LENS: REPLACING LAYERNORM AND SOFTMAX IN TRANSFORMERS WITH LEARNABLE NORMALIZATION STATES

Anonymous authors
Paper under double-blind review

ABSTRACT

LayerNorm and attention *Softmax* are critical operations in Transformers. However, they cannot be efficiently mapped to *General Matrix Multiplication* (GEMM) computations on hardware. Therefore, modern hardware accelerator designs, such as NPUs on ASIC/FPGA, consume a large amount of hardware resources or chip area to implement their specific arithmetic units. Building on previous work about *LayerNorm*, *Softmax*, and their alternatives, we introduce *Learnable Normalization States* (LENS), a framework that reformulates *LayerNorm* as tiny MLPs by abstracting a common structure from related methods and extends this method to *Softmax*, which moves more of the computation in *Transformers* to GEMM as a hardware-friendly architecture. We evaluate LENS on image classification with *Vision Transformers* (ViT) and autoregressive language modeling with *Pythia* models, including generalization tests on *WikiText* and *LAMBADA*. Across these tasks, LENS matches or exceeds baseline performance in several vision settings and retains most of the baseline performance in the remaining vision and language experiments. These findings may suggest that hand-designed normalization and attention functions are not the only choice as alternatives to *LayerNorm* and *Softmax*. LENS also provides a basis for exploring Transformer architectures for resource-limited accelerators.

1 INTRODUCTION

Transformer (Vaswani et al., 2017) has been widely adopted in recent research as a foundational network architecture across various scenarios. Following the rapid expansion of model sizes in *Transformer*-based networks for vision and language tasks, massive compute requirements have driven and accelerated the development of hardware accelerator designs. Considering that *General Matrix Multiplication* (GEMM) is the most important and intensive computation workload during network inference, modern neural network hardware accelerators, whether *Graphics Processing Units* (GPUs) or *Neural Processing Units* (NPUs), usually consume a massive amount of hardware resources or chip area to implement the matrix computation core, such as the *Tensor Cores* on *NVIDIA* GPUs (NVIDIA Corporation, 2025) and *Systolic Arrays* on NPUs.

However, in the *Transformer* architecture, *Layer Normalization* (*LayerNorm*) (Ba et al., 2016) and *Softmax* in the attention layers are two heterogeneous nonlinear operators that cannot be directly mapped to GEMM on hardware. As a result, hardware accelerators for *Transformer* have to implement the additional special arithmetic units for *LayerNorm* and *Softmax* or map them onto the existing functional cores. For instance, *NVIDIA* GPUs accelerate the reciprocal, square root, and exponential operations in *LayerNorm* and *Softmax* on *Special Function Units* (SFUs) and execute the rest of the standard arithmetic operations on CUDA cores (NVIDIA Corporation, 2009). NPUs also need to customize the specific *LayerNorm* and *Softmax* units to further accelerate related computations on *Application-Specific Integrated Circuits* (ASICs) or *Field-Programmable Gate Arrays* (FPGAs).

As a result, designing a hardware accelerator for *Transformers* involves a critical trade-off: allocating more hardware resources or chip area to implementing GEMM computation cores can speed up linear-layer inference but may leave *LayerNorm*, *Softmax*, and other heterogeneous nonlinear

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

Table 1: Hardware Resource Consumption of Systolic Array, LayerNorm, and Softmax Reported by Chen et al. (2024)

Kernel	Hardware Resource Consumption			Relative to Systolic Array		
	LUTs	FFs	DSPs	LUTs	FFs	DSPs
Systolic Array with DSP Packing	244439	79969	128	100.00%	100.00%	100.00%
Softmax	7447	4835	38	3.05%	6.05%	29.69%
LayerNorm	12746	18751	80	5.21%	23.45%	62.50%
Softmax + 2×LayerNorm	32939	42337	198	13.48%	52.94%	154.69%

operators as a performance bottleneck. Allocating more hardware resources or chip area to heterogeneous nonlinear operators can reduce their inference latency. However, it may also reduce the available area for implementing matrix-computation cores and slow linear-layer inference. For instance, Chen et al. (2024) successfully implemented the FPGA-based *Large Language Model* (LLM) accelerator for *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin et al., 2019) and *Generative Pre-trained Transformer 2* (GPT-2) (Radford et al., 2019) on an *AMD Xilinx Alveo U280* FPGA device. They report that the *LayerNorm* and *Softmax* units do not significantly increase the overall latency in their fully pipelined design. However, as shown in Table 1, the hardware resources consumed by two *LayerNorm* units and one *Softmax* unit in their accelerator are equivalent to 13.48% of the LUTs, 52.94% of the FFs, and 154.69% of the DSPs used by a single systolic-array kernel. Therefore, given that the vast majority of computation in *Transformers* is linear and can be mapped to GEMMs, this example suggests that hardware accelerators may still need to allocate a massive amount of hardware resources to prevent the *LayerNorm* and *Softmax* from becoming a performance bottleneck and increasing the inference latency, even though they account for only a small fraction of the total computation in *Transformer* inference.

Previous work, such as *Root Mean Square Layer Normalization* (RMSNorm) (Zhang & Sennrich, 2019), *Dynamic Tanh* (DyT) (Zhu et al., 2025), *Dynamic erf* (Derf) (Chen et al., 2026), *SeeDNorm* (Cai et al., 2026), *SimA* (Koohpayegani & Pirsiavash, 2024), and *I-ViT* (Li & Gu, 2023), has widely explored the different variants, alternatives, and approximations to simplify the *LayerNorm* and *Softmax*. Considering that most of the above-mentioned works match or exceed the performance of the native *Transformer* architecture in different vision and language tasks, we therefore rethink the related works and have the following questions:

- If those different replacements of *LayerNorm* and *Softmax* work properly in *Transformers*, is there any shared underlying mechanism across some of these different methods?
- If there is a shared underlying mechanism across these methods, can we follow it and let training determine the appropriate functions to replace *LayerNorm* and *Softmax*, rather than relying on hand-designed mathematical formulas?
- Is it possible to reformulate the heterogeneous nonlinear operators of *LayerNorm* and *Softmax* to map more computations to GEMMs?

Following these questions, we present the *Learnable Normalization States* (LENS) in this work to replace both *LayerNorm* and *Softmax* with tiny *Multilayer Perceptrons* (MLPs). The following are our contributions in this work:

- We first outlined the mathematical formulas for *LayerNorm* and its alternatives and discovered that they can be unified into a common functional form $\mathbf{a}(\mathbf{x})\odot\mathbf{x}+\mathbf{b}(\mathbf{x})$. We therefore adopted a tiny MLP with a two-neuron output layer to replace it. And then, we extended a similar mechanism to the *Softmax* and also replaced it with two tiny MLPs.
- We tested the LENS architecture across various vision and language tasks using different datasets. The results show that LENS recovers most of the baseline performance and even exceeds it in several vision settings after retraining.
- We further conducted a preliminary analysis of the LENS training process, as well as the weight changes and function plots observed before and after replacing *LayerNorm* and *Softmax*. The results show that LENS is not merely a simple approximation of *LayerNorm* and *Softmax*. Instead, it underwent joint adaptation with the model weights during retraining, significantly changing some weight values and thereby learning a different functional mapping.

Our work gives a proof of concept suggesting that hand-designed mathematical functions are not the only choices for replacing *LayerNorm* and *Softmax*. LENS also provides a possible GEMM-centric, hardware-friendly architecture that can shift more nonlinear computation in *Transformers* onto GEMMs on accelerators.

