Transformers without Normalization

Jiachen Zhu[1,2] Xinlei Chen[1] Kaiming He[3] Yann LeCun[1,2] Zhuang Liu[1,4,†]

[†]project lead

[1]FAIR, Meta [2]New York University [3] MIT [4]Princeton University

Abstract

Normalization layers are ubiquitous in modern neural networks and have long been considered essential. This work demonstrates that Transformers without normalization can achieve the same or better performance using a remarkably simple technique. We introduce Dynamic Tanh (DyT), an element-wise operation $\mathrm{DyT}(\boldsymbol{x}) = \tanh(\alpha \boldsymbol{x})$*, as a drop-in replacement for normalization layers in Transformers. DyT is inspired by the observation that layer normalization in Transformers often produces tanh-like, S-shaped input-output mappings. By incorporating DyT, Transformers without normalization can match or exceed the performance of their normalized counterparts, mostly without hyperparameter tuning. We validate the effectiveness of Transformers with DyT across diverse settings, from recognition to generation, supervised to self-supervised learning, and computer vision to language models. These findings challenge the conventional understanding that normalization layers are indispensable in modern neural networks, and offer new insights into their role in deep networks.*

1. Introduction

Over the past decade, normalization layers have solidified their positions as one of the most fundamental components of modern neural networks. It all traces back to the invention of batch normalization in 2015 [41], which enabled drastically faster and better convergence in visual recognition models and quickly gained momentum in the following years. Since then, many variants of normalization layers have been proposed for different network architectures or domains [5, 81, 83, 89]. Today, virtually all modern networks use normalization layers, with layer normalization (Layer Norm, or LN) [5] being one of the most popular, particularly in the dominant Transformer architecture [20, 82].

The widespread adoption of normalization layers is largely driven by their empirical benefits in optimization [10, 68]. In addition to achieving better results, they help accelerate and stabilize convergence. As neural networks become wider and deeper, this necessity becomes ever more critical [11, 38]. Consequently, normalization layers are widely regarded as crucial, if not indispensable, for effective training of deep networks. This belief is subtly evidenced by the fact that, in recent years, novel architectures often seek to replace attention or convolution layers [22, 29, 75, 78], but almost always retain normalization.

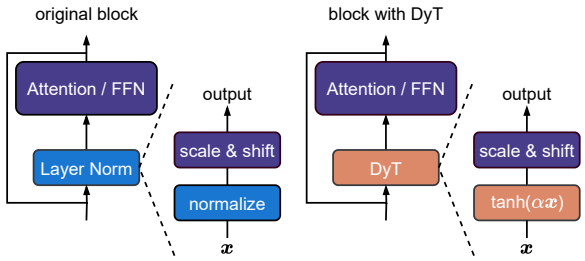

Figure 1. *Left:* original Transformer block. *Right:* block with our proposed Dynamic Tanh (DyT) layer. DyT is a straightforward replacement for commonly used Layer Norm [5] (in some cases RMSNorm [89]) layers. Transformers with DyT match or exceed the performance of their normalized counterparts.

This paper challenges this belief by introducing a simple alternative to normalization layers in Transformers. Our exploration starts with the observation that LN layers map their inputs to outputs with tanh-like, S-shaped curves, scaling the input activations while squashing the extreme values. Inspired by this insight, we propose an element-wise operation termed Dynamic Tanh (DyT), defined as: $\mathrm{DyT}(\boldsymbol{x}) = \tanh(\alpha \boldsymbol{x})$, where α is a learnable parameter. This operation aims to emulate the behavior of LN by learning an appropriate scaling factor through α and squashing extreme values via the bounded tanh function. Notably, unlike normalization layers, it achieves both effects without the need to compute activation statistics.

Employing DyT is straightforward, as shown in Figure 1: we directly replace existing normalization layers with DyT in architectures such as vision and language Transformers. We empirically demonstrate that models with DyT can train stably and achieve high final performance across a wide range of settings. It often does not require tuning the hy-

