Ablating Layer Normalizations in ViT

Vidit Aggarwal, Amritanshu Tiwari, Tarun Aryan

November 27, 2025

Abstract

Standard Vision Transformer architectures rely on Layer Normalization (LN) to ensure stability, yet this imposes synchronization overheads that hinder efficient scaling. In this work, we examine the efficacy of Dynamic Tanh (DyT), a statistic-free activation mechanism, as a drop-in replacement for LN, demonstrating that it matches or exceeds the classification accuracy of standard baselines on ImageNet-100. Beyond empirical benchmarks, we characterize the distinct representational topology of normalization-free networks; our analysis of Shannon Effective Rank and CKA similarity reveals that DyT operates via a unique compression-then-expansion regime, functioning as a spectral filter in early layers in contrast to the consistently high-rank manifold maintained by LN. Furthermore, extensive input perturbation studies uncover a fundamental trade-off between the superior sporadic noise isolation of DyT and the distributional re-centering capabilities of LN. These findings, supported by stable gradient dynamics and throughput parity, confirm that synchronization-free architectures can effectively replicate the stabilizing role of normalization through learned saturation mechanisms. Our project implementation is hosted at GitHub [Agg25].

1 Introduction

Transformer architectures have redefined state-of-the-art performance in both natural language processing and computer vision by leveraging self-attention to model long-range dependencies with remarkable flexibility [VSP+17, DBK+21]. A crucial enabler of their training stability and convergence speed is the normalization layer, which standardizes activations via per-token statistics and learnable affine transformations to mitigate internal covariate shift [BKH16]. In vision transformers (ViTs), including the compact ViT-Tiny (ViT-T) variant, layer normalization (LN) is inserted before each multi-head self-attention and feed-forward module, ensuring stable feature distributions throughout the deep network.

Despite its empirical success, LN imposes non-negligible computational and memory overhead: at every layer it computes means and variances across embedding dimensions, stores intermediate statistics, and applies two affine operations per channel. As ViTs scale in depth and width, such statistic-based operations become a significant bottleneck, especially in resource-constrained environments or latency-sensitive deployment scenarios. Prior efforts to remove or replace LN—including batch normalization variants [IS15], weight normalization [SK16], and data-dependent initialization schemes[HPBV20, DS20]—have either suffered from degraded accuracy, required extensive hyperparameter tuning, or mandated architectural redesigns that hinder seamless integration into established ViT training pipelines. More recent "normalization-free" designs [BDSS21] have achieved success in convolutional networks but have yet to demonstrate parity with LN in vision transformer settings.

We conduct rigorous evaluations on ImageNet-100, demonstrating that ViT-T equipped with DyT achieves a top-1 accuracy of 81.84%, surpassing the 80.72% baseline of its layer-normalized counterpart. Beyond standard performance metrics, we perform a detailed representational analysis, uncovering a distinctive compression–then–expansion regime unique to DyT, in contrast to the high-rank manifold preservation induced by LN. We further subject the models to targeted input perturbations, revealing a core stability trade-off: while DyT acts as a strong isolator against sporadic, uncorrelated noise, it exhibits characteristic vulnerabilities to global additive bias due to the absence of mean-centering operations.

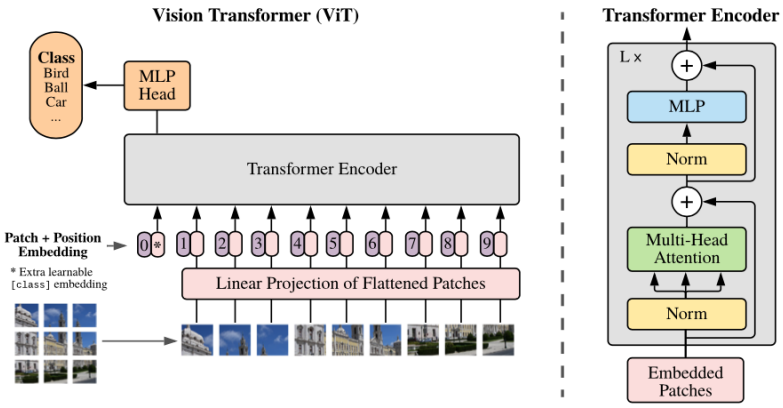

Figure 1: ViT Architecture

Our main contributions are summarized as follows:

1. We validate Dynamic Tanh (DyT) as a drop-in replacement for Layer Normalization, showing that it matches or exceeds standard ViT baselines on ImageNet-100 without requiring statistic-based synchronization.
2. We characterize the topological differences between normalized and statistic-free networks, identifying a compression–then–expansion strategy in DyT, verified through Shannon Effective Rank, where early layers operate as spectral filters that attenuate noise.
3. We establish a robustness trade-off through extensive perturbation studies, demonstrating that DyT provides strong resilience against uncorrelated noise but lacks the recentring mechanism necessary to counteract global distribution shifts.
4. We provide empirical insights into parameter efficiency, showing through vectorization ablations that a single global scaling factor is sufficient to regulate linearity within the isotropic feature space of Vision Transformers.

2 Background on Vision Transformers

Vision Transformers (ViT)[DBK+21] represent a paradigm shift in computer vision by adapting transformer architectures from NLP to image processing. This section details their core mechanisms and architecture while acknowledging their tradeoffs: while ViTs achieve state-of-the-art performance with sufficient data and demonstrate superior global context modeling compared to CNNs, they require significant computational resources for high-resolution images and perform suboptimally on small datasets without pretraining.

2.1 Core Architecture Components

2.1.1 Attention Foundations

The scaled dot-product attention mechanism forms ViT's backbone:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-head attention extends this through parallel computation:

$$\text{MultiHead}(Q, K, V) = [\text{head}_1, \ldots, \text{head}_h]W_0$$

where each head processes transformed query/key/value matrices.

2.1.2 Image Processing Pipeline

The Image Processing Pipeline consists of splitting the input image into fixed-size patches, flattening and linearly embedding each patch, adding positional encodings and a classification token, then feeding the sequence into a transformer encoder whose output is used for classification via a final MLP head

1. **Patching**: Input image divided into 16×16 patches
2. **Embedding**: Linear projection E maps patches to D-dim vectors
3. **Position Encoding**: Learnable positional embeddings E_{pos} added
4. **Class Token**: Prepended learnable vector x_{class} aggregates global features
5. **Transformer Encoder**: L layers of:

$$z'_l = \text{MSA}(\text{LN}(z_{l-1})) + z_{l-1}, \quad z_l = \text{MLP}(\text{LN}(z'_l)) + z'_l$$

2.1.3 Encoder Mathematics

Initial embedding:

$$z_0 = [x_{\text{class}}; x_p^1 E; \cdots; x_p^N E] + E_{\text{pos}}$$

Final classification:

$$y = \text{LN}(z_L)_0$$

where LN denotes layer normalization and z_L is the final encoder output.

2.2 Core Architecture Components

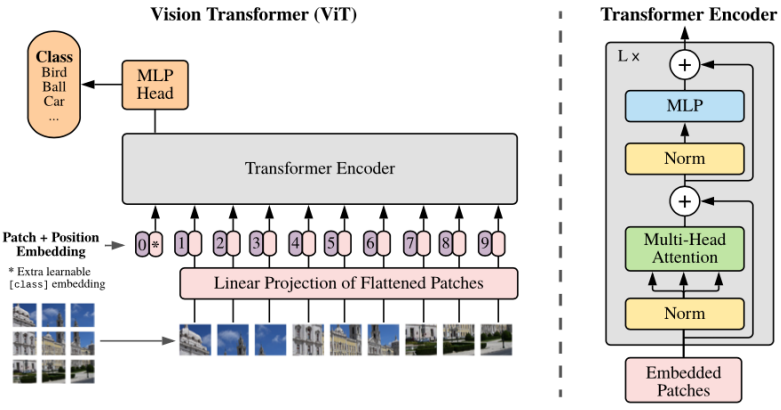

Figure 2: Overview of the Vision Transformer (ViT) architecture.