2 Mechanism Analysis and LENS

2.1 Related Work

To explore the potential shared mechanism, we summarized some representative works first, starting with methods that simplify or alter the *LayerNorm*:

- *RMSNorm* (Zhang & Sennrich, 2019) is a classic design in related works that removes the mean-centering operation in *LayerNorm* and normalizes the input using the *Root Mean Square* (RMS).
- Zhu et al. (2025) present the *Dynamic Tanh* (DyT) to replace *LayerNorm* with an element-wise $\tanh(\alpha \mathbf{x}) \odot \gamma + \beta$, where α is a learnable scale factor, γ and β are learnable affine parameters. Therefore, DyT has no input-statistics computation, such as mean and variance, in *LayerNorm*.
- *Dynamic erf* (Derf) (Chen et al., 2026) extends DyT by replacing *tanh* with *erf* and adding one additional learnable bias factor s. The function of Derf becomes $\mathrm{erf}(\alpha \mathbf{x} + s) \odot \gamma + \beta$.
- *SeeDNorm* (Cai et al., 2026) extends the *RMSNorm*, adding a dynamically adjusted scaling coefficient based on the current input to the γ in *RMSNorm*.

And the following are some representative alternatives to *Softmax*:

- *SimA* (Koohpayegani & Pirsiavash, 2024) replaces *Softmax* by applying ℓ_1 normalization to each feature dimension of the query and key matrices across tokens as $\widehat{\mathbf{Q}}$ and $\widehat{\mathbf{K}}$. The attention output is then directly computed as $\widehat{\mathbf{Q}}\widehat{\mathbf{K}}^{\mathsf{T}}\mathbf{V}$.
- Wortsman et al. (2023) present the *ReLU Attention*, which replaces *Softmax* with an element-wise *ReLU* function applied to $\mathbf{Q}\mathbf{K}^{\mathsf{T}}/\sqrt{d}$ and divides the result by the sequence length N. The complete formula is $\mathrm{ReLU}(\mathbf{Q}\mathbf{K}^{\mathsf{T}}/\sqrt{d})/N$.
- Lu et al. (2021) present the *SOFT* to replace scaled dot-product *Softmax* attention with an unnormalized *Gaussian* kernel similarity function. It further applies the *Nyström* low-rank decomposition and approximates the required *Moore-Penrose* inverse via *Newton-Raphson* iterations, achieving linear complexity with respect to the sequence length.

Furthermore, *I-ViT* (Li & Gu, 2023) proposes integer-only approximations to *Softmax* and *LayerNorm* for quantized *Transformers*. It rewrites the exponential function in *Softmax* from e^x to $2^{x \log_2 e}$ and adopts $1 + \frac{1}{2} - \frac{1}{16}$ to approximate the $\log_2 e$, which can be converted into additions and bitwise shifting. And for the *LayerNorm*, it computes the mean and variance using integer arithmetic and adopts a fixed 10-step integer Newton iteration to approximate the square root. *I-ViT* does not replace the functions of *Softmax* and *LayerNorm*, but approximates them for integer-only *Transformer* inference.

2.2 Converting LayerNorm to LENSNorm

Summarizing the representative *LayerNorm* alternatives listed in Section 2.1, we find that: *RMSNorm* and *SeeDNorm* remove the variance in *LayerNorm* but still require computing the reciprocal, mean, and square root. DyT and Derf replace the reciprocal, mean, variance, and square root with *tanh* or *erf*, both of which are nonlinear functions. Furthermore, none of these four methods can be simply mapped to GEMMs either.

Therefore, we rethink the formula of *LayerNorm*, $\frac{\mathbf{x}-\mathrm{Mean}(\mathbf{x})}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}} \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$, and find that we can re-organize it in the format of $\mathbf{a}(\mathbf{x}) \odot \mathbf{x} + \mathbf{b}(\mathbf{x})$, where $\mathbf{a}(\mathbf{x})$ denotes $\frac{\boldsymbol{\gamma}}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}}$ and $\mathbf{b}(\mathbf{x})$ denotes

Table 2: Unified formulation of *LayerNorm* and its alternatives as $\mathbf{a}(\mathbf{x}) \odot \mathbf{x} + \mathbf{b}(\mathbf{x})$ in *LENSNorm*.

Methods	Original	$\mathbf{a}(\mathbf{x})$	$\mathbf{b}(\mathbf{x})$
$\mathrm{LayerNorm}(\mathbf{x})$	$\frac{\mathbf{x}-\mathrm{Mean}(\mathbf{x})}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}} \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$	$\frac{\boldsymbol{\gamma}}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}}$	$\frac{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}\,\boldsymbol{\beta}-\mathrm{Mean}(\mathbf{x})\,\boldsymbol{\gamma}}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}}$
$\mathrm{RMSNorm}(\mathbf{x})$	$\frac{\mathbf{x}}{\sqrt{\mathrm{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}} \odot \boldsymbol{\gamma}$	$\frac{\boldsymbol{\gamma}}{\sqrt{\mathrm{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}}$	0
$\mathrm{DyT}(\mathbf{x})$	$\tanh(\alpha\mathbf{x}) \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$	$\alpha\boldsymbol{\gamma} - \frac{\alpha^3}{3}\boldsymbol{\gamma} \odot \mathbf{x}^{\odot 2}$	$\boldsymbol{\beta}$
$\mathrm{Derf}(\mathbf{x})$	$\mathrm{erf}(\alpha\mathbf{x}+s) \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$	$\mathcal{R}(\alpha\mathbf{x}+s) \odot \alpha\boldsymbol{\gamma}$	$\mathcal{R}(\alpha\mathbf{x}+s) \odot s\boldsymbol{\gamma} + \boldsymbol{\beta}$
$\mathrm{SeeDNorm}(\mathbf{x})$	$\frac{\mathbf{x}}{\sqrt{\mathrm{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}} \odot \left[\sigma\left(\mathbf{x}\boldsymbol{\beta}^{\mathsf{T}}\right)\boldsymbol{\alpha} + \boldsymbol{\gamma}\right]$	$\frac{\sigma\left(\mathbf{x}\boldsymbol{\beta}^{\mathsf{T}}\right)\boldsymbol{\alpha}+\boldsymbol{\gamma}}{\sqrt{\mathrm{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}}$	0
$\mathrm{LENSNorm}(\mathbf{x})$	$a(\mathbf{x})\mathbf{x} + b(\mathbf{x})$	$a(\mathbf{x})$	$b(\mathbf{x})$

$\frac{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}\,\boldsymbol{\beta}-\mathrm{Mean}(\mathbf{x})\,\boldsymbol{\gamma}}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}}$. Following the same methods, as shown in Table 2, we can also convert the formulas of *RMSNorm* and *SeeDNorm* into the format of $\mathbf{a}(\mathbf{x}) \odot \mathbf{x} + \mathbf{b}(\mathbf{x})$. Moreover, if we approximate *tanh* as $\tanh(\mathbf{x}) \approx \mathbf{x} - \frac{\mathbf{x}^3}{3}$ based on third-order *Taylor Expansion* around zero and rationally approximate *erf* as $\mathrm{erf}(\mathbf{x}) \approx \mathrm{sign}(\mathbf{x})\left(1 - \frac{1}{(1+a_1|\mathbf{x}|+a_2|\mathbf{x}|^2+a_3|\mathbf{x}|^3+a_4|\mathbf{x}|^4)^4}\right)$, where $a_1 = 0.278393$, $a_2 = 0.230389$, $a_3 = 0.000972$, and $a_4 = 0.078108$ (Abramowitz & Stegun, 1964), DyT and Derf can also be unified to the format of $\mathbf{a}(\mathbf{x}) \odot \mathbf{x} + \mathbf{b}(\mathbf{x})$. The $\mathcal{R}(z)$ in Table 2 denotes $\mathcal{R}(z) = \frac{1-(1+a_1|z|+a_2|z|^2+a_3|z|^3+a_4|z|^4)^{-4}}{|z|}$. The detailed mathematical derivation process can be found in Appendix A.1.