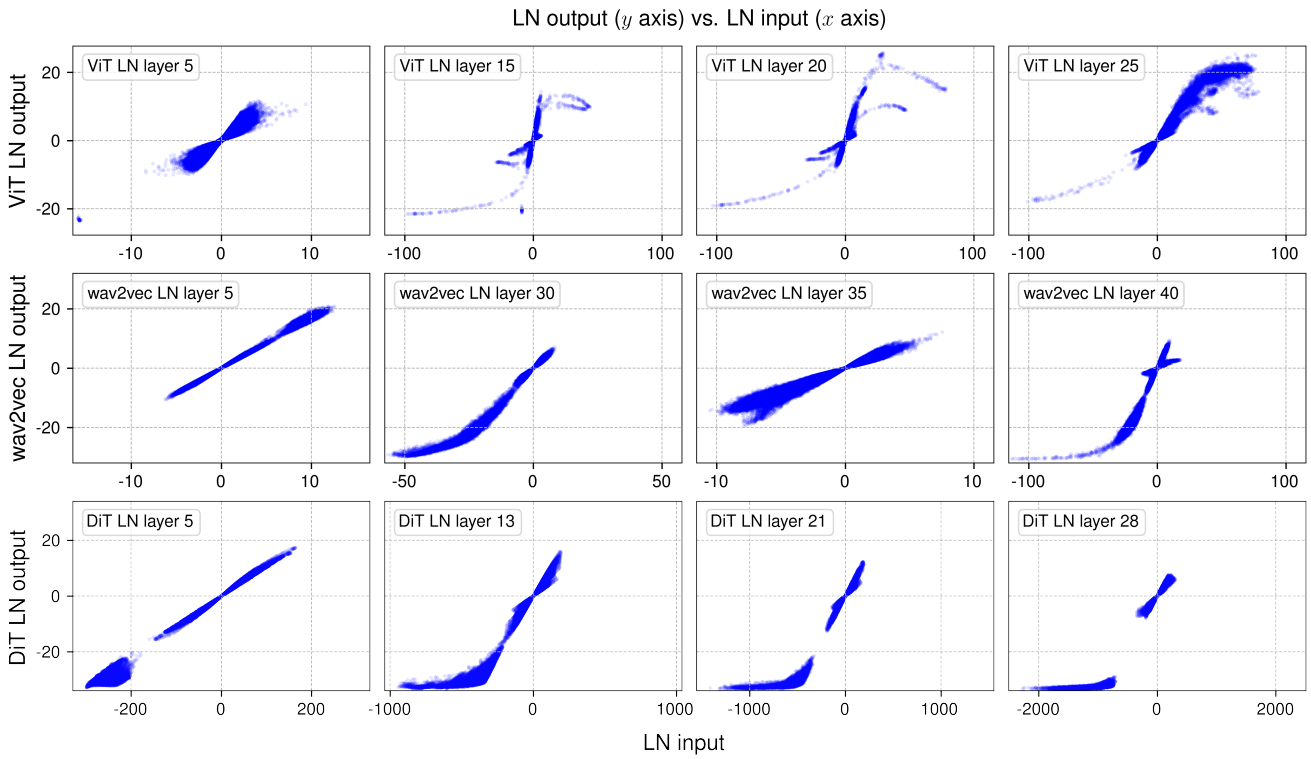

Figure 2. **Output vs. input of selected layer normalization (LN) layers in Vision Transformer (ViT) [20], wav2vec 2.0 (a Transformer model for speech) [7], and Diffusion Transformer (DiT) [64].** We sample a mini-batch of samples and plot the input / output values of four LN layers in each model. The outputs are before the affine transformation in LN. The S-shaped curves highly resemble that of a tanh function. The more linear shapes in earlier layers can also be captured by the center part of a tanh curve. This motivates us to propose Dynamic Tanh (DyT) as a replacement, with a learnable scaler α to account for different scales on the x axis.

perparameters on the original architecture. Our work challenges the notion that normalization layers are indispensable for training modern neural networks and provides empirical insights into the properties of normalization layers.

2. Background: Normalization Layers

We begin by reviewing the normalization layers. Most normalization layers share a common formulation. Given an input $\boldsymbol{x}$ with shape (B, T, C), where B is the batch size, T is the number of tokens, and C is the embedding dimension per token, the output is generally computed as:

$$\text{normalization}(\boldsymbol{x}) = \boldsymbol{\gamma} * \left(\frac{\boldsymbol{x} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} \right) + \boldsymbol{\beta} \qquad (1)$$

where ϵ is a small constant, and $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable vector parameters of shape $(C,)$. They are "scaling" and "shifting" affine parameters that allow the output to be in any range. The terms $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ denote the mean and variance of the input. Different methods mainly differ in how these two statistics are computed. This results in $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ having different dimensions, each with appropriate broadcasting needed in the tensor calculation.

Batch normalization (BN) [41] is the first modern normalization layer, and it has been primarily used in ConvNet models [33, 76, 84]. Its introduction represents a major milestone in deep learning architecture designs. BN computes the mean and variance across both the batch and token dimensions, specifically: $\mu_k = \frac{1}{BT}\sum_{i,j} x_{ijk}$ and $\sigma_k^2 = \frac{1}{BT}\sum_{i,j}(x_{ijk} - \mu_k)^2$. Other normalization layers popular in ConvNets, such as group normalization [83] and instance normalization [81], were initially proposed for specialized tasks such as object detection and image stylization. They share the same overall formulation but differ in the axes and ranges over which the statistics are computed.

Layer normalization (LN) [5] and root mean square normalization (RMSNorm) [89] are the major two types of normalization layers used in Transformer architectures. LN computes these statistics independently for each token in each sample, where $\mu_{ij} = \frac{1}{C}\sum_k x_{ijk}$ and $\sigma_{ij}^2 = \frac{1}{C}\sum_k (x_{ijk} - \mu_{ij})^2$. RMSNorm [89] simplifies LN by removing the mean-centering step and normalizing the input with $\mu_{ij} = 0$ and $\sigma_{ij}^2 = \frac{1}{C}\sum_k x_{ijk}^2$. Today, most modern neural networks use LN due to its simplicity and universality. Recently, RMSNorm has gained popularity, particularly in language models like T5 [66], LLaMA [21, 79, 80], Mistral [43], Qwen [8, 87], InternLM [13, 91] and DeepSeek [30, 49]. The Transformers we examine in this work all use LN, except that LLaMA uses RMSNorm.

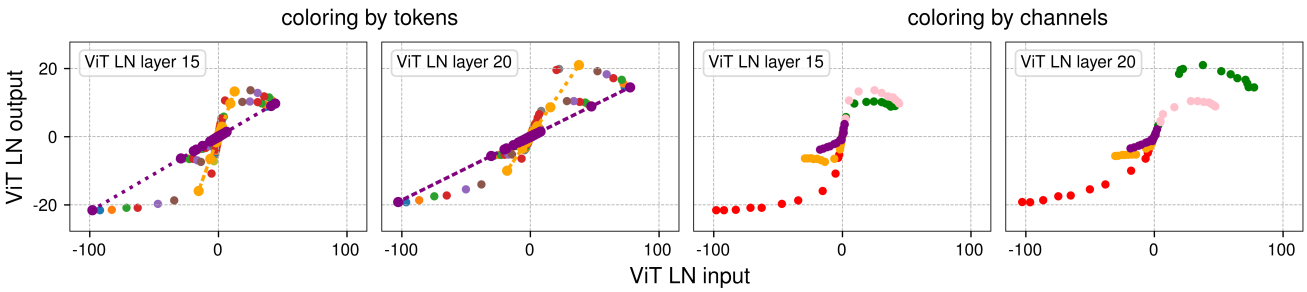

Figure 3. **Output vs. input of two LN layers, with tensor elements colored to indicate different channel and token dimensions.** The input tensor has a shape of (samples, tokens, and channels), with elements visualized by assigning consistent colors to the same tokens (left two panels) and channels (right two panels). *Left two panels*: points representing the same token (same color) form straight lines across different channels, as LN operates linearly across channels for each token. Interestingly, when plotted collectively, these lines form a non-linear tanh-shaped curve. *Right two panels*: each channel's input spans different ranges on the x-axis, contributing distinct segments to the overall tanh-shaped curve. Certain channels (e.g., red, green, and pink) exhibit more extreme x values, which are squashed by LN.

3. What Do Normalization Layers Do?

Analysis setup. We first empirically study the behaviors of normalization layers in trained networks. For this analysis, we take a Vision Transformer model (ViT-B) [20] trained on ImageNet-1K [19], a wav2vec 2.0 Large Transformer model [7] trained on LibriSpeech [63], and a Diffusion Transformer (DiT-XL) [64] trained on ImageNet-1K. In all cases, LN is applied in every Transformer block and before the final linear projection.

