set14 before and after fine-tuning.

Model	DaMN (no scaling)		DaMN (non-sequential)		DaMN (sequential)	
	Before FT	After FT	Before FT	After FT	Before FT	After FT
MambaIR (x4)	NaN	NaN	NaN	NaN	26.40	29.14
SRFormer (x4)	NaN	NaN	27.16	29.04	28.61	29.09

Figures S5 and S6 visualize the corresponding training dynamics. For MambaIR, even with smooth removal the training loss for non-sequentially calibrated surrogates remains unstable and cannot be reliably stabilized, confirming that smoothing alone cannot compensate for incorrect statistics. For SRFormer, smooth removal is more effective and eventually recovers good performance, but the loss curve still exhibits noticeable spikes and oscillations compared to the much smoother trajectory obtained with sequential calibration. Overall, these results demonstrate that recalibrating layer-by-layer while propagating the modified activations is better for stability and final accuracy; smooth removal acts as a complementary mechanism, but cannot reliably fix poorly calibrated surrogates.

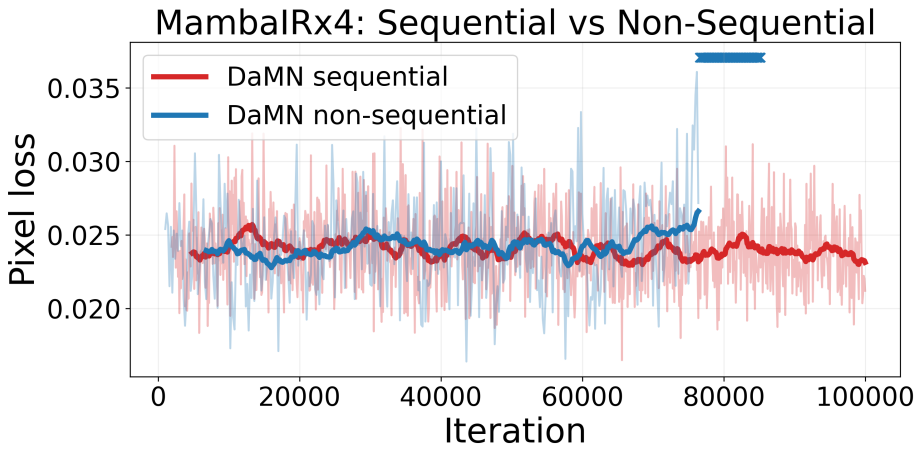

Fig. S5. Training losses for MambaIRx4 for sequential and non-sequential calibrations.

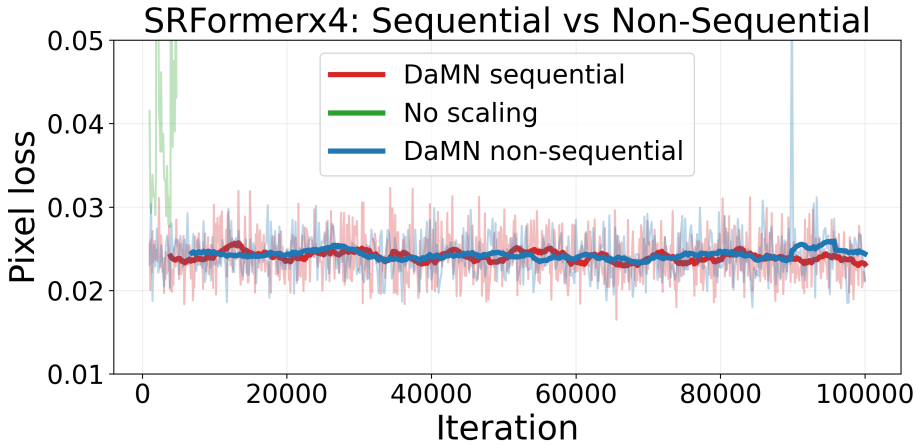

Fig. S6. Training losses for SRFormerx4 for sequential, non-sequential calibrations. And also for no scaling mode.