Based on the above-mentioned mathematical conversion, we therefore unified *LayerNorm* and its representative alternatives in the common format of $\mathbf{a}(\mathbf{x}) \odot \mathbf{x} + \mathbf{b}(\mathbf{x})$. Inspired by this common format, we propose *LENSNorm*, which uses two scalar normalization states $a(\mathbf{x})$ and $b(\mathbf{x})$ for input-dependent scaling and bias, in contrast to the vector-valued functions $\mathbf{a}(\mathbf{x})$ and $\mathbf{b}(\mathbf{x})$ above.

We parameterize these two scalar states with a tiny MLP as Equation 1 consisting of two *Fully Connected* (FC) linear layers. To limit additional parameters and computational overhead and avoid unnecessary over-parameterization, the first FC layer in LENSNorm has 32 neurons, followed by a *ReLU* activation. The second FC layer has only two neurons that generate the scalar normalization states $a(\mathbf{x})$ and $b(\mathbf{x})$. To maintain numerical stability, $a(\mathbf{x})$ is constrained to $[0.05, 12]$ using clamping, while $b(\mathbf{x})$ remains unconstrained. The two states are then broadcast across all hidden dimensions of the input token.

$$\begin{bmatrix} a(\mathbf{x}) \\ b(\mathbf{x}) \end{bmatrix} = \mathrm{MLP_N}(\mathbf{x}), \qquad \mathrm{LENSNorm}(\mathbf{x})_i = \left[\mathrm{MLP_N}(\mathbf{x})\right]_1 x_i + \left[\mathrm{MLP_N}(\mathbf{x})\right]_2 \tag{1}$$

2.3 Converting Softmax to LENSMax

Summarizing the representative *Softmax* alternatives listed in Section 2.1, we find that: *SimA* applies ℓ_1 normalization to each feature dimension of the query and key matrices before the score computation of $\mathbf{QK}^{\mathsf{T}}$. *ReLU Attention* applies the element-wise *ReLU* activation after the computation of $\mathbf{QK}^{\mathsf{T}}/\sqrt{d}$. *SOFT* directly replaces the *Softmax* with an unnormalized *Gaussian* kernel and applies *Newton–Raphson* iterations to approximate the required matrix inverse. Furthermore, computing the unnormalized *Gaussian* kernel still involves exponential functions. As with the alternatives to *LayerNorm*, most alternatives to *Softmax* cannot be directly mapped to GEMMs. Considering that existing Softmax alternatives modify different stages of attention computation and do not exhibit a clear shared computational structure, instead of forcing these methods into a unified formulation, we extend the state-extraction and element-transformation principle of *LENSNorm* to *Softmax*.

Given an input vector $\mathbf{x} = [x_1, \ldots, x_N]$ as one row in $\mathbf{Q}\mathbf{K}^\mathsf{T}/\sqrt{d}$, the i-th output of *Softmax* is $\mathrm{Softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{N}\exp(x_j)}$. If we define two functions as $\lambda(\mathbf{x}) = \sum_{j=1}^{N}\exp(x_j)$ and $\mu(\mathbf{x}) = \log\lambda(\mathbf{x})$, $\mathrm{Softmax}(\mathbf{x})_i = \exp\left(x_i - \mu(\mathbf{x})\right) = g\left(x_i, \mu(\mathbf{x})\right)$. Inspired by the $g\left(x_i, \mu(\mathbf{x})\right)$ format of *Softmax*, we define *LENSMax* as one row-state function $r(\mathbf{x})$ and one element-mapping function $e(x_i, r(\mathbf{x}))$. We then adopt two tiny MLPs to parameterize these two functions in Equation 2. The detailed mathematical derivation process can be found in Appendix A.2.

$$\mathrm{LENSMax}(\mathbf{x})_i = \frac{\mathrm{MLP}_2\left(x_i, \mathrm{MLP}_1(\mathbf{x})\right)}{2^{\lfloor \log_2 N \rfloor}} \tag{2}$$

The first MLP has two FC layers: The first FC layer maps one row in $\mathbf{Q}\mathbf{K}^\mathsf{T}/\sqrt{d}$ to 32 neurons, followed by a *ReLU* activation function. We retain the standard scaling factor $\sqrt{d}$, where d denotes the head dimension. Since $1/\sqrt{d}$ is a fixed constant determined by the model architecture, it does not require an input-dependent division and can be implemented using constant scaling on hardware. The second FC layer has only one neuron to generate the row state.

The second MLP also has two FC layers: The first FC layer maps one score and its corresponding row state from the first MLP to 16 neurons. The second FC layer has only one neuron to compute the new element result, followed by a *Sigmoid* function. Then, the *Sigmoid* result is divided by $2^{\lfloor \log_2 N \rfloor}$, where N denotes the number of valid elements in the row. Therefore, the sum of each *LENSMax* output row is lower than 2. We use $2^{\lfloor \log_2 N \rfloor}$, rather than N, also for hardware efficiency, which can also be easily converted to bitwise shifting in fixed-point format or to an exponent adjustment in floating-point format.

Compared with the previous representative alternatives to *Softmax*, *LENSMax* is hardware-friendly and also moves more computations to GEMMs. Although *ReLU Attention* has a simpler computational structure than *LENSMax*, its authors report no clear best choice among several point-wise functions, including *ReLU*, *ReLU*^{2}, *GELU* (Hendrycks & Gimpel, 2016), and *Softplus*, when combined with sequence-length scaling. They select *ReLU* mainly because of its higher computational speed, while leaving open why the L^{-1} scaling improves performance and whether a better activation function exists. These observations motivate us to learn the row-state and element mappings instead of manually selecting a fixed function.

3 Experiments

To analyze the network performance with *LENSNorm* and *LENSMax* after replacing *LayerNorm* and *Softmax*, we evaluated them in vision classification and causal language tasks. All training experiments were conducted using a single *NVIDIA* A40 GPU. For the vision task, we implemented a self-designed single-head *Vision Transformer* (ViT) (Dosovitskiy et al., 2021) on *CIFAR-10* and *CIFAR-100* datasets (Krizhevsky, 2009) to enable rapid assessment of the feasibility of replacing *LayerNorm* and *Softmax* with LENS. We then extended the evaluation to *Data-Efficient Image Transformer* (DeiT) (Touvron et al., 2021) with *ImageNet-1K* (Deng et al., 2009) classification. Moreover, we evaluated LENS using *Pythia*-14M/31M/70M (Biderman et al., 2023) to test whether this architecture can be generalized to causal language models, rather than being limited to vision tasks.

3.1 CIFAR-10 and CIFAR-100 using Single-Head ViT

We first conducted proof-of-concept experiments on *CIFAR-10* and *CIFAR-100* using a lightweight single-head *ViT*. We input 65 tokens, each with a hidden dimension of 128, to the model. 64 tokens are generated from the 32×32-pixel RGB images with a 4×4 convolution kernel and a stride of 4. One additional *Classification* (CLS) token is adopted, and learnable absolute positional embeddings are added to all tokens. The *Transformer* backbone of the single-head *ViT* consists of three *Pre-Layer Normalization* (Pre-LN) blocks. Each block uses single-head scaled dot-product self-attention and a *Feed Forward Network* (FFN) with dimensions 128, 512, and 128. The FFN applies the *GELU* activation function. Both the attention and FFN sublayers use residual connections. The output projection and FFN layers use a dropout rate of 0.1, while the attention weights use no dropout. A

Table 3: Experimental Results on *CIFAR-10* and *CIFAR-100* using a Single-Head ViT with *LayerNorm*, *Softmax*, *LENSNorm* and *LENSMax*.