For all three trained networks, we sample a mini-batch of samples and do a forward pass through the network. We then measure the input and output for the normalization layers, i.e., tensors immediately before and after the normalization operation, before the learnable affine transformation. Since LN preserves the dimensions of the input tensor, we can establish a one-to-one correspondence between input and output elements, allowing a direct visualization of their relationship. We plot the resulting mappings in Figure 2.

Tanh-like mappings with layer normalization. For all three models, in earlier LN layers (1st column of Figure 2), we find this input-output relationship to be mostly linear, resembling a straight line in an x-y plot. However, deeper LN layers are where we make more intriguing observations.

A striking observation from these deeper layers is that most of these curves' shapes resemble full or partial S-shaped curves represented by a tanh function. One might expect LN layers to linearly transform the input tensor, as subtracting the mean and dividing by standard deviation are linear operations. LN normalizes in a per-token manner, only linearly transforming each token's activations. As tokens have different mean and standard deviation values, the linearity does not hold collectively on all activations of the input tensor. Nonetheless, it is surprising that the non-linear transformation is highly similar to a scaled tanh function.

For such an S-shaped curve, we note that the central part, represented by points with x values close to zero, is mainly in a linear shape. Most points ($\sim$99%) fall in this linear range. However, there are many points that clearly fall out of this range, which are considered to have "extreme" values, e.g., those with x larger than 50 or smaller than -50 in the ViT model. Normalization layers' main effect for these values is to *squash* them into less extreme values, more in line with the majority of points. This is where normalization layers could not approximated by a simple affine transformation layer. We hypothesize this non-linear and disproportional squashing effect on extreme values is what makes normalization layers important and indispensable.

Recent findings by Ni et al. [62] similarly highlight the strong non-linearities introduced by LN layers, demonstrating how the non-linearity enhances a model's representational capacity. Moreover, this squashing behavior mirrors the saturation properties of biological neurons for large inputs, a phenomenon observed about a century ago [1–3].

Normalization by tokens and channels. How does an LN layer perform a linear transformation for each token but also squash the extreme values in such a non-linear fashion? To understand this, we visualize the points grouped by tokens and channels, respectively. This is plotted in Figure 3 by taking the second and third subplots for ViT from Figure 2, but with a sampled subset of points for more clarity. When we select the channels to plot, we make sure to include the channels with extreme values.

On the left two panels of Figure 3, we visualize each token's activations using the same color. We observe that all points from any single token do form a straight line. However, since each token has a different variance, the slopes are different. Tokens with smaller input x ranges tend to have smaller variance, and the normalization layer will divide their activations using a smaller standard deviation, hence producing a larger slope in the straight line. Collectively, they form an S-shaped curve that resembles a tanh function. In the two panels on the right, we color each channel's activations using the same color. We find that different channels tend to have drastically different input ranges, with only a few channels (e.g., red, green, and pink) exhibiting large extreme values. These are the channels that get squashed the most by the normalization layer.

4. Dynamic Tanh (DyT)

Inspired by the similarity between shapes of normalization layers and a scaled tanh function, we propose Dynamic Tanh (DyT) as a replacement for normalization layers. Given input tensor $\boldsymbol{x}$, a DyT layer is defined as follows:

$$\mathrm{DyT}(\boldsymbol{x}) = \boldsymbol{\gamma} * \tanh(\alpha \boldsymbol{x}) + \boldsymbol{\beta} \tag{2}$$

where α is a learnable scalar parameter that allows scaling the input differently based on its range, accounting for varying x scales (Figure 2). This is also why we name the whole operation "Dynamic" Tanh. $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable, per-channel vector parameters, the same as those used in all normalization layers—they allow the output to scale back to any scales. This is sometimes considered a separate affine layer; for our purposes, we consider them to be part of the DyT layer, just like how normalization layers also include them. See Algorithm 1 for Pytorch-like pseudocode of DyT.

Algorithm 1 Pseudocode of DyT layer.

```
# input x has the shape of [B, T, C]
# B: batch size, T: tokens, C: dimension
class DyT(Module):
    def __init__(self, C, init_α):
        super().__init__()
        self.α = Parameter(ones(1) * init_α)
        self.γ = Parameter(ones(C))
        self.β = Parameter(zeros(C))

    def forward(self, x):
        x = tanh(self.alpha * x)
        return self.γ * x + self.β
```

Integrating DyT layers into an existing architecture is straightforward: one DyT layer replaces one normalization layer (see Figure 1). This applies to normalization layers within attention blocks, FFN blocks, and the final normalization layer. Although DyT may look like or be considered an activation function, this study only uses it to replace normalization layers without altering parts of the activation functions in the original architectures, such as GELU or ReLU. Other parts of the network remain intact. We also observe that there is minimal need to tune the hyperparameters for the original architectures for DyT to perform well.

On scaling parameters. We always simply initialize $\boldsymbol{\gamma}$ to an all-one vector and $\boldsymbol{\beta}$ to an all-zero vector following normalization layers. For the scaler parameter α, a default initialization of 0.5 is generally sufficient, except for LLM training. A detailed analysis of α initialization is provided in Section 7. Unless explicitly stated otherwise, α is initialized to 0.5 in our subsequent experiments.

Remarks. DyT is *not* a new type of normalization layer, as it operates on each input element from a tensor independently during a forward pass without computing statistics or other types of aggregations. It does, however, preserve the effect of normalization layers in squashing the extreme values in a non-linear fashion while almost linearly transforming the very central parts of the input.

5. Experiments

To demonstrate the effectiveness of DyT, we experiment with Transformers and a few other modern architectures across a diverse range of tasks and domains. In each experiment, we replace the LN or RMSNorm in the original architectures with DyT layers and follow the official open-source protocols to train and test both versions of the models. Detailed instructions for reproducing our results are provided in Appendix A. Notably, to highlight the simplicity of adapting DyT, we use hyperparameters identical to those used by the normalized counterparts. For completeness, additional results regarding tuning of learning rates and initial values of α are provided in Appendix F.