Appendix E. Noise injection

We 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. Given the parameters $\gamma, \beta \in \mathbb{R}^C$ computed through our sequential calibration procedure, we construct perturbed versions $\tilde{\gamma}$ and $\tilde{\beta}$ by adding zero-mean Gaussian noise scaled by the typical magnitude of each parameter:

$$\tilde{\gamma}_i = \gamma_i + \sigma \cdot \left(\frac{1}{C} \sum_{i=1}^{C} |\gamma_i| \right) \cdot \varepsilon_i^{\gamma}, \quad \tilde{\beta}_i = \beta_i + \sigma \cdot \left(\frac{1}{C} \sum_{i=1}^{C} |\beta_i| \right) \cdot \varepsilon_i^{\beta},$$

where $\varepsilon_i^{\gamma}, \varepsilon_i^{\beta} \sim \mathcal{N}(0, 1)$. This formulation ensures that σ represents an interpretable relative noise level. We then replace the original affine parameters with the perturbed versions across all layers and re-evaluate the model on the validation set. By varying σ and measuring the resulting performance degradation, we directly quantify how small errors in the estimated statistics – and the derived affine parameters – translate into accuracy loss.

Figure 4 in the main text of the paper demonstrates that even modest perturbations in γ and β can lead to significant performance degradation, emphasizing the importance of numerically stable and precise moment estimation during the calibration process. To mitigate such numerical instability during the online estimation of moments, we employ the Welford algorithm [16] for computing means and variances.

Appendix F. FP baseline reproduction for image restoration

Image Super-Resolution. We evaluate DaMN on three representative transformer-based models for image super-resolution: SwinIR [9], MambaIR [4], and SRFormer [19]. All models were trained using their original repositories [2] [3] [4] with DF2K dataset [10] following the provided training protocols for SwinIR and MambaIR. Regarding SRFormer, we note that the configurations in our reproduction differed slightly from the checkpoint architecture provided in the original repository, i.e. we used the training configuration file provided in the original repository and the architecture specified in this configuration differs from the architecture of the pre-trained model ('.pth' file) also provided by the repository. For the $\times 3$ and $\times 4$ upscaling tasks, we adopted the same training strategy as in MambaIR and SwinIR, where the SRFormer was initialized using the $\times 2$ model checkpoint and further trained by modifying only the upsampling module, while keeping the feature extraction layers fixed.

The reproduction results for these models at different scales are presented in Table S4. For **SwinIR** and **MambaIR**, the model reproduction yielded results very close to the original models. For **SRFormer**, the configuration used in the reproduction yielded slightly improved baseline performance. To ensure a fair comparison, DaMN was evaluated on both the official checkpoint provided in the original repository and our independently reproduced model, in conjunction with baseline methods including DyT and Identity. Experimental results demonstrated that DaMN consistently outperformed both DyT and Identity across both checkpoints, as detailed in the main text of the paper.

Table S4. PSNR/SSIM for the original model and our reproduction at scales $\times 2$, $\times 3$, $\times 4$ for three transformer-based models on standard SR benchmarks.

Model	Scale	Method	Set5		Set14		BSD100		Urban100		Manga109	
			PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
SwinIR	x2	Original	38.42	0.9623	34.46	0.9250	32.53	0.9041	33.81	0.9427	39.92	0.9797
		Reproduction	38.40	0.9618	34.44	0.9243	32.55	0.9030	33.81	0.9427	39.92	0.9797
	x3	Original	34.97	0.9318	30.93	0.8534	29.46	0.8145	29.75	0.8826	35.12	0.9537
		Reproduction	34.85	0.9299	30.83	0.8524	29.35	0.8133	29.66	0.8810	34.99	0.9520
	x4	Original	32.92	0.9044	29.09	0.7950	27.92	0.7489	27.45	0.8254	32.03	0.9260
		Reproduction	32.90	0.9045	29.10	0.7934	27.85	0.7488	27.38	0.8240	32.00	0.9247
MambaIR	x2	Original	38.57	0.9627	34.67	0.9261	32.58	0.9048	34.15	0.9446	40.28	0.9806
		Reproduction	38.54	0.9624	34.69	0.9263	32.55	0.9046	34.12	0.9443	40.26	0.9804
	x3	Original	35.08	0.9323	30.99	0.8536	29.51	0.8157	29.93	0.8841	35.43	0.9546
		Reproduction	35.09	0.9323	31.00	0.8539	29.51	0.8157	29.94	0.8842	35.44	0.9546
	x4	Original	33.03	0.9046	29.20	0.7961	27.98	0.7503	27.68	0.8287	32.32	0.9272
		Reproduction	33.02	0.9041	29.19	0.7958	27.97	0.7498	27.63	0.8273	32.29	0.9268
SRFormer	x2	Original	38.51	0.9627	34.45	0.9253	32.56	0.9045	34.08	0.9448	40.07	0.9802
		Reproduction	38.58	0.9630	34.79	0.9270	32.60	0.9050	34.35	0.9461	40.24	0.9806
	x3	Original	35.02	0.9323	30.94	0.8540	29.48	0.8156	30.04	0.8865	35.26	0.9543
		Reproduction	35.05	0.9325	31.00	0.8543	29.51	0.8159	30.09	0.8869	35.30	0.9547
	x4	Original	32.92	0.9043	29.08	0.7952	27.93	0.7499	27.67	0.8310	32.20	0.9271
		Reproduction	32.96	0.9045	29.12	0.7955	27.96	0.7501	27.72	0.8313	32.24	0.9274