Dataset	Training Path	Epochs	Peak LR	Norm.	Attn.	Top-1 Acc. (%)	Δ Acc. (pp)
CIFAR-10	From Scratch	400	1×10^{-3}	LayerNorm	Softmax	80.33 ± 0.28	0.00 (Baseline)
				LENSNorm	Softmax	80.18 ± 0.80	-0.15
				LayerNorm	LENSMax	78.82 ± 0.43	-1.51
				LENSNorm	LENSMax	79.97 ± 0.79	-0.36
	Only-LENS	50	1×10^{-3}	LENSNorm	LENSMax	68.26 ± 3.38	-12.07
	Scratch→LENS	50	5×10^{-4}	LENSNorm	Softmax	81.41 ± 0.17	$+1.08$
				LayerNorm	LENSMax	80.06 ± 0.38	-0.27
				LENSNorm	LENSMax	81.00 ± 0.19	$+0.67$
CIFAR-100	From Scratch	400	1×10^{-3}	LayerNorm	Softmax	51.22 ± 0.55	0.00 (Baseline)
				LENSNorm	Softmax	52.80 ± 1.28	$+1.58$
				LayerNorm	LENSMax	50.63 ± 0.71	-0.59
				LENSNorm	LENSMax	53.03 ± 1.44	$+1.81$
	Only-LENS	50	1×10^{-3}	LENSNorm	LENSMax	38.50 ± 1.70	-12.72
	Scratch→LENS	50	5×10^{-4}	LENSNorm	Softmax	52.14 ± 0.83	$+0.92$
				LayerNorm	LENSMax	51.01 ± 0.72	-0.21
				LENSNorm	LENSMax	52.05 ± 0.60	$+0.83$

Note: *Norm.* and *Attn.* denote the normalization and attention methods, respectively. *From Scratch* denotes training from scratch. *Scratch→LENS* denotes retraining the indicated LENS configuration from the corresponding *From Scratch* trained model. *Only-LENS* denotes initialization from the corresponding *From Scratch* trained model, with all original model parameters frozen and only the LENSNorm and LENSMax parameters trained. The *From Scratch* models using both *LayerNorm* and *Softmax* serve as the baselines.

dropout rate of 0.1 is also applied after adding the positional embeddings. After a final normalization layer, the CLS representation is passed to a linear classifier with 10 or 100 outputs.

As shown in Table 3, we trained the baseline single-head ViTs for *CIFAR-10* and *CIFAR-100* from scratch for 400 epochs using *LayerNorm* and *Softmax*. We then replaced *LayerNorm*, *Softmax*, or both with the corresponding LENS modules. We evaluated these configurations either by training from scratch or by initializing them from the corresponding trained baseline models and retraining all model parameters for 50 epochs. Furthermore, in the *Only-LENS* setting, we froze all loaded baseline parameters and trained only *LENSNorm* and *LENSMax* MLPs for 50 epochs. All models were trained with a batch size of 256 using the *AdamW* (Loshchilov & Hutter, 2019) optimizer with a weight decay of 0.05 and a cosine annealing learning rate scheduler. Results are reported as mean $\pm$ standard deviation over five random seeds, $\{42, 886, 1024, 1991, 2333\}$. We applied *RandomCrop* with a padding of 4 and *RandomHorizontalFlip* for data augmentation.

Therefore, based on the experimental results, we find that LENS can replace both *LayerNorm* and *Softmax* and can be trained from scratch. When both *LayerNorm* and *Softmax* are replaced, the *From Scratch* models achieve accuracy close to the baseline on *CIFAR-10* and exceed it on *CIFAR-100* after 400 epochs. Initializing from the trained baselines enables faster recovery in only 50 epochs and exceeds the baselines by 0.67 and 0.83 percentage points, respectively. *LENSNorm* generally performs better than *LENSMax*. When we only replace *LayerNorm* with *LENSNorm*, the replaced model exceeds the baseline on both datasets after retraining. In contrast to *LENSNorm*, only replacing *Softmax* with *LENSMax* causes small accuracy losses. Moreover, if we freeze the loaded model parameters and train only the LENS MLPs, accuracy loss is substantially larger than in other settings. This suggests that LENS does not simply approximate *LayerNorm* and *Softmax*, but jointly adapts with the original model parameters during retraining.

3.2 ImageNet using DeiT-Tiny and DeiT-Small

To evaluate the performance of LENS on larger ViT models and larger datasets, we further conducted experiments of LENS on DeiT-Tiny and DeiT-Small using ImageNet-1k. As shown in Table 4, we downloaded the official checkpoints of DeiT-Tiny and DeiT-Small as baselines, and then replaced all *LayerNorm* and *Softmax* with LENS. We retrained the LENS-replaced models for 50 epochs using cosine annealing or exponential-decay learning-rate schedules with or without *Knowledge Distillation* (KD). For DeiT-Tiny, we additionally explored training from scratch after replacing

Table 4: Experimental Results on *ImageNet* using DeiT-Tiny and DeiT-Small with *LayerNorm*, *Softmax*, *LENSNorm* and *LENSMax*.

Model	Training Path	LR Schedule	Knowledge Distillation	Top-1 Acc. (%)	Top-1 Δ Acc. (pp)	Top-5 Acc. (%)	Top-5 Δ Acc. (pp)
DeiT-Tiny (Base.)	Official	-	-	72.21	0.00	91.10	0.00
DeiT-Tiny+LENS	From Scratch	Cos.	No	51.67	−20.54	75.29	−15.81
	Official→LENS	Cos.	No	71.94	−0.27	90.53	−0.57
	Official→LENS	Exp.	No	72.44 ± 0.03	+0.23	90.95 ± 0.05	−0.15
	Official→LENS	Exp.	Yes	73.66 ± 0.26	+1.45	91.91 ± 0.11	+0.81
DeiT-Small (Base.)	Official	-	-	79.87	0.00	95.06	0.00
DeiT-Small+LENS	Official→LENS	Exp.	No	74.51	−5.36	92.07	−2.99
	Official→LENS	Exp.	Yes	78.30	−1.57	94.12	−0.94

Note: *Official* denotes the downloaded official pretrained DeiT model using *LayerNorm* and *Softmax*. *From Scratch* and *Official→LENS* denote LENS models trained from scratch and initialized from the downloaded official pretrained model, respectively. All LENS models use both *LENSNorm* and *LENSMax*. *Base.* denotes the baseline. Results using DeiT-Tiny with the exponential-decay learning-rate schedule are reported as mean ± standard deviation over three seeds; all other results are from single runs. *Cos.* and *Exp.* denote the cosine annealing and exponential decay learning-rate schedules.

LayerNorm and *Softmax* with the LENS modules. The training results of DeiT-Tiny models using exponential-decay learning-rate schedules are reported as the mean ± standard deviation across three random seeds, $\{42, 1024, 2333\}$. All experiments were trained with a batch size of 256 and a peak learning rate of 7.5×10^{-5} using the *AdamW* optimizer with a weight decay of 0.05. The exponential decay rates γ in DeiT-Tiny and DeiT-Small are 0.90 and 0.95, respectively. For the KD setting used in related experiments, we applied KD only to the output logits using the corresponding official checkpoint as the teacher, with a temperature $T = 2$ and a weight $\alpha = 0.5$. We followed the standard ImageNet data preprocessing and augmentation settings of the official DeiT implementation.