Supervised learning in vision. We train Vision Transformer (ViT) [20] and ConvNeXt [50] of Base" and Large" sizes on the ImageNet-1K classification task [19]. These models are selected due to their popularity and distinct operations: attention in ViT and convolution in ConvNeXt. Table 1 reports the top-1 classification accuracies. DyT performs slightly better than LN across both architectures and model sizes. We further plot the training loss for ViT-B and ConvNeXt-B in Figure 4. The curves show that the convergence behaviors of DyT and LN-based models are aligned.

model	LN	DyT	change
ViT-B	82.3%	82.5%	↑0.2%
ViT-L	83.1%	83.6%	↑0.5%
ConvNeXt-B	83.7%	83.7%	-
ConvNeXt-L	84.3%	84.4%	↑0.1%

Table 1. **Supervised classification accuracy on ImageNet-1K.** DyT achieves better or similar performance than LN across both architectures and model sizes.

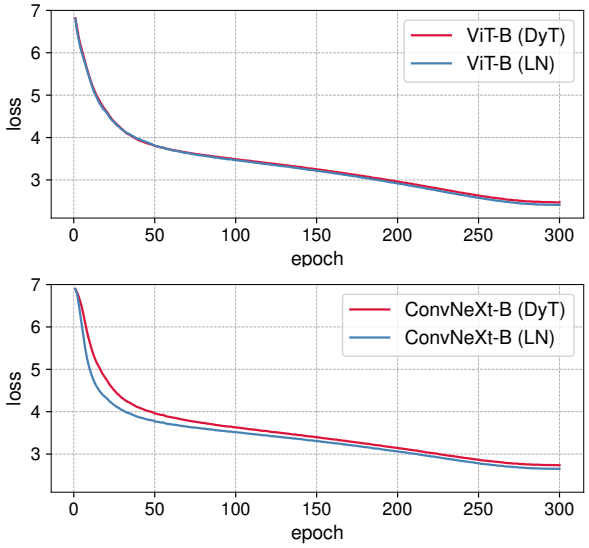

Figure 4. **Training loss curves for ViT-B and ConvNeXt-B.** The curves for both models exhibit similar patterns between LN and DyT, suggesting that they may share similar learning dynamics.

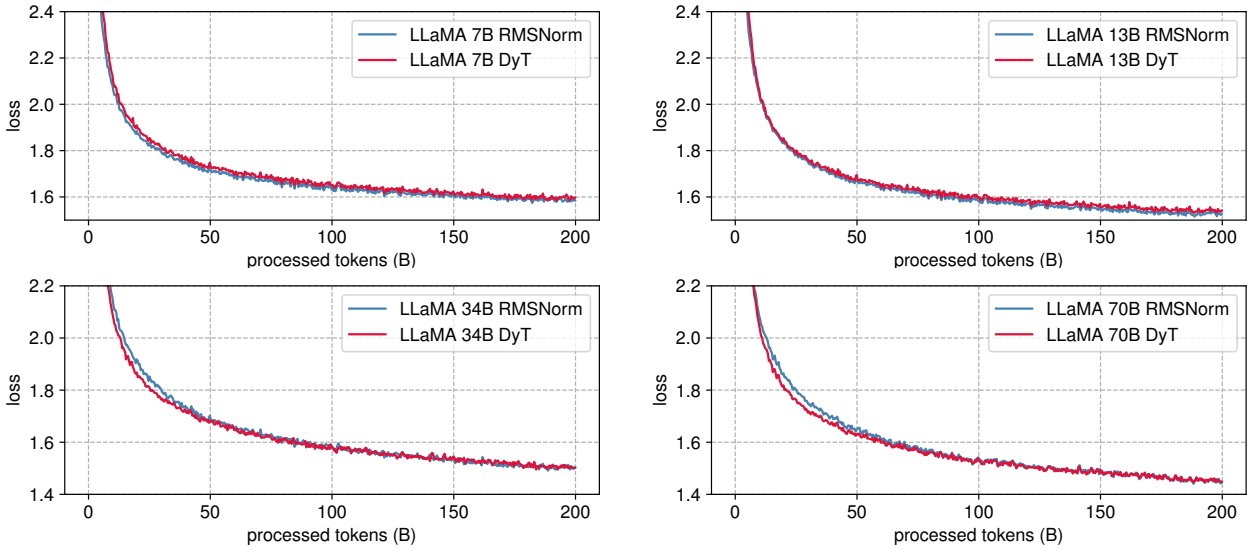

Figure 5. **LLaMA pretraining loss.** The loss curves of DyT and RMSNorm models are closely aligned across model sizes.

Self-supervised learning in vision. We benchmark with two popular visual self-supervised learning methods: masked autoencoders (MAE) [34] and DINO [14]. Both by default use Vision Transformers as backbones, but have different training objectives: MAE is trained with a reconstruction loss, and DINO uses a joint-embedding loss [47]. Following the standard self-supervised learning protocol, we first pretrain models on ImageNet-1K without using labels and then test the pretrained models by attaching a classification layer and fine-tuning them with labels. The fine-tuning results are presented in Table 2. DyT consistently performs on par with LN in self-supervised learning tasks.

model	LN	DyT	change
MAE ViT-B	83.2%	83.2%	-
MAE ViT-L	85.5%	85.4%	↓0.1%
DINO ViT-B (patch size 16)	83.2%	83.4%	↑0.2%
DINO ViT-B (patch size 8)	84.1%	84.5%	↑0.4%

Table 2. **Self-supervised learning accuracy on ImageNet-1K.** DyT performs on par with LN across different methods and sizes.

Diffusion models. We train three Diffusion Transformer (DiT) models [64] of sizes B, L and XL on ImageNet-1K [19]. The patch size is 4, 4, and 2, respectively. Note that in DiT, the LN layers' affine parameters are used for class conditioning in DiT, and we keep them that way in our DyT experiments, only replacing the normalizing transformation with the $\tanh(\alpha \boldsymbol{x})$ function. After training, we evaluate the Fréchet Inception Distance (FID) scores using the standard ImageNet "reference batch", as presented in Table 3. DyT achieves comparable or improved FID over LN.

model	LN	DyT	change
DiT-B	64.9	63.9	↓1.0
DiT-L	45.9	45.7	↓0.2
DiT-XL	19.9	20.8	↑0.9

Table 3. **Image generation quality (FID) on ImageNet.** DyT achieves comparable FID scores to LN across various model sizes.