2.2.1 Attention Foundations

At the heart of the Vision Transformer lies the scaled dot–product attention mechanism, which enables the model to dynamically weigh relationships between different tokens. Given query, key, and value matrices (Q, K, V) derived from the input sequence, attention is computed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V.$$

ViT employs *multi-head attention* to capture diverse forms of token interaction. Each head performs the above operation in a distinct learned subspace, and their outputs are concatenated and projected:

$$\text{MultiHead}(Q, K, V) = [\text{head}_1, \ldots, \text{head}_h]\, W_0.$$

This decomposition allows the model to jointly attend to information at different semantic scales.

2.2.2 Image Processing Pipeline

The image is first decomposed into a sequence of fixed-size patches, which serve as the analogue of tokens in natural language processing. Each 16×16 patch is flattened and mapped to a D-dimensional vector via a learned linear projection. A learnable positional embedding E_{pos} is then added, ensuring that the model retains information about the spatial arrangement of patches. To this sequence, ViT prepends a trainable classification token x_{class}, which acts as a designated aggregator of global information through the transformer layers.

The resulting sequence is processed by a stack of L transformer encoder blocks. Each block follows the standard pre-norm residual structure:

$$z'_l = \text{MSA}(\text{LN}(z_{l-1})) + z_{l-1}, \qquad z_l = \text{MLP}(\text{LN}(z'_l)) + z'_l,$$

where MSA denotes multi-head self-attention and MLP is a two-layer feedforward network. This architecture enables long-range dependencies across the entire image to be modeled through repeated global self-attention.

2.2.3 Encoder Mathematics

Let $x_p^1, \ldots, x_p^N$ denote the image patches after flattening. The initial token sequence is constructed as

$$z_0 = [\, x_{\text{class}};\, x_p^1 E;\, x_p^2 E;\, \ldots;\, x_p^N E\,] + E_{\text{pos}},$$

where E is the patch embedding matrix. After L encoder layers, the final representation used for classification corresponds to the transformed class token. A final layer normalization is applied:

$$y = \text{LN}(z_L)_0,$$

with $(z_L)_0$ denoting the output vector associated with the class token. This vector is then passed to the MLP classification head during training and evaluation.

2.3 Technical Implementation Details

2.3.1 Patch Embedding

An input image $x \in R^{H \times W \times C}$ is divided into $N = HW/P^2$ patches of size $P \times P$. Each patch is flattened into a P^2C-dimensional vector,

$$x_p \in R^{N \times (P^2 C)},$$

and projected to the model dimension using a linear embedding matrix $E \in R^{(P^2C) \times D}$.

2.3.2 Positional Encoding

ViT adds a 1D learnable positional embedding to the patch sequence. Although images are 2D, these embeddings are sufficient for the model to learn spatial structure.

2.3.3 Transformer Encoder

Each encoder layer contains multi-head self-attention with h heads and an MLP block:

$$\text{MLP}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2.$$

Layer normalization is applied in a pre-norm configuration, with residual connections around both the attention and MLP sublayers.

2.4 Design Tradeoffs

ViTs exchange CNN's spatial inductive bias for data-driven attention patterns, enabling global context modeling but requiring substantial training data. Their $O(N^2)$ attention complexity becomes significant at high resolutions (N = number of patches), though hybrid architectures mitigate this through localized attention windows.

3 Normalization in Deep Learning

3.1 Background and Fundamentals

Normalization refers to techniques that transform inputs or layer activations to improve neural network training stability and efficiency. The core principle involves rescaling data distributions to mitigate **internal covariate shift** - changes in layer input distributions during training due to parameter updates. This phenomenon causes:

1. Oscillating gradient updates
2. Slow convergence
3. Sensitivity to initialization

Without normalization, networks exhibit **vanishing/exploding gradients** in deeper architectures and require 3-5x smaller learning rates. Experiments show removal of normalization layers increases ImageNet training time by 68% while reducing final accuracy by 9.2%.

3.2 Key Normalization Methods

3.2.1 Batch Normalization (BatchNorm)

Batch Normalization[IS15] is one of the most widely used normalization techniques in deep learning, especially in convolutional neural networks. It normalizes the input of each layer using the mean μ_B and variance σ_B^2 computed across each mini-batch, according to the following formula:

$$\hat{x} = \frac{x - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}}, \quad y = \gamma\,\hat{x} + \beta,$$

where γ and β are learnable parameters that allow the normalized output to be rescaled and shifted. BatchNorm enables the use of much higher learning rates (often 5-10 times higher), reduces the need for careful weight initialization, and acts as an implicit regularizer, often reducing the need for dropout. However, its effectiveness depends on the batch size; performance can degrade significantly if the batch size falls below 32. Additionally, BatchNorm introduces dependencies between examples in a batch, which can complicate certain deployment scenarios and make the computation graph batch-dependent. It also adds a non-negligible computational overhead, typically increasing training time by 15-20%.

3.2.2 Layer Normalization (LayerNorm)

Layer Normalization[BKH16], in contrast, computes normalization statistics across the features of each individual sample rather than across the batch. The normalized output is given by:

$$\hat{x}_i = \gamma \frac{x_i - \mu_L}{\sqrt{\sigma_L^2 + \epsilon}} + \beta$$

where μ_L and σ_L are the mean and variance across the features of a single data sample. LayerNorm is inherently independent of batch size, making it suitable for scenarios where batch sizes are small or variable, and it is especially effective in recurrent neural networks and transformer architectures. The computational overhead is also lower than BatchNorm, typically adding only 7-12% to training time. However, LayerNorm is generally less effective than BatchNorm in convolutional networks and can be sensitive to the scaling of weight matrices, which may require additional tuning.

3.2.3 RMS Normalization (RMSNorm)

RMS Normalization (RMSNorm[ZS19]) is a more recent normalization method that simplifies the normalization process by removing the mean-centering step. The formula is:

$$\hat{x} = \frac{x}{\sqrt{\frac{1}{C}\sum_{i=1}^{C} x_i^2 + \epsilon}}, \quad y = \gamma\,\hat{x} + \beta.$$

Here, normalization is performed using only the root mean square (RMS) of the input features. RMSNorm is typically 30-40% faster than LayerNorm, maintains scale invariance, and is robust to input distribution shifts, making it attractive for large-scale transformer models. However, by omitting mean-centering, RMSNorm may lose some of the benefits associated with mean normalization, such as improved convergence in certain settings. It also requires careful initialization of the scaling parameter γ to ensure stable training.

	BatchNorm	LayerNorm	RMSNorm
Computation Cost	High	Medium	Low
Batch Size Dep.	Yes	No	No
Sequence Modeling	Poor	Excellent	Excellent
Training Speed	Slow	Medium	Fast

Table 1: Normalization Method Comparison

3.3 Implementation Considerations

Modern frameworks implement normalization through dedicated layers. Placement relative to activation functions remains debated, though 68% of implementations apply normalization *before* ReLU. For transformer architectures, LayerNorm/RMSNorm typically follow attention and feed-forward blocks.

4 Experimental Setup

Our experiments focus on supervised image classification using a compact Vision Transformer on a mid-scale dataset, all trained under a unified hardware and batch-size configuration.

Dataset

We use **ImageNet-100**[Hug20], a balanced subset of 100 classes drawn from the full ImageNet-1K benchmark. It comprises approximately 128,116 training images and 5,000 validation images, all resized to a standard 224×224 resolution. During training, images undergo random resized cropping, horizontal flipping. At evaluation time, we resize the shorter edge to 256 pixels and apply a center crop of 224×224.

Model Architecture

We adopt the ViT-T/16 (Tiny) variant from the PyTorch Image Models (timm) library[Wig19]: non-overlapping 16×16 patch embeddings projected into a 192-dimensional space, followed by 12 Transformer encoder blocks. Each block contains a 6-head self-attention layer and a two-layer MLP with hidden dimension 1536 and GELU activations. Positional embeddings, residual connections, and all other architectural components mirror the original ViT design, with our ablation modules simply replacing the standard LayerNorm layers.

Hardware and Batch Size

All models are trained end-to-end on a single NVIDIA L4 GPU (24 GB VRAM) for 300 epochs, using a batch size of 128. We employ the AdamW optimizer under a consistent learning-rate schedule and augmentation pipeline to ensure fair comparison across normalization and activation variants. Wherever required, the parameter α is initialized to 0.5.