According to the experimental results in Table 4, under the same 50-epoch training budget, training the LENS-replaced DeiT-Tiny from scratch leads to a large loss in accuracy, whereas initializing from the official pretrained model significantly improves accuracy recovery. Compared with cosine annealing, the exponential-decay learning-rate schedule performed better in our DeiT-Tiny experiments. With exponential decay, both DeiT-Tiny experiments with and without KD exceed the Top-1 accuracy of the baseline. The KD setting exceeds the baseline by 1.45 and 0.81 percentage points in Top-1 and Top-5 accuracy, respectively. For DeiT-Small, after applying KD and the exponential-decay learning-rate schedule, the LENS-replaced model also recovers most of the baseline accuracy, with losses of only 1.57 and 0.94 percentage points in Top-1 and Top-5 accuracy, respectively.

3.3 Language Modeling using Pythia-14M/31M/70M

To evaluate whether LENS can be adopted in causal language modeling tasks rather than only vision models, we conducted proof-of-concept experiments using the *Pythia*-14M/31M/70M models (Biderman et al., 2023). Table 5 reports the validation and generalization results of the official *Pythia*-14M checkpoint trained on *The Pile* (Gao et al., 2020), its LENS-replaced version retrained on *The Pile Subset*, and the *Pythia*-14M/31M/70M models trained by us on *FineWeb-Edu-3B* (Penedo et al., 2024). All models were evaluated on *WikiText* (Merity et al., 2017) and *LAMBADA* (Paperno et al., 2016) using the *lm-evaluation-harness* toolkit (Gao et al., 2024). All models trained on *FineWeb-Edu-3B* used a batch size of 16, the *AdamW* optimizer with a weight decay of 0.01, and a cosine-annealing learning-rate schedule. The maximum sequence length was fixed at 512. For causal attention, future positions were masked before *LENSMax*, and their output weights were set to zero, while N was set to the number of valid elements in each row.

We first replaced all *LayerNorm* and *Softmax* in the official *Pythia*-14M checkpoint with LENS and retrained the model for two epochs on a subset of *The Pile*, containing 624000 training samples and 5000 validation samples. Immediately after replacement, the model performance dropped to near-random levels. As shown by the validation trajectories in Figure 3 of Appendix A.3, its performance was gradually recovered during retraining. This indicates that LENS replacement is not conventional fine-tuning, but joint retraining initialized from pretrained model parameters. Although the retrained model largely recovered its validation performance on *The Pile Subset*, its performance collapsed

Table 5: Validation and Generalization Results of Pythia-14M/31M/70M using *LayerNorm*, *Softmax*, *LENSNorm*, and *LENSMax*.

Validation Results

Pythia Model	Training Dataset	Training Path	Epochs	LR Schedule	PPL ↓	Δ PPL	Next-Token Acc. (%) ↑	Next-Token Δ Acc. (pp)
14M (Base.)	The Pile	Official	-	-	98.01	0.00	28.95	0.00
14M+LENS	The Pile Subset	Official→LENS	2	Step.	92.54	−5.47	27.45	−1.50
14M (Base.)		From Scratch			56.67	0.00	31.56	0.00
14M+LENS	FineWeb-Edu-3B	From Scratch	1	Cos.	64.70	+8.03	30.13	−1.43
14M+LENS		Scratch→LENS			57.27 ± 0.26	+0.60	31.35 ± 0.07	−0.21
31M (Base.)	FineWeb-Edu-3B	From Scratch	1	Cos.	39.89	0.00	34.72	0.00
31M+LENS		Scratch→LENS			40.77	+0.88	34.38	−0.34
70M (Base.)	FineWeb-Edu-3B	From Scratch	1	Cos.	30.45	0.00	37.36	0.00
70M+LENS		Scratch→LENS			31.00	+0.55	37.03	−0.33

Generalization Evaluation Results

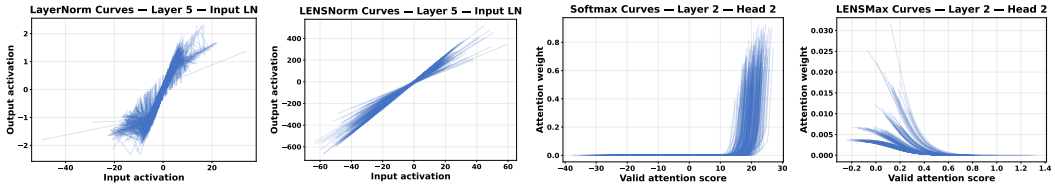
Pythia Model	Training Dataset	Training Path	WikiText PPL ↓	WikiText Δ PPL	WikiText BPB ↓	WikiText Δ BPB
14M (Base.)	The Pile	Official	154.28	0.00	1.359	0.000
14M+LENS	The Pile Subset	Official→LENS	632.81	+478.53	1.740	+0.381
14M (Base.)		From Scratch	158.71	0.00	1.367	0.000
14M+LENS	FineWeb-Edu-3B	From Scratch	184.55	+25.84	1.408	+0.041
14M+LENS		Scratch→LENS	160.14 ± 0.84	+1.43	1.370 ± 0.002	+0.003
31M (Base.)	FineWeb-Edu-3B	From Scratch	103.92	0.00	1.253	0.00
31M+LENS		Scratch→LENS	107.29	+3.37	1.261	+0.008
70M (Base.)	FineWeb-Edu-3B	From Scratch	74.14	0.00	1.162	0.00
70M+LENS		Scratch→LENS	76.33	+2.19	1.170	+0.008

Pythia Model	Training Dataset	Training Path	LAMBADA PPL ↓	LAMBADA Δ PPL	LAMBADA Acc. (%) ↑	LAMBADA Δ Acc. (pp)
14M (Base.)	The Pile	Official	914.80 ± 208.72	0.00	12.00 ± 2.30	0.00
14M+LENS	The Pile Subset	Official→LENS	27028.53 ± 6481.85	+26113.73	0.00 ± 0.00	−12.00
14M (Base.)		From Scratch	3426.19 ± 179.30	0.00	8.64 ± 0.39	0.00
14M+LENS	FineWeb-Edu-3B	From Scratch	5484.52 ± 272.83	+2058.33	4.70 ± 0.29	−3.94
14M+LENS		Scratch→LENS	3587.40 ± 349.60	+161.21	6.99 ± 0.58	−1.65
31M (Base.)	FineWeb-Edu-3B	From Scratch	830.77 ± 42.35	0.00	14.94 ± 0.50	0.00
31M+LENS		Scratch→LENS	875.69 ± 43.99	+44.92	12.83 ± 0.47	−2.11
70M (Base.)	FineWeb-Edu-3B	From Scratch	346.34 ± 17.14	0.00	18.32 ± 0.54	0.00
70M+LENS		Scratch→LENS	304.73 ± 14.80	−41.61	18.09 ± 0.54	−0.23

Note: *Official* is the released pretrained Pythia-14M checkpoint. *Official→LENS* and *Scratch→LENS* initialize LENS from this checkpoint and the corresponding baseline we trained. *From Scratch* uses a random initialization. All LENS models contain both *LENSNorm* and *LENSMax*. *Base.* uses *LayerNorm* and *Softmax*. The *Pythia*-14M *Scratch→LENS* results on *FineWeb-Edu-3B* are mean ± standard deviation over three training seeds; all others are single runs. For the single-run *LAMBADA* evaluations, ± indicates the standard error reported by the evaluation harness. *Cos./Step.* indicate cosine annealing/two-stage step decay. *PPL/BPB* indicate perplexity/bits per byte.

on *WikiText* and *LAMBADA*. This suggests that retraining on the limited subset adapted the model to that subset but could not recover the broad language knowledge learned from the complete *Pile* dataset. This failure motivated us to train the baseline and LENS models on the same *FineWeb-Edu-3B* dataset.