Blind Image Denoising. To reproduce the checkpoint for blind image denoising, we used the original Restormer repository[5] without any modifications. The reproduction results are presented in Table S5. The reproduced model demonstrated performance closely aligned with that of the original.

Table S5. Reproduction results for blind denoising on the SIDD dataset.

Method	**PSNR (dB)**	**SSIM**
Original	40.02	0.9604
Reproduction	40.02	0.9603

[2] https://github.com/JingyunLiang/SwinIR
[3] https://github.com/csguoh/MambaIR
[4] https://github.com/retsuh-bqw/SRFormer-Text-Det
[5] https://github.com/swz30/Restormer

Appendix G. SR results for $\{\times 2, \times 3, \times 4\}$ scales

The super-resolution (SR) results for scale factors $\times 2$, $\times 3$, and $\times 4$ are presented in Table S6. As shown, DaMN consistently achieves the highest PSNR and SSIM values across all scales and datasets for the MambaIR model, outperforming DyT. DyT demonstrates competitive performance on SwinIR and SRFormer.

Figures S7-S11 illustrate representative results at scale $\times 4$, comparing the original MambaIR and SRFormer models with their respective Identity, DyT, and DaMN variants, all without normalization layers. For MambaIR, the Identity variant is not displayed due to training instability. The visual comparison reveals that DaMN consistently preserves sharper lines and finer details compared to Identity, which produces overly blurred outputs, and DyT, which also introduces some blurring, albeit less severe. Overall, DaMN demonstrates superior visual quality compared to the other methods evaluated.

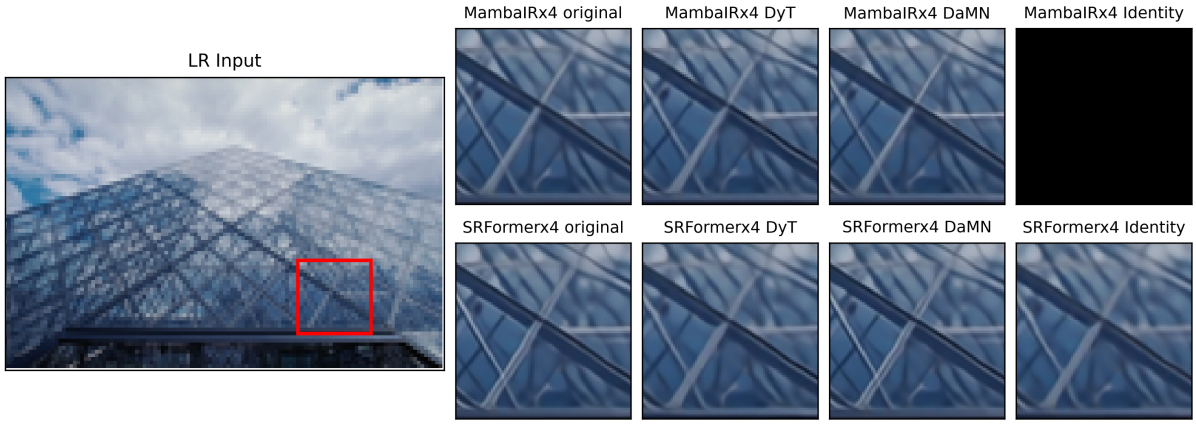

Fig. S7. BSD100 image #223061.