Large Language Models. We pretrain LLaMA 7B, 13B, 34B, and 70B models [21, 79, 80] to assess DyT performance relative to RMSNorm [89], the default normalization layer used in LLaMA. The models are trained on The Pile dataset [25] with 200B tokens, following the original recipe outlined in LLaMA [80]. On LLaMA with DyT, we add a learnable scalar parameter after the initial embedding layer, and adjust the initial value of α, as detailed in Section 7. We report the loss value after training and also follow OpenLLaMA [26] to benchmark the models on 15 zero-shot tasks from `lm-eval` [24]. As shown in Table 4, DyT performs on par with RMSNorm across all four model sizes. Figure 5 illustrates the loss curves, demonstrating similar trends across all model sizes, with training losses closely aligned throughout training.

score / loss	RMSNorm	DyT	change
LLaMA 7B	0.513 / 1.59	0.513 / 1.60	- / ↑0.01
LLaMA 13B	0.529 / 1.53	0.529 / 1.54	- / ↑0.01
LLaMA 34B	0.536 / 1.50	0.536 / 1.50	- / -
LLaMA 70B	0.549 / 1.45	0.549 / 1.45	- / -

Table 4. **Language models' training loss and average performance with 15 zero-shot `lm-eval` tasks.** DyT achieves a comparable zero-shot performance and training loss to RMSNorm.

Self-supervised learning in speech. We pretrain two wav2vec 2.0 Transformer models [7] on the LibriSpeech dataset [63]. We retain the first group normalization layer, as it functions as data normalization to handle the unnormalized input data. Table 5 reports the final validation loss. We observe that DyT performs comparably to LN.

model	LN	DyT	change
wav2vec 2.0 Base	1.95	1.95	-
wav2vec 2.0 Large	1.92	1.91	↓0.01

Table 5. **Speech pretraining validation loss on LibriSpeech.** DyT performs comparably to LN for both wav2vec 2.0 models.

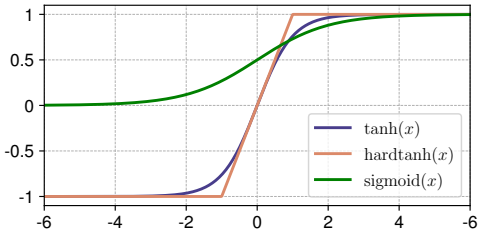

Figure 6. Curves of three squashing functions: tanh, hardtanh, and sigmoid. All three functions squash inputs into a bounded range, but tanh achieves the best performance when used in DyT layers. We suspect it is due to its smoothness and zero-centered properties.

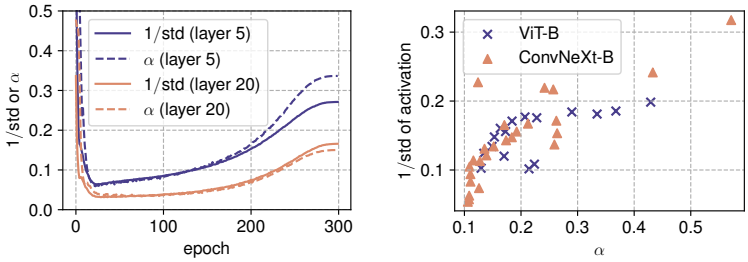

Figure 7. *Left:* For two selected DyT layers from the ViT-B model, we track α and the inverse of the standard deviation ($1/\text{std}$) of activations at the end of each epoch, observing that they evolve together during training. *Right:* We plot the final α values of two models, ViT-B and ConvNeXt-B, against the $1/\text{std}$ of the input activations, demonstrating a strong correlation between the two values.

DNA sequence modeling. On the long-range DNA sequence modeling task, we pretrain the HyenaDNA model [60] and the Caduceus model [69]. The pretraining uses human reference genome data from [27], and the evaluation is on GenomicBenchmarks [28]. Results are presented in Table 6. DyT maintains performance comparable to LN.

model	LN	DyT	change
HyenaDNA [60]	85.2%	85.2%	-
Caduceus [69]	86.9%	86.9%	-

Table 6. **DNA classification accuracy on GenomicBenchmarks**, averaged over each dataset in GenomicBenchmarks. DyT achieves comparable performance to LN.

6. Analysis

We conduct two studies examining the roles of the tanh function and the learnable scale α.

6.1. Ablations of tanh and α

To further investigate the role of tanh and α in DyT, we conduct experiments to evaluate the model's performance when these components are altered or removed.

Replacing and removing tanh. We replace tanh in DyT layers with alternative squashing functions, specifically hardtanh and sigmoid (Figure 6), while keeping the learnable scaler α intact. Furthermore, we assess the impact of completely removing tanh by replacing it with the identity function while still retaining α. As shown in Table 7, the squashing function is essential for stable training. Using the identity function leads to unstable training and divergence, whereas squashing functions enable stable training. Among the squashing functions, tanh performs the best. This is possibly due to its smoothness and zero-centered properties.

model	identity	tanh	hardtanh	sigmoid
ViT-S	58.5% $\rightarrow$ failed	**80.3%**	79.9%	79.6%
ViT-B	61.0% $\rightarrow$ failed	**82.5%**	82.2%	81.6%

Table 7. **ImageNet-1K classification accuracy with different squashing functions.** All experiments follow the same training recipe as the original LN-based models. Squashing functions play a crucial role in preventing divergence, with tanh achieving the highest performance among the three functions. "$\rightarrow$ failed" indicates that training diverged after some progress, with the preceding number representing the accuracy reached before divergence.

Removing α. Next, we evaluate the impact of removing the learnable α while retaining the squashing functions. As shown in Table 8, removing α results in performance degradation across all squashing functions, highlighting the critical role of α in overall model performance.

model	tanh	hardtanh	sigmoid
without α	81.1%	80.7%	80.7%
with α	**82.5%**	**82.2%**	**81.6%**

Table 8. **ImageNet-1K classification accuracy with ViT-B.** All experiments follow the same training recipe as the original LN-based models. α is essential for enhancing model performance.

6.2. Values of α

During training. Our analysis reveals that the α closely tracks the $1/\text{std}$ of activations throughout training. As illustrated in the left panel of Figure 7, α first decrease and then increase during training, but always fluctuate consistently with the standard deviation of activations. This supports the important role of α in maintaining activations within a suitable range, which leads to stable and effective training.

After training. Our further analysis of the final values of α in trained networks reveals a strong correlation with the $1/\text{std}$ of the input activations. As shown on the right panel of Figure 7, higher $1/\text{std}$ values generally correspond to larger α values, and vice versa. Additionally, we observe that deeper layers tend to have activations with larger standard deviations. This trend aligns with characteristics of deep residual networks, as shown in Brock et al. [11] for ConvNets, and Sun et al. [74] for Transformers.