For the *FineWeb-Edu-3B* experiments, each training or retraining stage used one epoch. The learning rate was linearly warmed up over the first 1% of training steps to 1×10^{-3} and then cosine-annealed to 1×10^{-4}. Under the same one-epoch training budget, the Pythia-14M LENS model trained from random initialization performs worse than the standard *LayerNorm*/*Softmax* baseline on validation, *WikiText*, and *LAMBADA*, indicating that LENS is more difficult to train from scratch. However, when initialized from our trained baseline and retrained on the same dataset, the LENS models recover most of the baseline performance across Pythia-14M/31M/70M. Their validation next-token accuracy differs from the baselines by only -0.21, -0.34, and -0.33 percentage points, respectively. Their generalization performance on *WikiText* and *LAMBADA* also recovers substantially, with the 70M LENS model even achieving lower LAMBADA PPL than its baseline. These results suggest that LENS is applicable to causal language models, but its optimization is more difficult and benefits substantially from initialization with trained model parameters.



(a) LayerNorm (left) and LENSNorm (right) in Pythia-14M (b) Softmax (left) and LENSMax (right) in Pythia-14M

Figure 1: Representative functional curves of LayerNorm and Softmax compared with LENS.

4 Discussion

We discuss the features and limitations of LENS through the following questions:

Q1. What is the goal of LENS?

The goal of LENS is not to achieve state-of-the-art accuracy or immediate speedup, but to eliminate *LayerNorm* and *Softmax* as distinct computational primitives, making Transformer accelerators increasingly GEMM-centric. From a hardware perspective, this allows future accelerators to concentrate their computational resources around highly optimized GEMM/systolic-array fabrics, while minimizing the need for dedicated datapaths for normalization and *Softmax*.

Q2. What are the strengths of LENS?

LENS is a GEMM-centric, hardware-friendly architecture that replaces *LayerNorm* and *Softmax* with tiny MLPs. Therefore, most computations in LENS can be mapped to GEMMs, while the remaining *ReLU*, *Sigmoid*, and power-of-two scaling operations can be efficiently supported using well-established hardware implementation techniques. Our experiments with LENS suggest that hand-designed functions, including *LayerNorm*, *Softmax*, and their alternatives, may not be the only choices for *Transformers*.

Q3. Does LENS simply approximate *LayerNorm* and *Softmax*?

No. As shown in Figure 1, we plot representative functional curves of *LayerNorm* and *Softmax* and compare them with those produced by the corresponding LENS modules after retraining. For *LayerNorm* and *LENSNorm*, we evaluated 256 tokens to draw the functional curves. For *Softmax* and *LENSMax*, we collected 256 rows of attention scores. The results show that *LENSNorm* and *LENSMax* learn substantially different functional mappings from the original operators. In Appendix A.4, we further compare the relative ℓ_2 weight changes between the baseline and retrained LENS models. The results show substantial changes in some model parameters during retraining. Together with the large accuracy loss in the *Only-LENS* setting in the single-head ViT experiments, these observations suggest that LENS does not merely approximate the original operators but instead jointly adapts with the model parameters.

Q4. What are the limitations of LENS?

LENS is not a zero-shot replacement of *LayerNorm* and *Softmax*. Retraining is necessary and important for LENS. Some experimental results are not lossless compared with baselines. Moreover, the current *LENSMax* requires a predefined maximum input length. Supporting length-agnostic inputs, evaluating larger models, and conducting direct hardware evaluation remain our future work. The current LENS research is a proof of concept showing that tiny learnable MLPs can serve as alternatives to the hand-designed *LayerNorm* and *Softmax*.

5 Conclusion

In this work, we presented LENS, a GEMM-centric architecture that replaces all *LayerNorm* and attention *Softmax* operations in *Transformers* with tiny learnable MLPs. Experiments on CIFAR-10/100, ImageNet-1K with DeiT, and causal language modeling with Pythia show that LENS can recover most baseline performance and even exceed it in several vision settings. Our analyses further show that LENS does not simply approximate the original operators but jointly adapts with the model parameters during retraining. These results provide a proof of concept that hand-designed *LayerNorm* and *Softmax* functions are not the only choices for *Transformers* and motivate future research on scalable training and hardware implementation.

AI USE STATEMENT

The research question, core mechanism, and principal methodological design of LENS were conceived and developed by the authors independently of generative AI. We used the web-based *ChatGPT* interface as an interactive research assistant. No autonomous AI agent was used to design, execute, or manage the research workflow. The use of *ChatGPT* was author-driven: the authors first formulated the intended research ideas, experimental designs, and implementation approaches, and then selectively considered suggestions provided by *ChatGPT*. *ChatGPT* was used to assist with boilerplate code for loading pretrained backbone models and preparing datasets, to discuss implementation and debugging issues, to review intermediate experimental diagnostics, and to improve the grammar, clarity, and readability of parts of the manuscript. AI-generated code suggestions were generally used as references rather than automatically incorporated into the implementation. The authors independently wrote or modified the final code based on their own design decisions after considering these suggestions. Any AI-assisted code used in the experiments was manually reviewed, tested, and verified by the authors before use. All research hypotheses, methodological contributions, experimental decisions, interpretation of the final results, and scientific claims were determined and verified by the authors. The authors take full responsibility for the final code, experiments, and manuscript.

REPRODUCIBILITY STATEMENT

The complete source code for all *CIFAR-10* and *CIFAR-100* experiments, including the full implementation of LENS, is provided in the supplementary material. The code for the DeiT and Pythia experiments, together with an installable LENS package, will be publicly released on GitHub upon acceptance.

REFERENCES

Milton Abramowitz and Irene A. Stegun (eds.). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, volume 55 of *Applied Mathematics Series*. National Bureau of Standards, Washington, DC, 1964.

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

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: a suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Wenrui Cai, Defa Zhu, Siyuan Qiao, Qingjie Liu, and Qiyang Min. SeeDNorm: Self-rescaled dynamic normalization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=dFL7XExDvv.

Hongzheng Chen, Jiahao Zhang, Yixiao Du, Shaojie Xiang, Zichao Yue, Niansong Zhang, Yaohui Cai, and Zhiru Zhang. Understanding the potential of fpga-based spatial acceleration for large language model inference. *ACM Trans. Reconfigurable Technol. Syst.*, 18(1), December 2024. ISSN 1936-7406. doi: 10.1145/3656177. URL https://doi.org/10.1145/3656177.

Mingzhi Chen, Taiming Lu, Jiachen Zhu, Mingjie Sun, and Zhuang Liu. Stronger normalization-free transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 27418–27428, June 2026.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of*

the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

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, 2020. URL https://arxiv.org/abs/2101.00027.

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. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016. URL https://arxiv.org/abs/1606.08415.

Soroush Abbasi Koohpayegani and Hamed Pirsiavash. SimA: Simple softmax-free attention for vision transformers. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2595–2605, 2024. doi: 10.1109/WACV57701.2024.00259.

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

Zhikai Li and Qingyi Gu. I-ViT: Integer-only quantization for efficient vision transformer inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17065–17075, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Jiachen Lu, Jinghan Yao, Junge Zhang, Xiatian Zhu, Hang Xu, Weiguo Gao, Chunjing Xu, Tao Xiang, and Li Zhang. SOFT: Softmax-free transformer with linear complexity. In *NeurIPS*, 2021.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

NVIDIA Corporation. NVIDIA's Next Generation CUDA Compute Architecture: Fermi. White paper, NVIDIA Corporation, 2009. URL https://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf. Version 1.1.