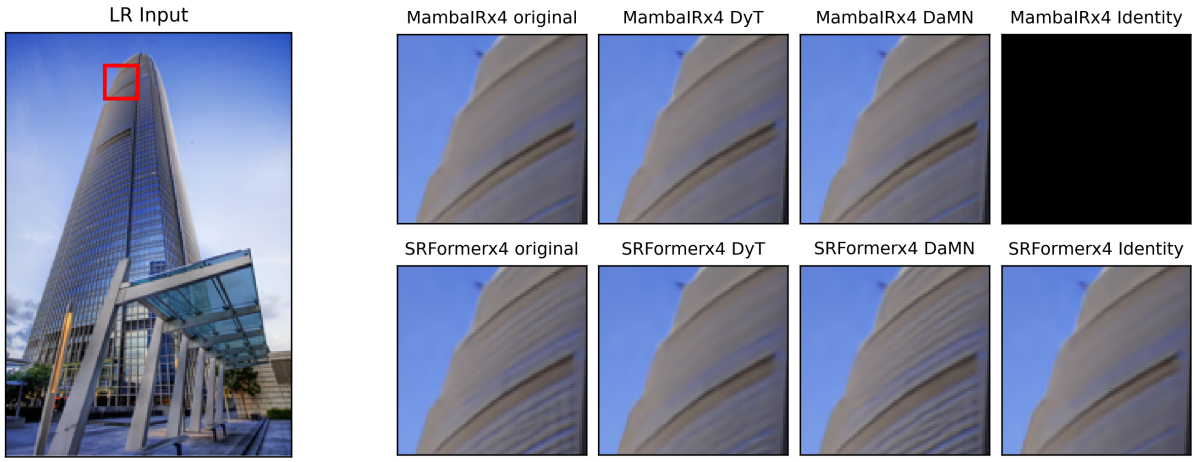

Fig. S8. Urban100 image #46.

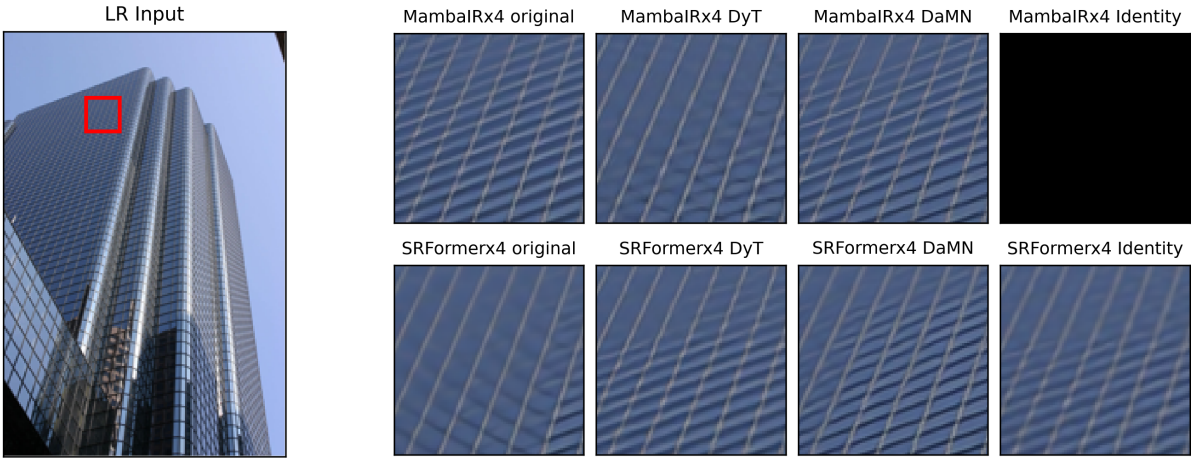

Fig. S9. Urban100 image #74.