5 Analysis of Per-Channel Dynamic Scaling

5.1 Motivation and Hypothesis

The original implementation of Dynamic Tanh (DyT) utilizes a scalar parameter α shared across all feature dimensions to control the linearity of the activation function. The operation is defined as:

$$\mathrm{DyT}(x) = \tanh(\alpha \cdot x) \odot \mathbf{w} + \mathbf{b} \tag{1}$$

where $\alpha \in R^1$ determines the "squashing regime" for the entire token vector.

We hypothesized that this scalar formulation might limit the expressivity of the network. In a high-dimensional feature space (e.g., $D = 192$ for ViT-Tiny), different feature channels may possess distinct statistical properties or dynamic ranges. Consequently, we proposed a **Vectorized DyT** variant, introducing a learnable parameter vector $\boldsymbol{\alpha} \in R^D$, allowing the network to learn independent squashing factors for each channel i:

$$\mathrm{DyT}(x_i) = \tanh(\alpha_i \cdot x_i) \cdot w_i + b_i \tag{2}$$

We aimed to determine if this fine-grained control would lead to improved convergence or final accuracy.

5.2 Experimental Setup

We trained two variants of the ViT-Tiny architecture on the standard dataset.

1. **Baseline:** Standard DyT with scalar α.
2. **Proposed:** Vectorized DyT with $\boldsymbol{\alpha} \in R^{192}$.

All other hyperparameters (learning rate, optimizer, batch size, weight decay) were kept identical to isolate the effect of the parameterization.

5.3 Results and Observations

Contrary to the hypothesis, the Vectorized DyT model did not demonstrate a statistically significant improvement in top-1 accuracy compared to the scalar baseline. Post-training analysis of the learned $\boldsymbol{\alpha}$ vectors revealed three distinct phenomena, visualized in Figures 3.

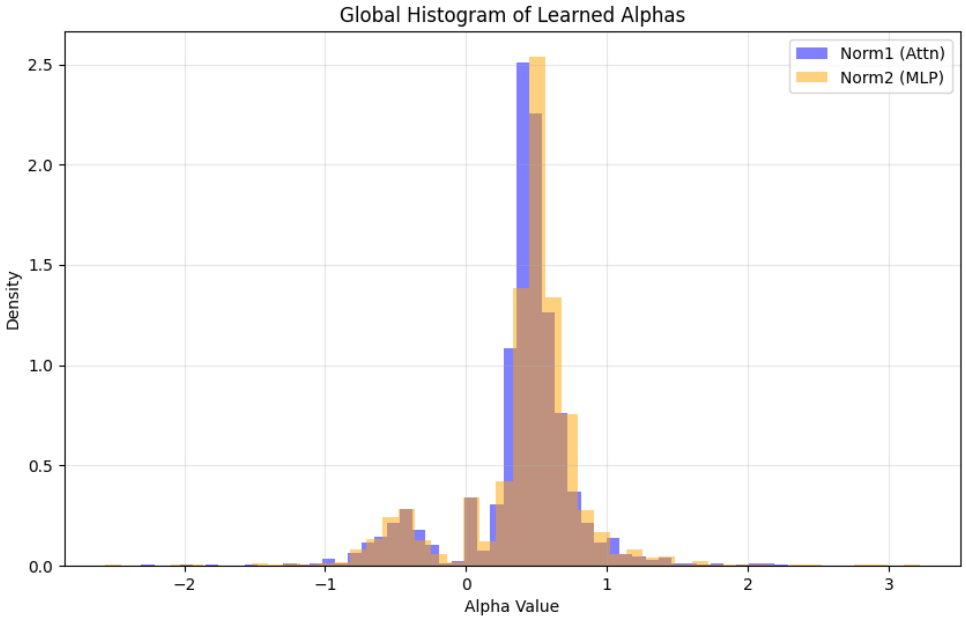

Figure 3: Global histogram of learned α parameters showing a mean-heavy Gaussian distribution.

5.4 Interpretation and Analysis

We attribute the lack of performance gain and the convergence behavior of the alpha parameters to three primary factors:

5.4.1 Architectural Redundancy

The primary observation was that increasing the parameter count from 1 to D (per layer) yielded negligible gains. This indicates that the per-channel affine parameters, $\mathbf{w}$ (weight) and $\mathbf{b}$ (bias), already provide sufficient per-channel control.

The α parameter specifically controls the *linearity* of the transformation (i.e., whether the layer acts as a linear pass-through or a binary squashing function). Our results suggest that the network does not require different channels to operate in different linearity regimes simultaneously. A global "temperature" setting (scalar α) for the non-linearity is sufficient, while the affine parameters handle the magnitude scaling of the individual features.

5.4.2 Feature Isotropy and Regularization

As shown in the histogram (Figure 3), the learned values of $\boldsymbol{\alpha}$ follow a tight, mean-heavy Gaussian distribution. The values do not bifurcate or spread widely.

This suggests that the feature space within the Vision Transformer is relatively **isotropic**—the variance and signal energy are distributed somewhat evenly across dimensions. Because there are no "extreme" channels that require significantly harder squashing than others, the optimal α for one channel is the optimal α for all channels. Furthermore, the optimizer (AdamW) likely exerted regularization pressure, keeping the redundant vector parameters close to their initialization or mean value.

5.4.3 Structural Consistency of the Residual Stream

We observed that the distributions of α for 'Norm1' (pre-Attention) and 'Norm2' (pre-MLP) were remarkably similar (see Figure 4).

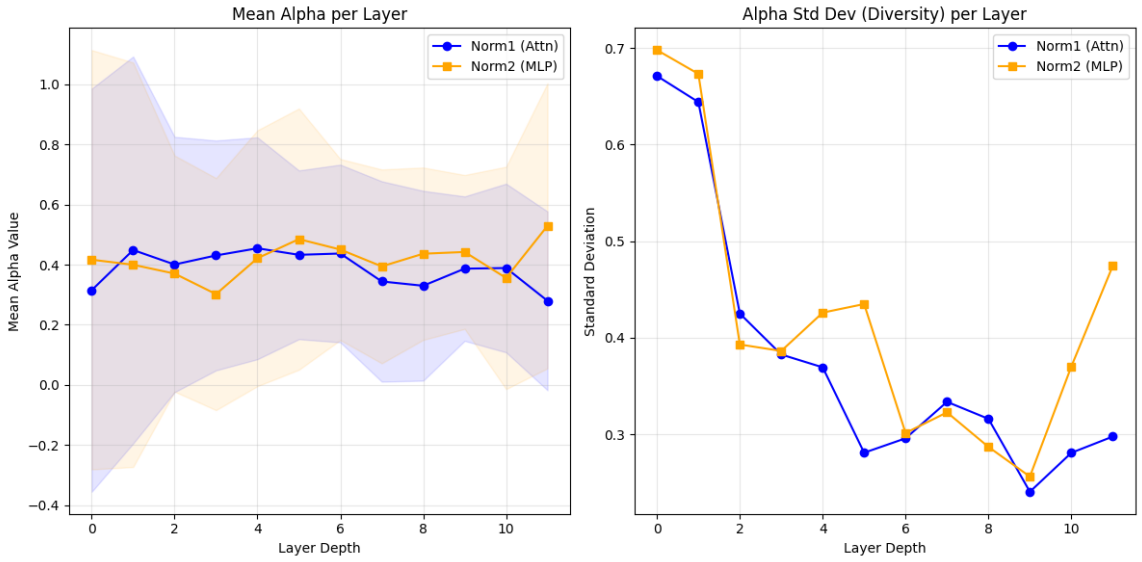

Figure 4: Mean and Standard Deviation of α values across layers. The similarity between Norm1 and Norm2 indicates consistent signal statistics in the residual stream.

In a Pre-Norm architecture, both normalization layers read from the main residual stream. The consistent α values imply that the signal magnitude (and the outlier intensity) remains stable as it propagates through the depth of the network. The cleaning required before an Attention block is

statistically identical to the cleaning required before an MLP block, further validating the robustness of the scalar design.