NVIDIA Corporation. NVIDIA RTX Blackwell GPU Architecture: Built for Neural Rendering. White paper, NVIDIA Corporation, 2025. URL https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf. Version 1.1.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Katrin Erk and Noah A. Smith (eds.),

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: decanting the web for the finest text data at scale. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10347–10357. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/touvron21a.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Mitchell Wortsman, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Replacing softmax with relu in vision transformers, 2023. URL https://arxiv.org/abs/2309.08586.

Biao Zhang and Rico Sennrich. *Root mean square layer normalization*. Curran Associates Inc., Red Hook, NY, USA, 2019.

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

A Appendix

A.1 Unified Formulation of LayerNorm and Its Alternatives for LENSNorm

A.1.1 LayerNorm

$$\begin{aligned}\mathrm{LayerNorm}(\mathbf{x}) &= \frac{\mathbf{x}-\mathrm{Mean}(\mathbf{x})}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}}\odot\boldsymbol{\gamma}+\boldsymbol{\beta}\\ &= \frac{\boldsymbol{\gamma}}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}}\odot\mathbf{x}+\boldsymbol{\beta}-\frac{\mathrm{Mean}(\mathbf{x})\boldsymbol{\gamma}}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}}\\ &= \mathbf{a}(\mathbf{x})\odot\mathbf{x}+\mathbf{b}(\mathbf{x}),\end{aligned} \tag{3}$$

$$\mathbf{a}(\mathbf{x}) = \frac{\boldsymbol{\gamma}}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}},$$

$$\mathbf{b}(\mathbf{x}) = \frac{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}\,\boldsymbol{\beta}-\mathrm{Mean}(\mathbf{x})\,\boldsymbol{\gamma}}{\sqrt{\mathrm{Var}(\mathbf{x})+\epsilon}}.$$

A.1.2 RMSNORM

$$\begin{aligned}\mathrm{RMSNorm}(\mathbf{x}) &= \frac{\mathbf{x}}{\sqrt{\mathrm{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}}\odot\boldsymbol{\gamma}\\ &= \frac{\boldsymbol{\gamma}}{\sqrt{\mathrm{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}}\odot\mathbf{x}\\ &= \mathbf{a}(\mathbf{x})\odot\mathbf{x}+\mathbf{b}(\mathbf{x}),\\ \mathbf{a}(\mathbf{x}) &= \frac{\boldsymbol{\gamma}}{\sqrt{\mathrm{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}},\\ \mathbf{b}(\mathbf{x}) &= \mathbf{0}.\end{aligned} \tag{4}$$

A.1.3 DYT

$$\begin{aligned}\mathrm{DyT}(\mathbf{x}) &= \tanh(\alpha\mathbf{x})\odot\boldsymbol{\gamma}+\boldsymbol{\beta}\\ &\approx \left(\alpha\mathbf{x}-\frac{\alpha^3}{3}\mathbf{x}^{\odot 3}\right)\odot\boldsymbol{\gamma}+\boldsymbol{\beta}\\ &= \left(\alpha\boldsymbol{\gamma}-\frac{\alpha^3}{3}\boldsymbol{\gamma}\odot\mathbf{x}^{\odot 2}\right)\odot\mathbf{x}+\boldsymbol{\beta}\\ &= \mathbf{a}(\mathbf{x})\odot\mathbf{x}+\mathbf{b}(\mathbf{x}),\\ \mathbf{a}(\mathbf{x}) &= \alpha\boldsymbol{\gamma}-\frac{\alpha^3}{3}\boldsymbol{\gamma}\odot\mathbf{x}^{\odot 2},\\ \mathbf{b}(\mathbf{x}) &= \boldsymbol{\beta}.\end{aligned} \tag{5}$$

A.1.4 DERF

The error function is approximated as (Abramowitz & Stegun, 1964)

$$\mathrm{erf}(\mathbf{x}) \approx \mathrm{sign}(\mathbf{x})\left(1-\frac{1}{\left(1+a_1|\mathbf{x}|+a_2|\mathbf{x}|^2+a_3|\mathbf{x}|^3+a_4|\mathbf{x}|^4\right)^4}\right). \tag{6}$$

Accordingly, $\mathcal{R}(z)$ in Table 2 denotes

$$\mathcal{R}(z) = \frac{1-\left(1+a_1|z|+a_2|z|^2+a_3|z|^3+a_4|z|^4\right)^{-4}}{|z|}. \tag{7}$$

$$\begin{aligned}\mathrm{Derf}(\mathbf{x}) &= \mathrm{erf}(\alpha\mathbf{x}+s)\odot\boldsymbol{\gamma}+\boldsymbol{\beta}\\ &\approx (\alpha\mathbf{x}+s)\odot\mathcal{R}(\alpha\mathbf{x}+s)\odot\boldsymbol{\gamma}+\boldsymbol{\beta}\\ &= [\alpha\boldsymbol{\gamma}\odot\mathcal{R}(\alpha\mathbf{x}+s)]\odot\mathbf{x}+s\boldsymbol{\gamma}\odot\mathcal{R}(\alpha\mathbf{x}+s)+\boldsymbol{\beta}\\ &= \mathbf{a}(\mathbf{x})\odot\mathbf{x}+\mathbf{b}(\mathbf{x}),\\ \mathbf{a}(\mathbf{x}) &= \alpha\boldsymbol{\gamma}\odot\mathcal{R}(\alpha\mathbf{x}+s),\\ \mathbf{b}(\mathbf{x}) &= s\boldsymbol{\gamma}\odot\mathcal{R}(\alpha\mathbf{x}+s)+\boldsymbol{\beta}.\end{aligned} \tag{8}$$

For $z=0$, $\mathcal{R}(z)$ is defined by continuity, with $\mathcal{R}(0)=4a_1$. And for a vector input, $\mathcal{R}(\cdot)$ is applied element-wise.

A.1.5 SEEDNORM

$$\begin{aligned}\text{SeeDNorm}(\mathbf{x}) &= \frac{\mathbf{x}}{\sqrt{\text{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}} \odot \left[\sigma\left(\mathbf{x}\boldsymbol{\beta}^{\mathrm{T}}\right)\boldsymbol{\alpha}+\boldsymbol{\gamma}\right] \\ &= \frac{\sigma\left(\mathbf{x}\boldsymbol{\beta}^{\mathrm{T}}\right)\boldsymbol{\alpha}+\boldsymbol{\gamma}}{\sqrt{\text{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}} \odot \mathbf{x} \\ &= \mathbf{a}(\mathbf{x})\odot\mathbf{x}+\mathbf{b}(\mathbf{x}), \\ \mathbf{a}(\mathbf{x}) &= \frac{\sigma\left(\mathbf{x}\boldsymbol{\beta}^{\mathrm{T}}\right)\boldsymbol{\alpha}+\boldsymbol{\gamma}}{\sqrt{\text{Mean}(\mathbf{x}\odot\mathbf{x})+\epsilon}}, \\ \mathbf{b}(\mathbf{x}) &= \mathbf{0}.\end{aligned} \tag{9}$$

A.2 CONVERSION FROM SOFTMAX TO LENSMAX

Given an input vector $\mathbf{x}=[x_1,\ldots,x_N]$ as one row in $\mathbf{Q}\mathbf{K}^{\mathsf{T}}/\sqrt{d}$, the i-th output of *Softmax* is

$$\text{Softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{N}\exp(x_j)}. \tag{10}$$