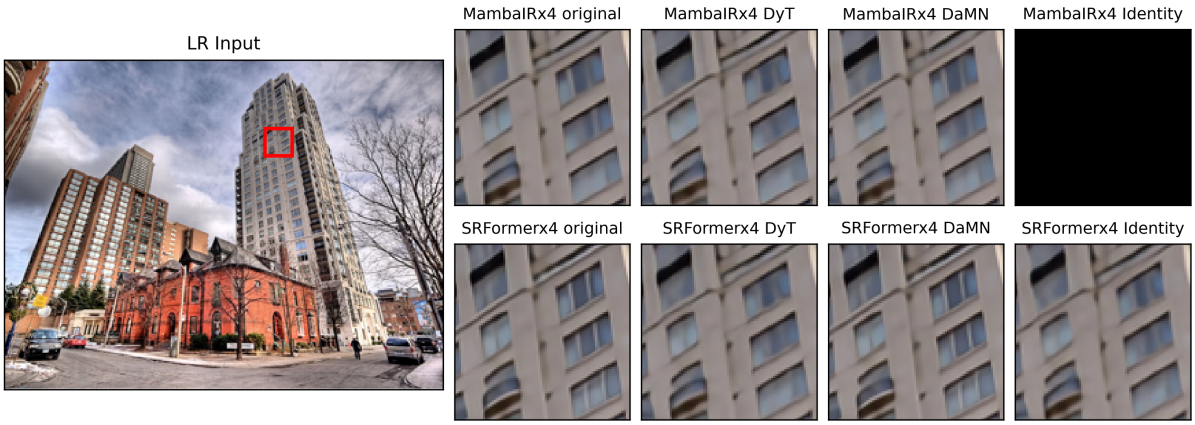

Fig. S10. Urban100 image #20.

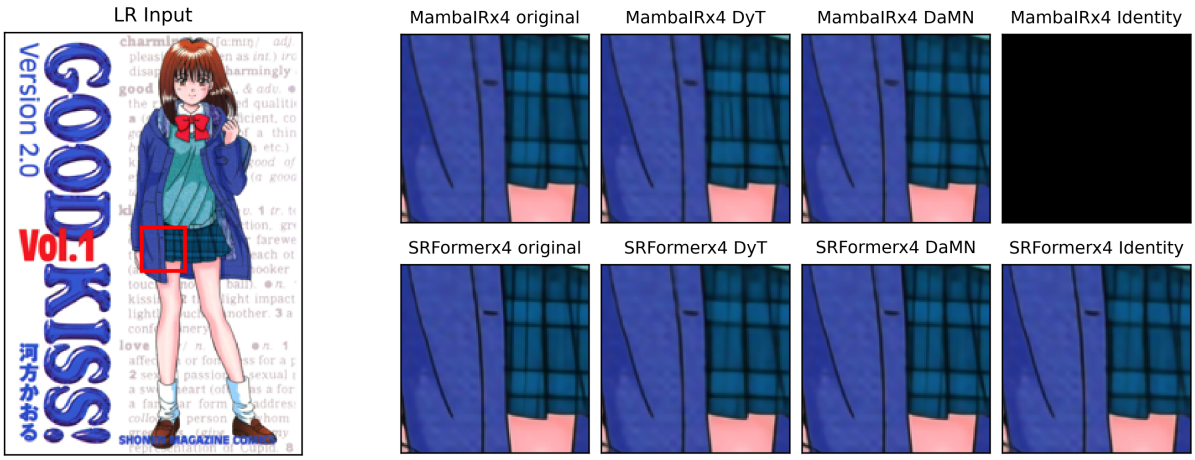

Fig. S11. Manga109 image #GOOD_KISS_Ver2.

Table S6. PSNR/SSIM for SwinIR, MambaIR and SRFormer at all scales on standard SR benchmarks. Red color indicates the best performance while blue color indicates the second best performance.