Both analyses suggest that α acts as a normalization mechanism, learning values close to $1/\text{std}$ of the input activations. Unlike LN, which normalizes per token, α normalizes the entire input collectively and cannot suppress extreme values non-linearly.

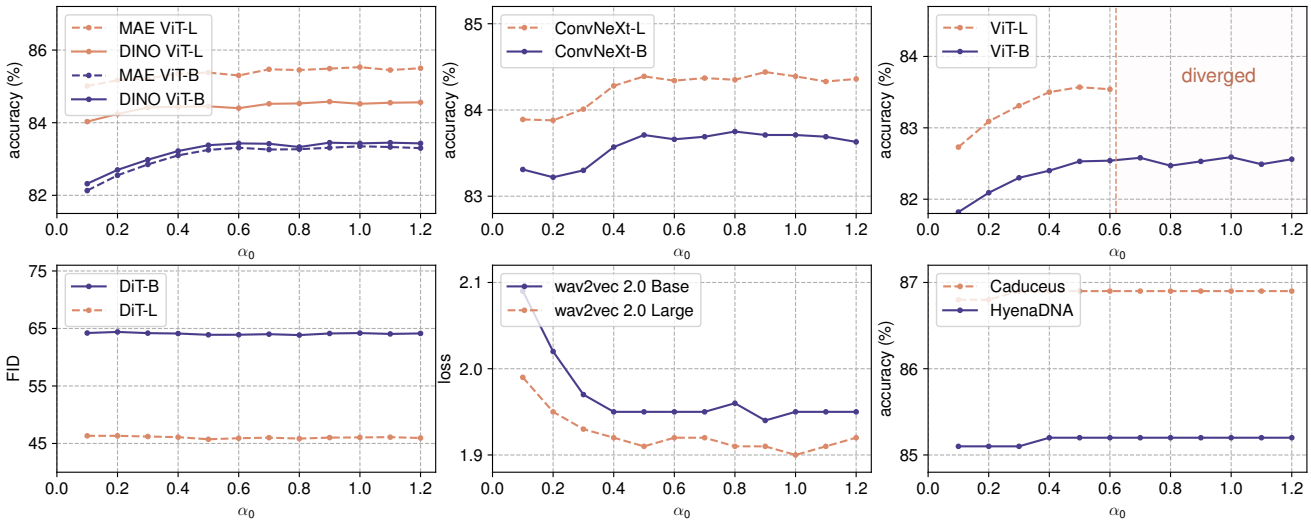

Figure 8. **Performance of different tasks across different α_0 values.** We benchmark the performance of all non-LLM tasks used in Section 5 with different initial values of α. Performance remains stable across a wide range of α_0 values. The only exception is that supervised ViT-L models (top right panel) will diverge for α_0 values larger than 0.6.

7. Initialization of α

We find that tuning the initialization of α (denoted α_0) rarely leads to significant performance improvements. The only exception is LLM training, where careful tuning of α_0 yields noticeable performance gains. In this section, we detail our findings on the impact of α initialization.

7.1. Initialization of α for Non-LLM Models

Non-LLM models are relatively insensitive to α_0. Figure 8 shows the effect of varying α_0 on validation performance across different tasks. All experiments follow the original setup and hyperparameters of their respective recipe. We observe that performance remains stable across a wide range of α_0 values, with values between 0.5 and 1.2 generally yielding good results. We observe that adjusting α_0 typically affects only the early stages of the training curves. The main exception is supervised ViT-L experiments, where training becomes unstable and diverges when α_0 exceeds 0.6. In such cases, reducing the learning rate restores stability, as detailed below.

Smaller α_0 results in more stable training. Building on previous observations, we further analyze the factors contributing to training instability. Our findings suggest that increasing either the model size or the learning rate requires lowering α_0 to ensure stable training. Conversely, a higher α_0 requires a lower learning rate to mitigate training instability. Figure 9 shows the ablation of the training stability of supervised ViT with ImageNet-1K dataset. We vary learning rates, model sizes, and α_0 values. Training a larger model is more prone to failure, requiring smaller α_0 values or learning rates for stable training. A similar instability pattern is also observed in LN-based models under comparable conditions, and setting $\alpha_0 = 0.5$ results in a stability pattern similar to that of LN.

Setting $\alpha_0 = 0.5$ as the default. Based on our findings, we set $\alpha_0 = 0.5$ as the default value for all non-LLM models. This setting provides training stability comparable to LN while maintaining strong performance.

7.2. Initialization of α for LLMs

Tuning α_0 enhances LLM performance. As discussed earlier, the default setting of $\alpha_0 = 0.5$ generally performs well across most tasks. However, we find tuning α_0 can substantially improve LLM performance. We tune α_0 across LLaMA models by pretraining each on 30B tokens and comparing their training losses. Table 9 summarizes the tuned α_0 values for each model. Two key findings emerge:

1. **Larger models require smaller α_0 values.** Once the optimal α_0 is determined for smaller models, the search space for larger models can be reduced accordingly.
2. **Higher α_0 values for attention blocks improve performance.** We find that initializing α with higher values for DyT layers in attention blocks and lower values for DyT layers in other locations (i.e., within FFN blocks or before the final linear projection) improves performance.

model	width	depth	optimal α_0 (attention/other)
LLaMA 7B	4096	32	0.8/0.2
LLaMA 13B	5120	40	0.6/0.15
LLaMA 34B	8196	48	0.2/0.05
LLaMA 70B	8196	80	0.2/0.05

Table 9. **Optimal α_0 for different LLaMA models.** Larger models require smaller α_0 values. We find it is important to initialize α differently in (1) attention blocks ("attention"), versus (2) the FFN blocks, and the final DyT layer before outputs ("other"). α_0 in attention blocks require larger values.

Further details and visualizations of the tuning of the initial value of α of LLMs are provided in the appendix C.