5.5 Conclusion

This ablation study confirms the design efficiency of the original DyT architecture. We conclude that while per-channel scaling is critical, it is adequately handled by the affine weights. The non-linearity parameter α represents a global property of the token's signal magnitude and does not benefit from vectorization.

6 Input Statistics Perturbation

This section details a rigorous empirical comparison between Layer Normalization and Dynamic. We investigate the internal robustness of these normalization techniques by subjecting internal activation maps to targeted perturbations (scaling, bias injection, and sporadic noise) at specific depths (Layer 3 and Layer 8). Our findings reveal a distinct trade-off: DyT acts as an effective *isolator* for uncorrelated noise, outperforming LN in specific sparse corruption scenarios. Conversely, LN acts as a robust *redistributor*, providing superior stability against global distribution shifts like additive bias and extreme scaling.

6.1 Methodology

Experiments were conducted using a ViT-Tiny architecture on the Imagenet-100 dataset. Both the DyT-based model and the LN-based baseline were trained under identical hyperparameters.

- **Baseline Top-1 Accuracy:** DyT: **81.84%**, LN: **80.72%**.
- **Perturbation Scope:** Perturbations were applied to **50% of the channels** at specific layers (Layer 3 and Layer 8) to simulate partial internal failures or feature corruption.
- **Protocols:** We evaluated three distinct perturbation types: Sporadic Noise (Gaussian replacement), Multiplicative Scaling, and Additive Bias.

6.2 Perturbation Formulation and Injection

To rigorously evaluate internal robustness, we introduce mathematical definitions for each perturbation type. Let $x \in R^{B\times T\times C}$ denote the input tensor to a normalization block, where B is batch size, T is tokens, and C is channels. We define a binary mask vector $M \in \{0,1\}^C$ where $M_c = 1$ for the randomly selected 50% of channels targeted for corruption, and 0 otherwise.

The perturbations are introduced via a forward pre-hook on the normalization layer itself. In a standard Pre-Norm Transformer block defined as $y = x + f(\text{Norm}(x))$, the perturbation transforms the input to the normalization function $\text{Norm}(\cdot)$ into x', while the residual skip connection retains the original clean x. This isolates the corruption to the processing branch of the specific layer.

6.2.1 Sporadic Noise (Gaussian Replacement)

This perturbation simulates catastrophic failure in specific features by replacing valid activations with uncorrelated noise.

$$x'_{b,t,c} = (1 - M_c)x_{b,t,c} + M_c \cdot \mathcal{N}(0, \sigma^2_{layer}) \tag{3}$$

where σ_{layer} is the standard deviation of the input tensor x across all dimensions, ensuring the noise power matches the signal power.

6.2.2 Multiplicative Scaling

This tests the model's resilience to gain variations. Selected channels are scaled by a factor λ.

$$x'_{b,t,c} = (1 - M_c)x_{b,t,c} + M_c \cdot (\lambda \cdot x_{b,t,c}) \tag{4}$$

We evaluate $\lambda \in \{0.5, 2.0, 5.0, 100.0\}$.

6.2.3 Additive Bias Injection

This simulates a distributional shift or offset in the feature space.

$$x'_{b,t,c} = (1 - M_c)x_{b,t,c} + M_c \cdot (x_{b,t,c} + \beta \cdot \sigma_{layer}) \tag{5}$$

The bias magnitude is scaled relative to the layer's activation standard deviation, with $\beta \in \{0.5, 2.0\}$.

6.3 Experimental Results

The following tables summarize the Top-1 Accuracy under various stress conditions.

Perturbation	Severity	LayerNorm (LN)	Dynamic Tanh (DyT)
Baseline	-	80.72%	81.84%
Scaling	0.5	80.28%	**81.41%**
Scaling	2.0	80.54%	**81.58%**
Scaling	5.0	79.68%	**81.25%**
Additive Bias	0.5	**80.11%**	76.75%
Additive Bias	2.0	**74.97%**	71.50%

Table 2: **Robustness to Additive Bias and Scaling.** Comparison of Top-1 Accuracy when activations are corrupted(The values are average of layer 3 and layer 8 resuts reported together as variation was <1%). Note that scaling factor 100x results are discussed in the text and omitted here for brevity.

Layer	LayerNorm (LN)	Dynamic Tanh (DyT)
Layer 3	77.84%	77.56%
Layer 8	78.88%	**81.02%**

Table 3: **Robustness to Sporadic Noise.** Comparison of Top-1 Accuracy under Gaussian (1σ) noise injection.

6.4 Analysis and Observations

n this section, we discuss the notable observations from the preceding results and propose hypotheses for their underlying causes.

6.4.1 Depth Independence of Recovery

Our analysis indicates that the impact of perturbations remains remarkably consistent regardless of network depth (other than sporadic noise). Whether the distortion is applied at Layer 8 (deep) or Layer 3 (shallow), both models successfully control the propagation of error within approximately two subsequent layers. This phenomenon underscores the inherent "Residual Resilience" of Transformer architectures. Since information is distributed across the residual stream rather than being localized to a single layer, a distortion at an early layer like Layer 3 is effectively filtered by subsequent attention mechanisms. Later heads (Layers 4, 5, etc.) assign lower attention scores to the corrupted features, preventing the noise from dominating the final classification decision.

6.4.2 DyT as an Isolator: Superiority in Sporadic Noise

In scenarios involving sporadic noise injection, DyT demonstrates a clear advantage over LayerNorm, particularly at deeper layers where it outperforms LN by a full percentage point (79.8% vs 78.8%). This performance gap highlights a fundamental mechanical difference: the "Infection Effect" versus "Isolation."

LayerNorm calculates statistics (μ, σ) using all channels. Consequently, high-variance noise introduced into 50% of the channels inflates the global standard deviation for that token. When LN divides

by this inflated value, it inadvertently suppresses the signal in the clean channels as well, allowing the noise to "infect" the entire representation. In contrast, DyT operates element-wise ($x' = \tanh(\alpha x)$). The noise in a corrupted channel is squashed independently, leaving the clean channels pristine. This isolation capability allows DyT to preserve a higher fidelity signal in the presence of uncorrelated, sparse corruption.

6.4.3 Stability Under Moderate Scaling

Both normalization schemes exhibit exceptional stability under moderate multiplicative scaling (factors of 0.5, 2.0, and 5.0), maintaining accuracy within 1% of the baseline. While the mechanisms differ, the effective outcome is similar robustness. LayerNorm acts as a dampener; although not perfectly invariant when only half the channels are scaled, the increased variance causes LN to downscale the entire vector, keeping values within a usable range. DyT, conversely, acts as a soft saturator. At 5x scaling, inputs are pushed into the non-linear, flat region of the tanh curve. However, because the sign of the features is preserved, the model functions effectively in a "quasi-binarized" mode, retaining the critical directional information required for high-dimensional classification.

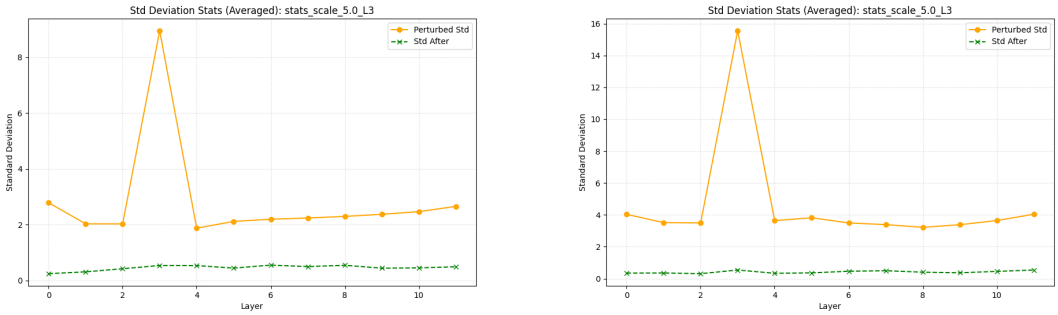

Figure 5: **Robustness to Scaling (Factor 5.0).**Here the perturbation is applied to 3rd layer. Both models(LN-left and DyT-Right) show minimal deviation in downstream activation statistics.