Model	Scale	Method	Set5		Set14		BSD100		Urban100		Manga109	
			PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
SwinIR	x2	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**
		DaMN (ours)	*38.30*	**0.9605**	*34.41*	*0.9240*	*32.41*	*0.9019*	*33.68*	*0.9415*	*39.70*	*0.9736*
	x3	Original	34.85	0.9299	30.83	0.8524	29.35	0.8133	29.66	0.8810	34.99	0.9520
		Identity	34.78	0.9293	30.70	0.8510	29.03	0.8021	29.51	0.8785	34.78	0.9476
		DyT	**35.05**	**0.9325**	**30.90**	**0.8537**	**29.52**	**0.8150**	**29.70**	**0.8826**	**35.05**	**0.9533**
		Naive Removal	34.34	0.9226	30.27	0.8471	28.58	0.7947	29.26	0.8713	34.64	0.9415
		DaMN (ours)	*34.80*	*0.9295*	*30.78*	*0.8518*	*29.32*	*0.8135*	*29.61*	*0.8815*	*34.95*	*0.9526*
	x4	Original	32.90	0.9045	29.10	0.7934	27.85	0.7488	27.38	0.8240	32.00	0.9247
		Identity	32.83	0.9039	28.87	0.7920	27.53	0.7376	27.23	0.8215	31.79	0.9203
		DyT	**32.90**	**0.9041**	**29.17**	**0.7947**	**28.02**	**0.7505**	**27.42**	**0.8256**	**32.06**	**0.9260**
		Naive Removal	32.39	0.8972	28.54	0.7881	27.08	0.7302	26.98	0.8143	31.65	0.9142
		DaMN (ours)	*32.85*	*0.9040*	*28.95*	*0.7925*	*27.70*	*0.7463*	*27.25*	*0.8225*	*31.85*	*0.9235*
MambaIR	x2	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*
		DaMN (ours)	**38.49**	**0.9625**	**34.52**	**0.9253**	**32.55**	**0.9046**	**33.93**	**0.9433**	**40.12**	**0.9801**
	x3	Original	35.09	0.9323	31.00	0.8539	29.51	0.8157	29.94	0.8842	35.44	0.9546
		Identity	Training diverged right from the start									
		Naive Removal	Training diverged right from the start									
		DyT	*34.90*	*0.9310*	*30.79*	*0.8500*	*29.40*	*0.8128*	*29.37*	*0.8750*	*34.94*	*0.9521*
		DaMN (ours)	**35.01**	**0.9317**	**30.86**	**0.8524**	**29.46**	**0.8145**	**29.70**	**0.8807**	**35.06**	**0.9532**
	x4	Original	33.02	0.9041	29.19	0.7958	27.97	0.7498	27.63	0.8273	32.29	0.9268
		Identity	Training diverged right from the start									
		Naive Removal	Training diverged right from the start									
		DyT	*32.84*	*0.9023*	*29.06*	*0.7921*	*27.90*	*0.7469*	*27.17*	*0.8165*	*31.92*	*0.9231*
		DaMN (ours)	**32.90**	**0.9031**	**29.14**	**0.7935**	**27.93**	**0.7477**	**27.37**	**0.8210**	**31.94**	**0.9219**
SRFormer	x2	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
		DaMN (ours)	*38.42*	**0.9622**	*34.41*	0.9249	**32.55**	**0.9040**	**34.13**	**0.9452**	**40.10**	**0.9802**
	x3	Original	35.05	0.9325	31.00	0.8543	29.51	0.8159	30.09	0.8869	35.30	0.9547
		Identity	34.87*	0.9307*	30.80*	0.8503*	29.37*	0.8121*	29.34*	0.8758*	34.82*	0.9519*
		Naive Removal	Training diverged right from the start									
		DyT	*34.93*	*0.9315*	*30.88*	*0.8520*	*29.42*	*0.8137*	*29.64*	*0.8801*	*35.06*	*0.9530*
		DaMN (ours)	**35.03**	**0.9324**	**30.96**	**0.8544**	**29.50**	**0.8159**	**30.09**	**0.8875**	**35.32**	**0.9546**
	x4	Original	32.96	0.9045	29.12	0.7955	27.96	0.7501	27.72	0.8313	32.24	0.9274
		Identity	32.24*	0.8968*	28.87*	0.7883*	27.74*	0.7424*	27.00*	0.8130*	31.64*	0.9199*
		Naive Removal	Training diverged right from the start									
		DyT	*32.85*	*0.9030*	*29.06*	*0.7926*	*27.88*	*0.7473*	*27.35*	*0.8218*	*31.93*	*0.9239*
		DaMN (ours)	**32.94**	**0.9043**	**29.09**	**0.7953**	**27.93**	**0.7501**	**27.71**	**0.8319**	**32.21**	**0.9271**

Appendix H. Image Classification Experiment Setup

Table S7. Reproduction results for image classification models on ImageNet (Top-1 Accuracy, %).