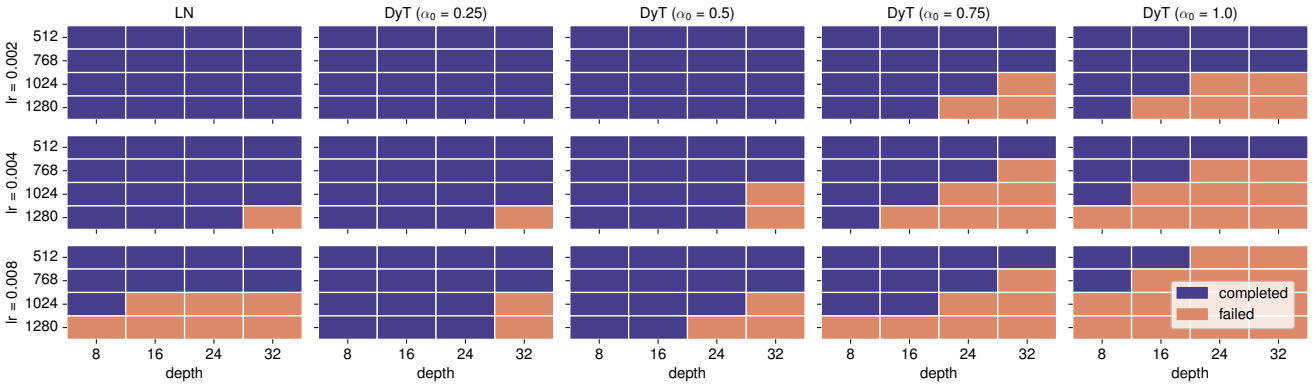

Figure 9. **Stability across varying α_0 values, learning rates, and model sizes.** We train supervised ViT models on the ImageNet-1K dataset and observe that larger models are more prone to instability for both LN and DyT models. Lowering the learning rate or reducing α_0 enhances stability. LN shows similar stability to DyT with $\alpha_0 = 0.5$.

8. Related Work

Mechanisms of normalization layers. There has been a rich line of work investigating normalization layers' role in enhancing model performance through various mechanisms. These include stabilizing gradient flow during training [9, 17, 52], reducing sensitivity to weight initialization [18, 70, 90], moderating outlier eigenvalues [10, 44], auto-tuning learning rates [4, 77], and smoothing the loss landscape for more stable optimization [68]. These earlier works focused on studying batch normalization. Recent studies [16, 53, 59] further highlight the connection between normalization layers and sharpness reduction, which contributes to better generalization.

Normalization in Transformers. With the rise of Transformer [82], research has increasingly focused on layer normalization [5], which has proven particularly effective for sequential data in natural language tasks [61, 85, 86]. Recent work [62] reveals that layer normalization introduces strong non-linearity, enhancing the model's representational capacity. Additionally, studies [48, 51] demonstrate that modifying the location of normalization layers within Transformers can improve convergence properties.

Removing normalization. Many studies have explored how to train deep models without normalization layers. Several works [6, 18, 90] explore alternative weight initialization schemes to stabilize training. Self-normalizing networks [45] introduce scaled exponential linear units (SELUs) and a carefully chosen initialization scheme to maintain stable activations and gradient flow without the need for explicit normalization layers. The pioneering work by Brock et al. [11, 12] show that high-performing ResNets can be trained without normalization [72] through combination of initialization techniques [18], weight normalization [37, 65, 67], and adaptive gradient clipping [12]. Additionally, their training strategy incorporates extensive data augmentation [15] and regularization [36, 73]. The studies above are based on various ConvNet models.

In Transformer architectures, He and Hofmann [32] explore modifications to Transformer blocks that reduce reliance on normalization layers and skip connections. Jha and Reagen [42] introduce AERO, a Softmax-only LLM that improves inference efficiency and privacy with minimal performance loss. Alternatively, Heimersheim [35] propose a method to gradually remove LN from pretrained networks by fine-tuning the model after removing each normalization layer. Unlike previous approaches, DyT requires minimal modifications to both the architecture and the training recipe. Despite its simplicity, DyT achieves stable training and comparable performance.

9. Limitations

We conduct experiments on networks using either LN or RMSNorm because of their popularity in Transformers and other modern architectures. Preliminary experiments (see Appendix E) indicate that DyT struggles to replace BN directly in classic networks like ResNets. It remains to be studied in more depth whether and how DyT can adapt to models with other types of normalization layers.

10. Conclusion

In this work, we demonstrate modern neural networks, in particular Transformers, can be trained without normalization layers. This is done through Dynamic Tanh (DyT), a simple replacement for traditional normalization layers. It adjusts the input activation range via a learnable scaling factor α and then squashes the extreme values through an S-shaped tanh function. Although a simpler function, it effectively captures the behavior of normalization layers. Under various settings, models with DyT match or exceed the performance of their normalized counterparts. The findings challenge the conventional understanding of the necessity of normalization layers in training modern neural networks. Our study also contributes to understanding the mechanisms of normalization layers, one of the most fundamental building blocks in deep neural networks.

References

[1] Edgar D Adrian. The impulses produced by sensory nerve endings: Part 1. *The Journal of Physiology*, 1926.

[2] Edgar D Adrian and Yngve Zotterman. The impulses produced by sensory nerve-endings: Part 2. the response of a single end-organ. *The Journal of Physiology*, 1926.

[3] Edgar D Adrian and Yngve Zotterman. The impulses produced by sensory nerve endings: Part 3. impulses set up by touch and pressure. *The Journal of Physiology*, 1926.

[4] Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. Theoretical analysis of auto rate-tuning by batch normalization. *arXiv preprint arXiv:1812.03981*, 2018.

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

[6] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *UAI*, 2021.

[7] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *NeurIPS*, 2020.

[8] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[9] David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *ICML*, 2017.

[10] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. *NeurIPS*, 2018.

[11] Andrew Brock, Soham De, and Samuel L Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. *arXiv preprint arXiv:2101.08692*, 2021.

[12] Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In *ICML*, 2021.

[13] Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*, 2024.

[14] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021.

[15] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR Workshops*, 2020.

[16] Yan Dai, Kwangjun Ahn, and Suvrit Sra. The crucial role of normalization in sharpness-aware minimization. *NeurIPS*, 2024.

[17] Hadi Daneshmand, Jonas Kohler, Francis Bach, Thomas Hofmann, and Aurelien Lucchi. Batch normalization provably avoids ranks collapse for randomly initialised deep networks. *NeurIPS*, 2020.

[18] Soham De and Sam Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *NeurIPS*, 2020.

[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[21] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[22] Leo Feng, Frederick Tung, Mohamed Osama Ahmed, Yoshua Bengio, and Hossein Hajimirsadegh. Were rnns all we needed? *arXiv preprint arXiv:2410.01201*, 2024.

[23] Foundation Model Stack. Github: FMS FSDP. https://github.com/foundation-model-stack/fms-fsdp. Accessed: 2025-01-23.

[24] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation.