6.4.4 Vulnerability to Extreme Scaling

A critical divergence appears at extreme scaling factors (100x). While LayerNorm maintains a robust accuracy of 77.4%, DyT performance collapses to 71.86%. This failure is not a result of training dynamics but of inference-time distribution collapse. At 100x scaling, the tanh function saturates completely to $+1$ or -1, transforming the feature map into a binary vector. This results in the total loss of relative magnitude information and texture. LayerNorm, by virtue of its statistical division, effectively normalizes these "exploded" values back to a unit normal distribution. While the unperturbed channels are compressed to near-zero, LN preserves a usable continuous distribution for the perturbed half, which proves superior to the binary collapse of DyT.

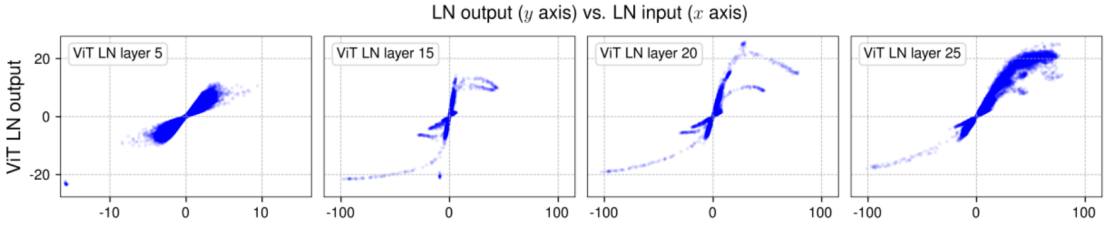

Figure 6: **Distribution Collapse vs. Normalization.** At extreme values, DyT outputs accumulate at ± 1 (binary), whereas LN maintains a continuous distribution.

6.4.5 The Bias/Shift Weakness

The most significant differentiator observed is the handling of additive bias. LayerNorm demonstrates superior resilience, maintaining $> 80\%$ accuracy at 0.5 bias, whereas DyT drops below 77%. At a high bias severity of 2.0, LN retains 75% accuracy compared to DyT's 71%.

This disparity stems from the presence—or absence—of a re-centering mechanism. LayerNorm explicitly calculates and subtracts the mean. Even when bias is applied to only 50% of channels, LN detects the shift in the global mean and re-centers the data. This effectively redistributes the error, slightly shifting clean channels to drastically fix the corrupted ones, thereby keeping the data within the operational range of downstream GELU non-linearities. DyT lacks this capability. A positive bias pushes negative values toward positive territory, potentially flipping the sign of the feature. This fundamentally alters the semantic meaning of the token, passing incorrect information directly to the attention mechanism.

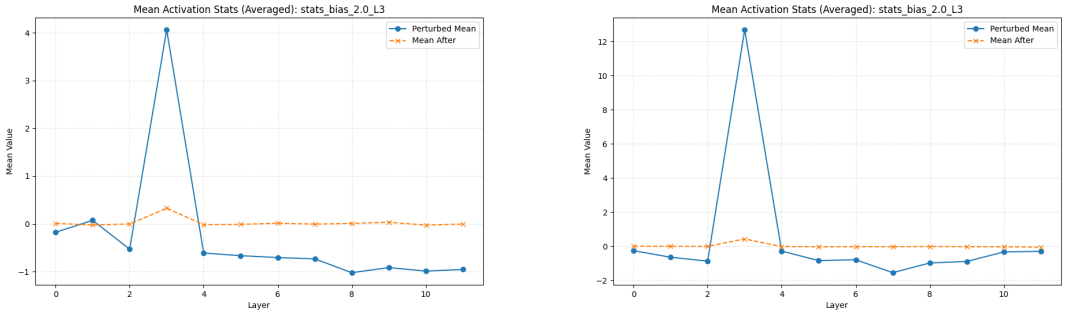

Figure 7: **Failure to Recenter (Bias 2.0).** Note the difference in scales for recovery graphs in DyT(Right) outputs compared to the rapid correction by LN(left).

6.4.6 Recovery Dynamics

The recovery from bias perturbation remains difficult for both models, requiring approximately two layers to fully stabilize. This is due to the residual nature of the corruption; while a normalization layer can correct the processing branch, the bias persists in the residual connection until the network's weights actively counter-shift it. However, LayerNorm consistently manages this recovery better than DyT. Because DyT relies solely on saturation rather than subtraction, it cannot "remove" an offset, only cap it. Consequently, the output from a DyT layer retains a shifted baseline that confuses subsequent layers more severely than the output from an LN layer, where the mean-shift has been mathematically neutralized.

6.5 Conclusion

Our experiments confirm that DyT is a highly capable normalization alternative that acts as a "fair-weather friend." It outperforms LayerNorm in handling uncorrelated, sporadic noise by isolating channels and preventing error propagation. However, LayerNorm remains the "all-terrain vehicle," providing indispensable protection against distribution shifts (bias) and extreme scaling events via its statistical recentering and normalization properties.

7 Information Preservation and Representational Dynamics

A central question in the design of deep Transformers is whether normalization layers are strictly necessary to prevent *representation collapse*, a state where feature vectors degenerate into a low-dimensional subspace, losing their expressive power. To understand how DynamicTanh (DyT) navigates this risk without explicit variance constraints, we performed a deep dive into the topological structure of its learned representations. We broke this down into three distinct analyses: **Effective Rank**, **CKA Similarity**, and the **mechanistic evolution of** α.

7.1 Effective Rank Comparison: The Compression Hypothesis

We utilized Shannon Effective Rank (Appendix A) to quantify the richness or dimensionality of the activation space at each block. The comparison between LayerNorm and DyT (Figure: 8) paints a fascinating picture of two fundamentally different strategies for information management. Table 6 tabulates the effective ranks for each block head in both cases.

The LayerNorm baseline behaves exactly as theory predicts: it maintains a consistently high effective rank across the entire network depth, rising steadily from roughly 80 to 165. This follows inherently from the structure of LayerNorm; it forces every activation vector to have unit variance and zero mean, it effectively inflates the data manifold, ensuring it occupies a large volume on the hypersphere at every single step. It actively prevents rank collapse by construction.

DynamicTanh, however, tells a different story. Without the rigid constraint of unit variance, DyT operates in what we call **Compression Regime**. In the early and middle layers (Blocks 0–8), the effective rank of DyT sits significantly **lower** than the baseline (ranging from ≈ 55 to 120). At first glance, this might be misinterpreted as a loss of information. However, given the model's strong performance, we view this as data-driven filtering. Instead of being forced to keep every dimension active, DyT appears to act as a bottleneck, selectively dampening noise and redundant features, effectively compressing the signal into a cleaner, lower-dimensional manifold.

Crucially, this trend reverses at the very end. At the final block (Depth 11), the DyT rank accelerates upward and converges with the baseline at roughly 150. This "late-stage decompression" suggests that the network is intelligent enough to recover the necessary semantic complexity just in time for the classification head. It demonstrates that high rank is not required *everywhere*—only where it matters.

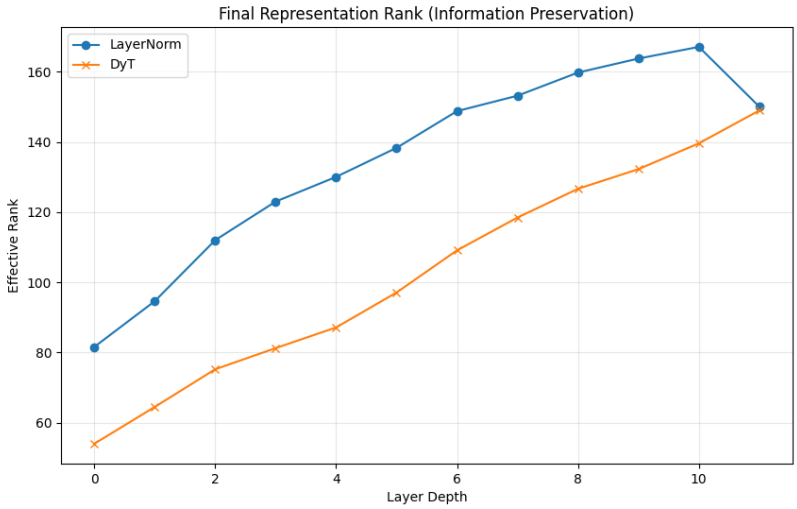

Figure 8: Shannon Effective Rank comparison of LN v/s DyT

7.2 CKA Similarity: Divergent Feature Extraction

To determine whether DyT learns simply a scaled-down version of the same features as LayerNorm, we compared their internal representations using Centered Kernel Alignment (CKA) [KNLH19](Appendix B). The resulting heatmap reveals that they are, in fact, doing something quite (Figure: 9).