Method	Swin-B	ViT-B	ConvNeXt
Original	83.4	83.6	83.8
Reproduction	83.4	83.0	83.1

Reproduction. We used the following transformer-based models for evaluating our method on the ImageNet-1K classification task: Swin-B, ViT-B, and ConvNeXt-B. These models were reproduced as follows. For Swin-B, we used the official repository provided by the original paper to ensure consistency with the reported results. For ViT-B, we employed the latest training strategies from the timm library [6], incorporating ROPE to align closely with the performance metrics described in the original ViT paper. For ConvNeXt-B, we followed the training protocol implemented in the timm repository.

The reproduced results for all models are presented in Table S7. The reproduction results are slightly different from those reported in the original papers, primarily due to adjustments in batch sizes to accommodate training on a single node with 8 GPUs.

DaMN experimental setup. In all experiments, approximately 20% of the training dataset was used to calibrate statistics during the initialization of surrogate models. The finetuning process using DaMN required only 30% of the original training epochs, with alpha scheduling applied throughout the entire fine-tuning phase.

[6] `https://pypi.org/project/timm/`

Appendix I. Details of LLM experiments

Table S8. Language model results with normalization removal. For Modded-NanoGPT we report validation loss (lower is better). For Qwen2.5-1.5B we report Pass@1 on MATH500 and GSM8K (higher is better).

Model	Size	Data / Task	Removed NLs	Metric	Baseline	DyT	DaMN
Modded-NanoGPT	128M	FineWeb	$30/45$ (67%)	Val. loss ↓	3.29	3.27	3.22
Qwen2.5-1.5B	1.5B	OpenMathInstruct-2	28/56 (50%)†	MATH500 Pass@1 ↑	29.6	–	28.8
				GSM8K Pass@1 ↑	72.3	–	67.6
Qwen2.5-1.5B	1.5B	FineWeb-Edu	28/56 (50%)†	MMLU ↑	58.1	–	54.2
				MMLU-Pro ↑	27.3	–	23.6
				HellaSwag ↑	49.4	–	48.6
				Social IQA ↑	46.4	–	46.3
				WinoGrande ↑	66.0	–	62.9
				OpenBookQA ↑	32.2	–	32.8
				ARC-Easy ↑	75.0	–	75.2
				ARC-Challenge ↑	42.2	–	41.4

Pre-trained GPT-2. The LLM training-from-scratch experiments were conducted on the FineWeb dataset [12] using a modified Modded-NanoGPT repository circa commit 45544f533ae9528ce0f94f2416fd9bd59c2ad54c. We note that this implementation differs from the vanilla GPT-2 [13] in the presence of QK-norms [5]. QK-norms are used in many modern transformers, so their removal is put in the same regard. Firstly, we reproduced the speedrun experiment, reaching 3.285 validation loss at finish, having made 1390 steps.

The normalization statistics were collected in sequential order for each block in a row: pre-attention, QK-Norms, pre-mlp, then the statistics computation was performed on the dataset. In our experiments, we found 1000 batches of the training dataset items sufficient for calibration. Statistics were collected in FP64 for maximum precision.

From the computed statistics, we selected the first N normalization layers to remove, starting from the one nearby the transformer's embedding layer. The removed normalizations are provided in Table S9. Note, that in the used architecture the self-attention is removed from 7th decoder layer (signed as N/A).

The DyT training was performed from scratch in the same setup as the original ModdedGPT training, with the exception of all RMS norms replaced with parametrized hyperbolic tangents. The total number of steps of the DyT training equals the count of the GPT training steps plus the number of steps the GPT was finetuned on during removal with DaMN, for fair comparison. It can be seen from Table S8 that the final validation loss at the end of the GPT DaMN's fine-tuning is lower than the baseline and the DyT training. We did an additional experiment with GPT for 4000 steps without adjustments to the repository's code, ending with the loss of 3.145. This shows that the speedrun's objective of replicating the performance of OpenAI's GPT-2 ends the training before reaching the convergence plateau of this evidently more capable customized model. This confirms that the loss increase from the norms removal was eventually compensated with the progression on the tuning dataset.

Finetuned Qwen. In more practical scenarios with larger models, generally, the training data are not available, even for open-source models. Thus, for the Qwen2.5 model we conducted two experiments: 1) with *post-training* on the FineWeb-Edu dataset to show applicability for base models in a classical setup, and 2) with *fine-tuning* on a mathematical dataset to show the applicability to the downstream task.