[25] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[26] Xinyang Geng and Hao Liu. Openllama: An open reproduction of llama, 2023.

[27] Ensembl GRCh38. p13 (genome reference consortium human build 38), insdc assembly, 2013.

[28] Katarína Grešová, Vlastimil Martinek, David Čechák, Petr Šimeček, and Panagiotis Alexiou. Genomic benchmarks: a collection of datasets for genomic sequence classification. *BMC Genomic Data*, 2023.

[29] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[30] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[31] HazyResearch. Github: Hyenadna. https://github.com/HazyResearch/hyena-dna.git. Accessed: 2025-01-23.

[32] Bobby He and Thomas Hofmann. Simplifying transformer blocks. *arXiv preprint arXiv:2311.01906*, 2023.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[34] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

[35] Stefan Heimersheim. You can remove gpt2's layernorm by fine-tuning. *arXiv preprint arXiv:2409.13710*, 2024.

[36] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[37] Lei Huang, Xianglong Liu, Yang Liu, Bo Lang, and Dacheng Tao. Centered weight normalization in accelerating training of deep neural networks. In *ICCV*, 2017.

[38] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application. *TPAMI*, 2023.

[39] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *ICML*, 2020.

[40] Hugging Face. Hugging Face: LLaMA 2. https://huggingface.co/meta-llama/Llama-2-7b-hf. Accessed: 2025-01-23.

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

[42] Nandan Kumar Jha and Brandon Reagen. Aero: Softmax-only llms for efficient private inference. *arXiv preprint arXiv:2410.13060*, 2024.

[43] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[44] Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. The normalization method for alleviating pathological sharpness in wide neural networks. *NeurIPS*, 2019.

[45] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *NeurIPS*, 2017.

[46] Kuleshov Group. Github: Caduceus. https://github.com/kuleshov-group/caduceus.git. Accessed: 2025-01-23.

[47] Yann LeCun. A path towards autonomous machine intelligence version 0.9.2, 2022-06-27. *Open Review*, 2022.

[48] Pengxiang Li, Lu Yin, and Shiwei Liu. Mix-ln: Unleashing the power of deeper layers by combining pre-ln and post-ln. *arXiv preprint arXiv:2412.13795*, 2024.

[49] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[50] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, 2022.

[51] Ilya Loshchilov, Cheng-Ping Hsieh, Simeng Sun, and Boris Ginsburg. ngpt: Normalized transformer with representation learning on the hypersphere. *arXiv preprint arXiv:2410.01131*, 2024.

[52] Ekdeep S Lubana, Robert Dick, and Hidenori Tanaka. Beyond batchnorm: Towards a unified understanding of normalization in deep learning. *NeurIPS*, 2021.

[53] Kaifeng Lyu, Zhiyuan Li, and Sanjeev Arora. Understanding the generalization benefit of normalization layers: Sharpness reduction. *NeurIPS*, 2022.

[54] Meta Research. Github: ConvNeXt. https://github.com/facebookresearch/ConvNeXt, . Accessed: 2025-01-23.

[55] Meta Research. Github: DINO. https://github.com/facebookresearch/dino, . Accessed: 2025-01-23.

[56] Meta Research. Github: DiT. https://github.com/facebookresearch/DiT, . Accessed: 2025-01-23.

[57] Meta Research. Github: MAE. https://github.com/facebookresearch/mae, . Accessed: 2025-01-23.

[58] Meta Research. Github: wav2vec 2.0. https://github.com/facebookresearch/fairseq, . Accessed: 2025-01-23.

[59] Maximilian Mueller, Tiffany Vlaar, David Rolnick, and Matthias Hein. Normalization layers are all that sharpness-aware minimization needs. *NeurIPS*, 2024.

[60] Eric Nguyen, Michael Poli, Marjan Faizi, Armin Thomas, Michael Wornow, Callum Birch-Sykes, Stefano Massaroli, Aman Patel, Clayton Rabideau, Yoshua Bengio, et al. Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution. *NeurIPS*, 2024.

[61] Toan Q Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. *arXiv preprint arXiv:1910.05895*, 2019.

[62] Yunhao Ni, Yuxin Guo, Junlong Jia, and Lei Huang. On the nonlinearity of layer normalization. *arXiv preprint arXiv:2406.01255*, 2024.

[63] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *ICASSP*, 2015.

[64] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

[65] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Micro-batch training with batch-channel normalization and weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.

[66] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020.

[67] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *NeurIPS*, 2016.

[68] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *NeurIPS*, 2018.

[69] Yair Schiff, Chia-Hsiang Kao, Aaron Gokaslan, Tri Dao, Albert Gu, and Volodymyr Kuleshov. Caduceus: Bi-

directional equivariant long-range dna sequence modeling. *arXiv preprint arXiv:2403.03234*, 2024.

[70] Jie Shao, Kai Hu, Changhu Wang, Xiangyang Xue, and Bhiksha Raj. Is normalization indispensable for training deep neural network? *NeurIPS*, 2020.

[71] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[72] Samuel L Smith, Andrew Brock, Leonard Berrada, and Soham De. Convnets match vision transformers at scale. *arXiv preprint arXiv:2310.16764*, 2023.

[73] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.

[74] Wenfang Sun, Xinyuan Song, Pengxiang Li, Lu Yin, Yefeng Zheng, and Shiwei Liu. The curse of depth in large language models. *arXiv preprint arXiv:2502.05795*, 2025.

[75] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

[76] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[77] Hidenori Tanaka and Daniel Kunin. Noether's learning dynamics: Role of symmetry breaking in neural networks. *NeurIPS*, 2021.

[78] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *NeurIPS*, 2021.

[79] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[80] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[81] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[83] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

[84] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[85] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *ICML*, 2020.

[86] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. *NeurIPS*, 2019.

[87] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[88] Shuangfei Zhai, Tatiana Likhomanenko, Etai Littwin, Dan Busbridge, Jason Ramapuram, Yizhe Zhang, Jiatao Gu, and Joshua M Susskind. Stabilizing transformer training by preventing attention entropy collapse. In *ICML*, 2023.

[89] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *NeurIPS*, 2019.

[90] Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

[91] Pan Zhang, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Rui Qian, Lin Chen, Qipeng Guo, Haodong Duan, Bin Wang, Linke Ouyang, et al. Internlm-xcomposer-2.5: A versatile large vision language model supporting long-contextual input and output. *arXiv preprint arXiv:2407.03320*, 2024.