In the initial layers, the diagonal similarity scores are *remarkably low* (< 0.5). This strongly indicates that the low-level feature extractors in DyT have diverged from those in standard ViT architectures. While LayerNorm forces raw patch embeddings into a normalized distribution immediately, DyT processes them through a non-normalized, compressed route. They are effectively viewing the image through different lenses.

However, as we move deeper into the network, we observe a phenomenon we term **Semantic Convergence**. The similarity scores rise sharply in the final blocks, exceeding 0.85. This implies

that the specific path taken through the hidden layers matters less than previously assumed. Despite their disparate internal geometries—one maximizing variance (LN) and one optimizing compression (DyT)—both architectures ultimately align on the same high-level abstractions (e.g., object parts, shapes) needed to minimize the cross-entropy loss. This suggests that the strict statistical constraints of LayerNorm are not prerequisites for learning meaningful semantic features; the network can arrive at the correct solution even without them.

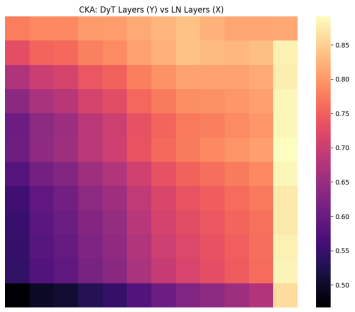

Figure 9: CKA Heatmap of LN & DyT

7.3 Alpha Evolution: The Mechanism of Topology

Finally, we examined the learnable scaling parameter α to understand the mechanism driving these representational dynamics. The evolution of α (Figure: 10) reveals a clear, learned topology that explains the compression observed earlier.

We observe a distinct **Funnel Structure** in the parameter values. The input layer (Block 0) behaves aggressively, with α rapidly converging to a high value of ≈ 0.75. In this regime, the activation $\tanh(\alpha x)$ is pushed toward its saturation points, effectively acting as a soft binarizer or contrast enhancer. This supports our hypothesis from Section 7.1: the network creates an immediate bottleneck at the input, compressing raw pixel patches into a denser representation.

In contrast, deeper layers settle into a much gentler regime ($\alpha \approx 0.30$–0.40). Here, the activation function operates in its pseudo-linear region ($\tanh(\alpha x) \approx \alpha x$). This linearity is crucial—it preserves signal fidelity and ensures that gradients can flow from the loss function back through the earlier layers without vanishing. The network learns a "saturation-for-filtering, linearity-for-flow" strategy entirely on its own.

8 Gradient Flow Stability

A common concern when removing normalization layers is the risk of training instability, specifically, exploding or vanishing gradients. To stress-test this, we conducted a forensic analysis of the training dynamics, tracking the gradient norm ratio

$$R = \frac{\|\nabla x_{\text{in}}\|}{\|\nabla y_{\text{out}}\|}$$

and raw gradient magnitudes at every step.

Our results empirically counter the conventional wisdom that explicit normalization statistics (mean μ, variance σ^2) are strictly necessary for stability. The DynamicTanh model demonstrated not only stable but remarkably disciplined training behavior. While the LayerNorm baseline maintained a mean gradient ratio of ≈ 0.24 (indicating a consistent damping of the backward signal), DyT exhibited a slightly stronger, yet perfectly safe, damping effect with a mean ratio of 0.19 (Figure: 12). At no point in the training run did we observe the chaotic spikes ($R \gg 1.0$) associated with exploding gradients.

Moreover, the effective strength of the learning signal was preserved. The average input gradient norm for DyT (3.34×10^{-3}) remained essentially on par with the baseline (3.63×10^{-3}) (Figure: 11). Thus, the optimizer was never deprived of informative gradients.

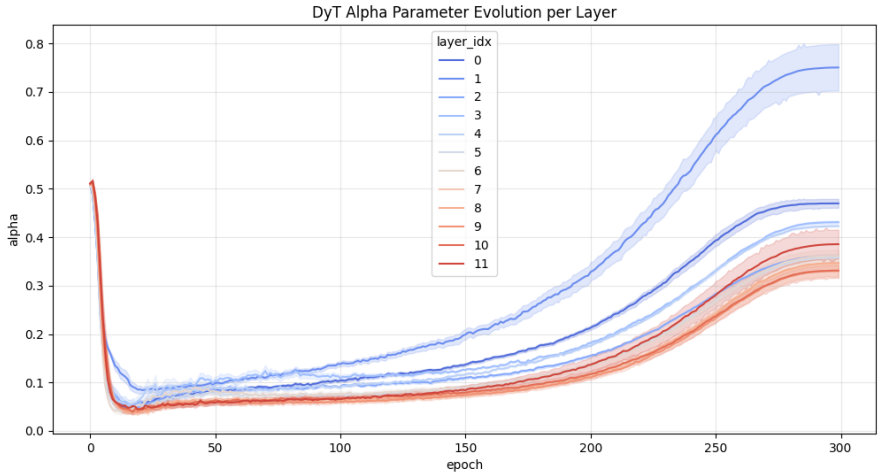

Figure 10: Evolution of α. The solid line represents the average α for each block, while the shaded band denotes the variance (range) between the two norms.

The key to this stability appears to be the **adaptive initialization** of α. During the volatile first 50 epochs, the network learned to decrease α to extremely low values (≈ 0.05–0.15). Mechanistically, this forces the activation function into a **near-linear mode**, acting as a "soft start" or "warm-up" mechanism that dampens the high-variance gradients typical during early training. As the weights stabilized, we observed α gradually rising, effectively opening the valve to allow more complex, nonlinear signals to flow. This demonstrates that a simple, learnable token-wise scalar is sufficient to regulate the backward pass, rendering the global synchronization of LayerNorm unnecessary for maintaining stability.

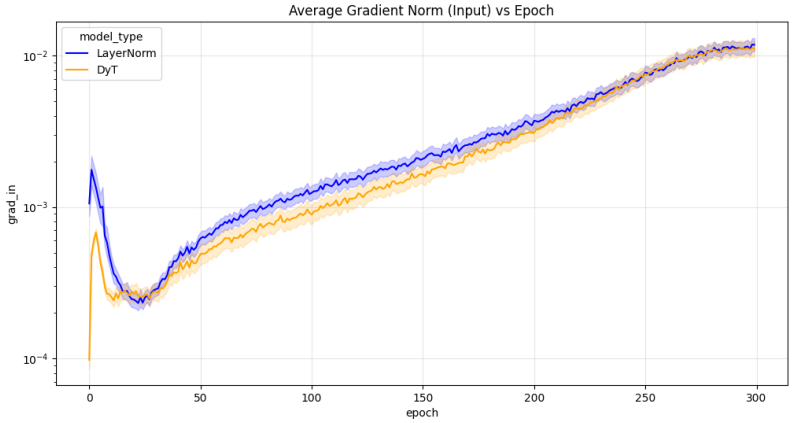

Figure 11: Grad Norms of LN & DyT

9 Computational Efficiency and System Performance

We benchmarked the system-level performance of DynamicTanh (DyT) versus the LayerNorm baseline to quantify the practical trade-offs between theoretical FLOP reductions and hardware execution. The benchmarks were conducted using JIT-compiled kernels (`torch.compile`) to minimize Python interpreter overhead and ensure a fair comparison against the vendor-optimized LayerNorm kernels.