For the finetuning experiment, we used Qwen2.5-1.5B and trained it on the OpenMathInstruct-2 dataset. We used a split of 1M instructions with a maximum sequence length of 2048 tokens and trained the initial Qwen model during a single epoch, obtaining a model, to which DaMN was then applied. For the post-training experiment, we used Qwen2.5-1.5B and further trained it on FineWeb-Edu with 1B tokens.

Qwen2.5 model's normalizations are two RMSNorm layers in each transformer block: pre-attention normalization (input_layernorm) and pre-MLP normalization (post_attention_layernorm). We find that removing pre-MLP normalizations leads

to divergence during training or a significant performance drop. Therefore, in this experiment, we delete only the pre-attention normalizations (see Table S10). We calculate statistics without Sequential Calibration, as in this particular case it leads to better results. For finetuning on OpenMathInstruct-2, we then applied Smooth Removal using 25% of the training data and continued training on the remaining 75%. For post-training on FineWeb-Edu, the training seemed to be less stable overall, and our best result was obtained using Smooth Removal during 1000 iterations comprising 262M tokens in total without further training.

For the fine-tuning experiment, we measure Pass1 accuracy on MATH-500 and GSM8K, and for the post-training experiment, we measure MMLU, MMLU-Pro, HellaSwag, SIQA, WinoGrande, OpenbookQA, ARC-Challenge, ARC-Easy benchmarks using HarnessEval. The obtained results are shown in Table S8. We observe a moderate performance drop on some benchmarks, while other metrics remain largely unchanged or show only a negligible decline (we also observe that the final loss in DaMN is the same as in the baseline setup). The drop may be due to a non-ideal setup, as LLM-based systems are inherently unstable. However, the key point is that the performance did not collapse and did not decline in some cases.

In conclusion, our experiments demonstrate that the proposed method can apply to both language models after the *pre-training* (GPT) and the *post-training* (Qwen) stages. Other removal strategies, such as removal schedules other than cosine and removing the normalization layers in an alternate order or from the middle of a transformer, and the effects of reinforcement learning setups can be explored in further research.

Table S9. Modded NanoGPT Removed Layer Normalizations Grouped by Block and Type.

Block	Q Norm	K Norm	Pre-Attn Norm	Pre-MLP Norm
0	Removed	Removed	Removed	Removed
1	Removed	Removed	Removed	Removed
2	Removed	Removed	Removed	Removed
3	Removed	Removed	Removed	Removed
4	Removed	Removed	Removed	Removed
5	Removed	Removed	Removed	Removed
6	Removed	Removed	Removed	Removed
7	N/A	N/A	N/A	Removed
8	Removed	Preserved	Preserved	Preserved
9	Preserved	Preserved	Preserved	Preserved
10	Preserved	Preserved	Preserved	Preserved
11	Preserved	Preserved	Preserved	Preserved

Table S10. Qwen2 Removed Layer Normalizations by Layer Index.

Layer Index	input_layernorm	post_attention_layernorm
0	Removed	Preserved
1	Removed	Preserved
2	Removed	Preserved
3	Removed	Preserved
4	Removed	Preserved
5	Removed	Preserved
6	Removed	Preserved
7	Removed	Preserved
8	Removed	Preserved
9	Removed	Preserved
10	Removed	Preserved
11	Removed	Preserved
12	Removed	Preserved
13	Removed	Preserved
14	Removed	Preserved
15	Removed	Preserved
16	Removed	Preserved
17	Removed	Preserved
18	Removed	Preserved
19	Removed	Preserved
20	Removed	Preserved
21	Removed	Preserved
22	Removed	Preserved
23	Removed	Preserved
24	Removed	Preserved
25	Removed	Preserved
26	Removed	Preserved
27	Removed	Preserved

Bibliography

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[2] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report, 2023.

[3] 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.

[4] 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.

[5] Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4246–4253, 2020.

[6] 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.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[8] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[9] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer, 2021.

[10] 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.

[11] 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, 2021.

[12] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[13] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI*, 2019.

[14] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[15] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. 2016.

[16] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[17] Yuxin Wu and Kaiming He. Group normalization. In *European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[18] Biao Zhang and Rico Sennrich. Root mean square layer normalization, 2019.

[19] 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.