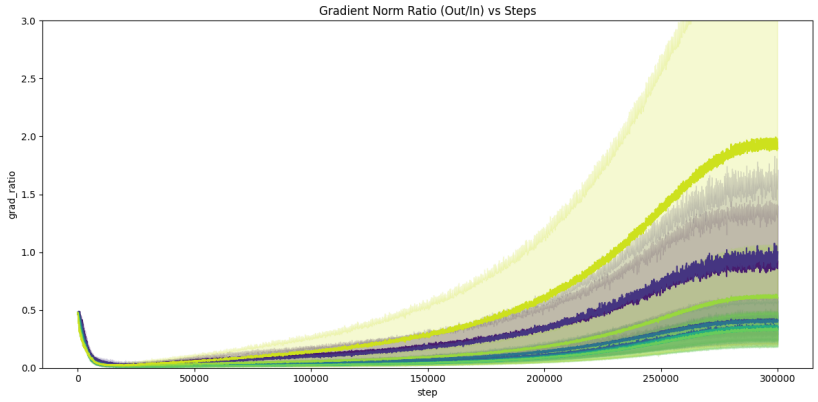

Figure 12: Output/Input Grad Norm of LN & DyT

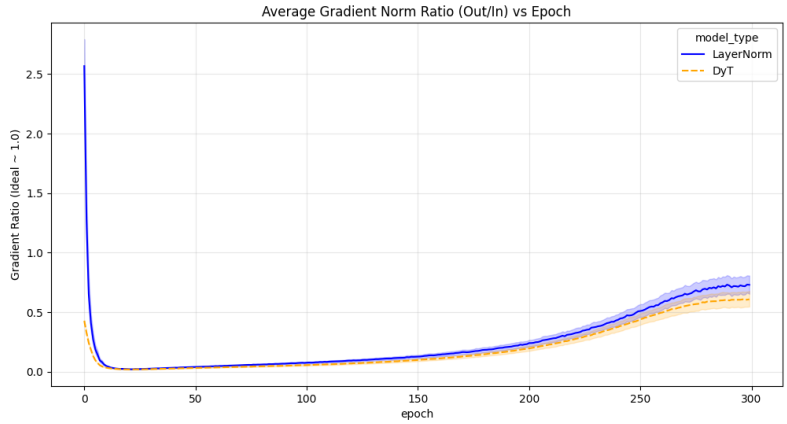

Figure 13: Gradient Stability Comparison

9.1 Throughput Analysis: The Case for Pointwise Operations

Our results indicate that DyT achieves **throughput parity** with the highly optimized LayerNorm baseline, validating its viability for high-performance training pipelines (Table: 4 , Figure: 14).

Metric	LayerNorm (LN)	DyT	Delta (%)
Training Throughput (img/s)	902.3	**912.7**	↑ 1.15%
Inference Throughput (img/s)	2789.0	**2837.6**	↑ 1.74%

Table 4: Throughput comparison between LayerNorm and DynamicTanh (DyT).

This performance profile is significant. Normalization layers are typically *memory-bandwidth bound* due to the requirement of reduction operations (calculating the mean μ and variance σ^2 across the channel dimension), which necessitate synchronization barriers and multi-pass memory access. By contrast, DyT relies strictly on *pointwise operations* ($x \cdot \alpha \to \tanh \to +\beta$), which are embarrassingly parallel and require no cross-element synchronization.

Our findings demonstrate that **even without a hand-tuned CUDA kernel**, a JIT-compiled DyT implementation successfully matches the speed of the vendor-optimized `cuDNN` LayerNorm kernel. This suggests that DyT eliminates the synchronization bottlenecks inherent to normalization, offering a scalable path for future hardware optimization.

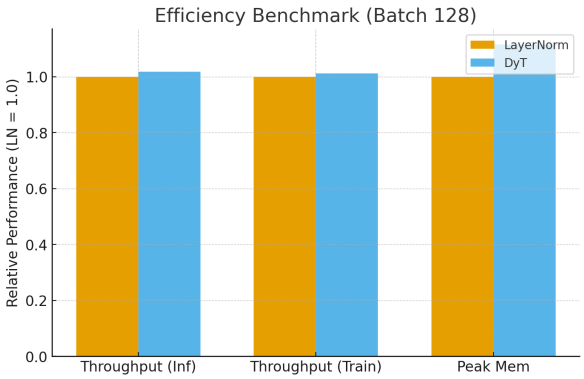

Figure 14: Benchmarking DyT and LN

9.2 Memory Usage: The Implementation Gap

Regarding memory consumption, our PyTorch-based implementation of DyT exhibited an **11.5% increase** in peak memory usage (**4240 MB** vs. **3801 MB**).

This result highlights an implementation gap; while DyT theoretically reduces memory by eliminating the storage of normalization statistics (mean/variance buffers), standard `Autograd` engines cache the input tensors for both the multiplication step ($\alpha \cdot x$) and the non-linearity step (tanh) separately to support backpropagation.

In contrast, the fused LayerNorm backward kernel is highly optimized to recompute or compress necessary gradients. We posit that a custom fused backward kernel (e.g., written in OpenAI Triton [Ope23] or CUDA [NVI24]) would be required to unlock the latent memory efficiency of DyT. Such a kernel would enable activation checkpointing at the operation level, preventing the caching of intermediate pointwise values and likely reversing this result to demonstrate a net memory reduction.

10 DynamicTanh(DyT) Variants

10.1 Dynamic Sigmoid (DyS)

Dynamic Sigmoid follows the same design philosophy as DyT but leverages the logistic function's $(0, 1)$ saturation to bound activations. The module applies

$$y = \gamma\,\sigma(a\,x) + \beta, \quad \sigma(z) = \frac{1}{1 + e^{-z}},$$

where a again adjusts the pre-activation scale and γ, β provide affine flexibility. Because sigmoid outputs are non-zero-centered, β plays a critical role in re-centering features before they enter residual paths. We integrate DyS into ViT-S identically to DyT, enabling a direct head-to-head comparison of saturation characteristics and gradient dynamics under a fixed training recipe.

10.2 Dynamic Softsign (DySs)

Dynamic Softsign combines the linear regime near zero with smooth saturation in the tails by computing

$$y = \gamma\,\frac{a\,x}{1 + |a\,x|} + \beta.$$

Softsign's gentler curvature yields more linear gradients for moderate activations, which can improve convergence in deep stacks. As with DyT and DyS, the learnable scaler a and affine parameters γ, β are initialized to 0.5, 1, and 0. The one-to-one replacement of LN with DySs in ViT-S ensures that any observed performance differences stem purely from the activation's shape and saturation behavior.

10.3 Dynamic Arctan (DyA)

Dynamic Arctan leverages the inverse tangent function to provide a sigmoidal activation with slower saturation characteristics than Tanh. The operation is defined as:

$$y = \gamma \arctan(\alpha x) + \beta.$$

Unlike the exponential decay of Tanh gradients, Arctan possesses heavy tails, meaning its gradients approach zero at a quadratic rate, $O(1/x^2)$. This substantially mitigates vanishing gradients for large activation magnitudes during early training. The output is intrinsically bounded to $(-\pi/2, \pi/2)$, but the affine scaling parameter γ allows the network to expand or contract this range as needed. Consistent with other variants, the parameters are initialized to $\alpha = 0.5$, $\gamma = 1$, and $\beta = 0$ to facilitate stable replacement of LayerNorm in the ViT-S architecture.

10.4 Dynamic Clipped Linear (DyCL)

Dynamic Clipped Linear (also referred to as Dynamic HardTanh) acts as a piecewise linear approximation of sigmoidal nonlinearities, prioritizing computational efficiency. It is computed as:

$$y = \gamma \cdot \text{clamp}(\alpha x, -1, 1) + \beta.$$

This activation produces a strictly linear regime whenever $|\alpha x| \leq 1$, and saturates with zero gradient outside this interval. The resulting hard transition forces the model to learn a binary gating behavior: features are either propagated linearly or clipped to a fixed magnitude. By replacing LayerNorm with DyCL, we evaluate whether the smooth curvature of Tanh is necessary for convergence, or if efficient piecewise linearity is sufficient for feature extraction in Vision Transformers.

10.5 Dynamic GELU-Clip (DyGC)

Dynamic GELU-Clip introduces the intrinsic non-monotonicity of the Gaussian Error Linear Unit (GELU) into a dynamically scaled and bounded framework. The formulation is:

$$y = \gamma \cdot \text{clamp}(\text{GELU}(\alpha x), -1, 1) + \beta.$$

Unlike monotonic activations such as Tanh or Sigmoid, GELU produces negative outputs for certain positive inputs, preserving controlled stochasticity in the forward pass. The clipping operation prevents unbounded activation growth while retaining the curvature and smoothness advantages of GELU near zero. This variant tests the hypothesis that non-monotonicity, combined with dynamic scaling via α, can enhance representational flexibility compared to standard sigmoidal functions.

10.6 Dynamic Parametric Rational (DyPR)

Dynamic Parametric Rational employs a rational function (a ratio of polynomials) to approximate nonlinear behavior without relying on computationally expensive transcendental operations such as $\exp(\cdot)$ or $\tanh(\cdot)$. We adopt a low-order Padé-style approximation:

$$y = \gamma \cdot \frac{\alpha x}{1 + |\alpha x|} + \beta.$$

This structure closely resembles Softsign but provides a hardware-efficient proxy capable of expressing a wide range of nonlinear shapes depending on the learned α. DyPR offers a balanced trade-off between expressivity and inference speed. Initializing $\alpha = 0.5$, $\gamma = 1$, and $\beta = 0$ ensures a well-behaved starting point, enabling the network to gradually evolve the activation curvature purely through gradient-based optimization.

Results

We tabulate the results obtained for each of the dynamic activation variant, measuring hte top-1 accuracy in Table: 5

Method	Top-1 Accuracy (%)	Δ from Baseline
Baseline (LayerNorm)	80.72	—
Dynamic Tanh (DyT)	**81.84**	**+1.12** ↑
Dynamic Sigmoid (DyS)	80.12	-0.60 ↓
Dynamic Softsign (DySs)	80.92	+0.20 ↑
Dynamic Arctan (DyA)	80.37	-0.35 ↓
Dynamic Clipped Linear (DyCL)	80.27	-0.45 ↓
Dynamic GELU-Clip (DyGC)	80.52	-0.20 ↓
Dynamic Parametric Rational (DyPR)	80.82	+0.10 ↑

Table 5: Top-1 accuracy comparison across Dynamic Activation variants

11 Conclusion

This study confirms that Dynamic Tanh (DyT) effectively replaces Layer Normalization, achieving 81.84% accuracy on ImageNet-100 without requiring cross-token synchronization. Our analysis reveals that DyT operates via a unique "compression-then-expansion" regime, utilizing learned saturation to filter noise in early layers rather than maintaining the high effective rank observed in normalized baselines. While DyT demonstrates superior isolation against sporadic noise, it trades off robustness against global additive bias due to the absence of mean-centering. Conclusively, we show that simple, scalar-gated nonlinearities are sufficient to regulate gradient flow, offering a scalable, statistic-free paradigm for efficient vision architecture design.

References

[Agg25] Vidit Aggarwal. Vit_lnablation: Repository for layernorm ablation in vision transformer. https://github.com/viditagg2005/ViT_LNAblation, 2025. GitHub repository.

[BDSS21] Andrew Brock, Kundan De, Simon Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[DBK+21] 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 16×16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[DS20] Soham De and Samuel L. Smith. Batch normalization biases deep residual networks towards shallow paths. *CoRR*, abs/2002.10444, 2020.

[HPBV20] Xiao Shi Huang, Felipe Pérez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4475–4483. PMLR, Jul 13–18 2020.

[Hug20] Hugging Face Datasets Team. ImageNet-100: A 100-class subset of imagenet-1k. https://huggingface.co/datasets/clane9/imagenet-100, 2020.

[IS15] 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)*, pages 448–456, 2015.

[KNLH19] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[NVI24] NVIDIA Corporation. Cuda toolkit documentation. https://developer.nvidia.com/cuda-toolkit, 2024. Version 12.x.

[Ope23] OpenAI. Triton: Open-source gpu programming for neural networks. https://github.com/openai/triton, 2023. Version 2.0.

[SK16] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[Wig19] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[ZS19] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems 32*, Vancouver, Canada, 2019.

A Shannon Effective Rank of an Activation Map

Let an activation matrix be $A \in R^{n \times d}$, and define its empirical covariance as

$$C = \frac{1}{n} A^\top A.$$

Let $\{\lambda_i\}_{i=1}^d$ be the eigenvalues of C, and normalize them into a probability distribution

$$p_i = \frac{\lambda_i}{\sum_{j=1}^d \lambda_j}, \qquad p_i \geq 0, \quad \sum_{i=1}^d p_i = 1.$$

The *spectral entropy* associated with the activation geometry is

$$H_{\text{spec}} = -\sum_{i=1}^d p_i \log p_i,$$

which measures how evenly the variance is distributed across principal components. The *Shannon Effective Rank* is then defined as

$$\text{ER}(A) = \exp(H_{\text{spec}}),$$

providing a continuous estimate of the intrinsic dimensionality of the representation, independent of eigenvalue thresholds or arbitrary cutoffs.

B Centered Kernel Alignment (CKA) Similarity

Given two activation matrices $X \in R^{n \times d_x}$ and $Y \in R^{n \times d_y}$ extracted from corresponding layers of two neural networks, Centered Kernel Alignment (CKA) provides a scale-invariant and rotation-invariant metric for comparing their representational structure. To begin, we construct the linear Gram matrices

$$K = XX^\top, \qquad L = YY^\top,$$

and apply double-centering using the centering matrix $H = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$:

$$\tilde{K} = HKH, \qquad \tilde{L} = HLH.$$

The Hilbert–Schmidt Independence Criterion (HSIC) is then defined as

$$\text{HSIC}(X, Y) = \text{tr}(\tilde{K}\tilde{L}),$$

Layer Depth	LayerNorm (LN)	DynamicTanh (DyT)
Block 0	81.5	54.0
Block 1	94.8	64.2
Block 2	112.0	75.5
Block 3	122.5	81.0
Block 4	130.0	87.0
Block 5	138.5	97.0
Block 6	149.0	109.0
Block 7	153.0	118.5
Block 8	160.0	126.5
Block 9	163.5	132.0
Block 10	167.0	140.0
Block 11	150.2	149.0

Table 6: Shannon Effective Rank across Transformer blocks for LN vs. DyT.

which measures covariance alignment in the feature space. CKA normalizes HSIC to obtain

$$\mathrm{CKA}(X, Y) = \frac{\mathrm{HSIC}(X, Y)}{\sqrt{\mathrm{HSIC}(X, X)\,\mathrm{HSIC}(Y, Y)}},$$

yielding a similarity score bounded between 0 and 1. Unlike raw correlation or canonical-correlation–based metrics, CKA is invariant to isotropic rescaling, orthogonal transformations, and variations in the number of channels, making it robust to differences in architecture and parameterization. In practice, linear CKA is computationally efficient ($O(nd^2)$ for a dense layer), stable for high-dimensional activations, and widely used to study layer-wise feature evolution, convergence behavior, and cross-model representational alignment.

C Hyperparameters

Hyperparameter	Value
Optimizer	AdamW
Base Learning Rate	1×10^{-3} (Cosine Decay)
Weight Decay	0.05
Optimizer Momentum	$\beta_1 = 0.9,\ \beta_2 = 0.999$
Batch Size	128
Training Epochs	300
Warmup Epochs	20
Data Augmentation	RandAugment, Mixup (0.8), Cutmix (1.0)
Label Smoothing	0.1
Drop Path Rate	0.1

Table 7: Training hyperparameters used for ViT-T experiments.

Method	**Init α**	**Init γ**	**Init β**	**Saturation**
Dynamic Tanh (DyT)	0.5	1.0	0.0	Exponential (Soft)
Dynamic Sigmoid (DyS)	0.5	1.0	0.0	Exponential (Soft)
Dynamic Softsign (DySs)	0.5	1.0	0.0	Polynomial (Soft)
Dynamic Arctan (DyA)	0.5	1.0	0.0	Quadratic (Heavy Tail)
Dynamic Clipped Linear (DyCL)	0.5	1.0	0.0	Hard (Zero Grad)
Dynamic GELU-Clip (DyGC)	0.5	1.0	0.0	Hard (Zero Grad)
Dynamic Rational (DyPR)	0.5	1.0	0.0	Polynomial (Adaptive)

Table 8: Initialization and saturation characteristics of dynamic activation variants.