We define two functions as

$$\lambda(\mathbf{x}) = \sum_{j=1}^{N}\exp(x_j), \qquad \mu(\mathbf{x}) = \log\lambda(\mathbf{x}). \tag{11}$$

The *Softmax* output can then be reorganized as

$$\begin{aligned}\text{Softmax}(\mathbf{x})_i &= \frac{\exp(x_i)}{\lambda(\mathbf{x})} \\ &= \frac{\exp(x_i)}{\exp(\log\lambda(\mathbf{x}))} \\ &= \exp\left(x_i-\mu(\mathbf{x})\right).\end{aligned} \tag{12}$$

$$f(\mathbf{x}) = \mu(\mathbf{x}), \qquad g(x_i, f(\mathbf{x})) = \exp\left(x_i - f(\mathbf{x})\right). \tag{13}$$

$$\text{Softmax}(\mathbf{x})_i = g\left(x_i, f(\mathbf{x})\right). \tag{14}$$

Inspired by the $g\left(x_i, f(\mathbf{x})\right)$ format of *Softmax*, we define *LENSMax* as one row-state function $r(\mathbf{x})$ and one element-mapping function $e(x_i, r(\mathbf{x}))$.

To limit the output range of $e(x_i, r(\mathbf{x}))$, we apply a *Sigmoid* function to its result and divide the *Sigmoid* result by $2^{\lfloor\log_2 N\rfloor}$, where N denotes the number of valid elements in the row. Therefore, the sum of each *LENSMax* output row is lower than 2:

$$\sum_{i=1}^{N}\text{LENSMax}(\mathbf{x})_i < \frac{N}{2^{\lfloor\log_2 N\rfloor}} < 2. \tag{15}$$

We then adopt two tiny MLPs to parameterize the row-state and element-mapping functions. *Sigmoid* is the activation function of the output layer in the MLP of the element-mapping function:

$$\begin{aligned}r(\mathbf{x}) &= \text{MLP}_1(\mathbf{x}), \\ e(x_i, r(\mathbf{x})) &= \text{MLP}_2\left(x_i, r(\mathbf{x})\right), \\ \text{LENSMax}(\mathbf{x})_i &= \frac{\text{MLP}_2\left(x_i, \text{MLP}_1(\mathbf{x})\right)}{2^{\lfloor\log_2 N\rfloor}}.\end{aligned} \tag{16}$$

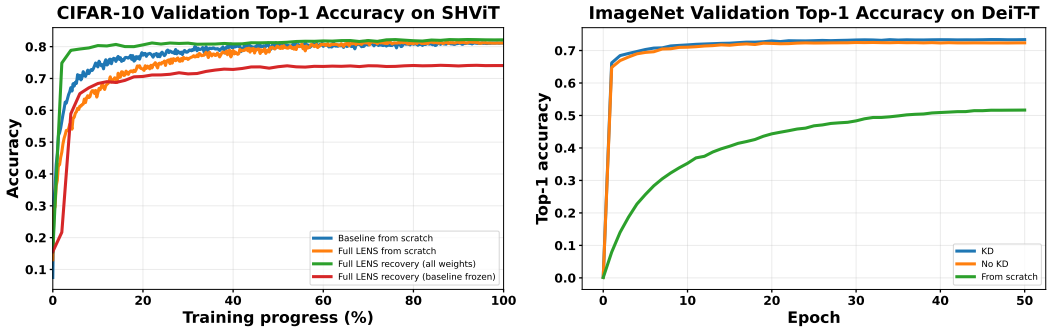


Figure 2: Validation trajectory of training on single-head ViT and DeiT-Tiny.

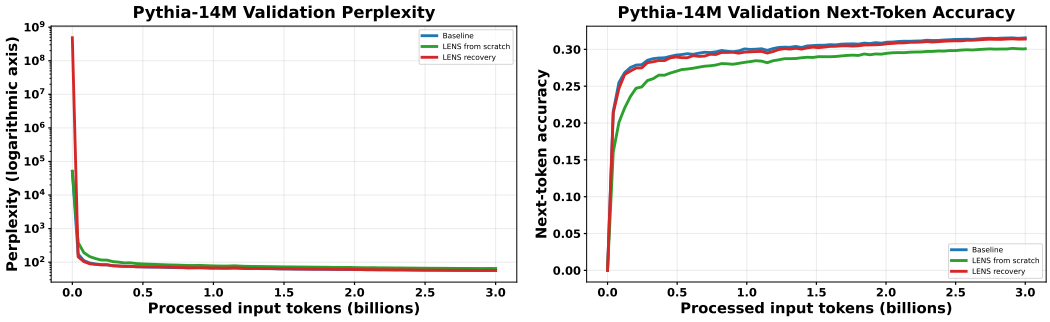


Figure 3: Validation trajectory of training on Pythia-14M.

A.3 Validation Accuracy Trajectory

We recorded the validation trajectory during training. The records show that the performance of all LENS models dropped to near-random levels immediately after replacement. The accuracy and language modeling performance of LENS models gradually recovered during retraining. This indicates that LENS replacement is not conventional fine-tuning, but joint retraining initialized from pretrained model parameters. Moreover, because replacing *LayerNorm* and *Softmax* with LENS causes a substantial architectural change and can produce extremely large gradients at the beginning of retraining, we clipped the global gradient norm to a maximum of 1.0.

A.4 Relative L2 of Inherited Weights after LENS Recovery Training

To quantify the changes in the inherited model parameters during LENS retraining, we compute the relative ℓ_2 weight change as $|\mathbf{W}\mathrm{LENS} - \mathbf{W}\mathrm{Base}|2/|\mathbf{W}\mathrm{Base}|_2$ for all shape-compatible parameters shared by the baseline and retrained LENS models.

For each shape-compatible parameter tensor, let $\mathbf{W}_{\mathrm{Base}}$ and $\mathbf{W}_{\mathrm{LENS}}$ denote its values in the baseline and retrained LENS models, respectively. We calculate its relative ℓ_2 change as

$$D_{\mathrm{rel}}(\mathbf{W}) = \frac{\|\mathrm{vec}(\mathbf{W}_{\mathrm{LENS}}) - \mathrm{vec}(\mathbf{W}_{\mathrm{Base}})\|_2}{\|\mathrm{vec}(\mathbf{W}_{\mathrm{Base}})\|_2 + \epsilon}, \tag{17}$$

where $\mathrm{vec}(\cdot)$ flattens the parameter tensor and ϵ is a small constant for numerical stability. Equivalently, for a tensor containing M elements,

$$D_{\mathrm{rel}}(\mathbf{W}) = \frac{\sqrt{\sum_{i=1}^{M} \left(W_{\mathrm{LENS},i} - W_{\mathrm{Base},i}\right)^2}}{\sqrt{\sum_{i=1}^{M} W_{\mathrm{Base},i}^2} + \epsilon}. \tag{18}$$

A value of zero indicates no weight change, while a value of one means that the ℓ_2 norm of the weight difference is equal to the ℓ_2 norm of the original parameter tensor.

In our experiment, newly introduced LENS parameters and replaced *LayerNorm* parameters are excluded.

The results show substantial and structured changes rather than uniformly small updates. For example, the relative changes of the first-block QKV and attention-output projection reach 0.303 and 0.486 on CIFAR-10. Similar changes are observed in DeiT, including approximately 0.542 for the final QKV layer of DeiT-Tiny and approximately 1.0 for the final FFN layer of DeiT-Small. Together with the poor *Only-LENS* results, these weight changes suggest that LENS jointly adapts with the inherited model parameters during retraining rather than merely approximating the replaced operators.

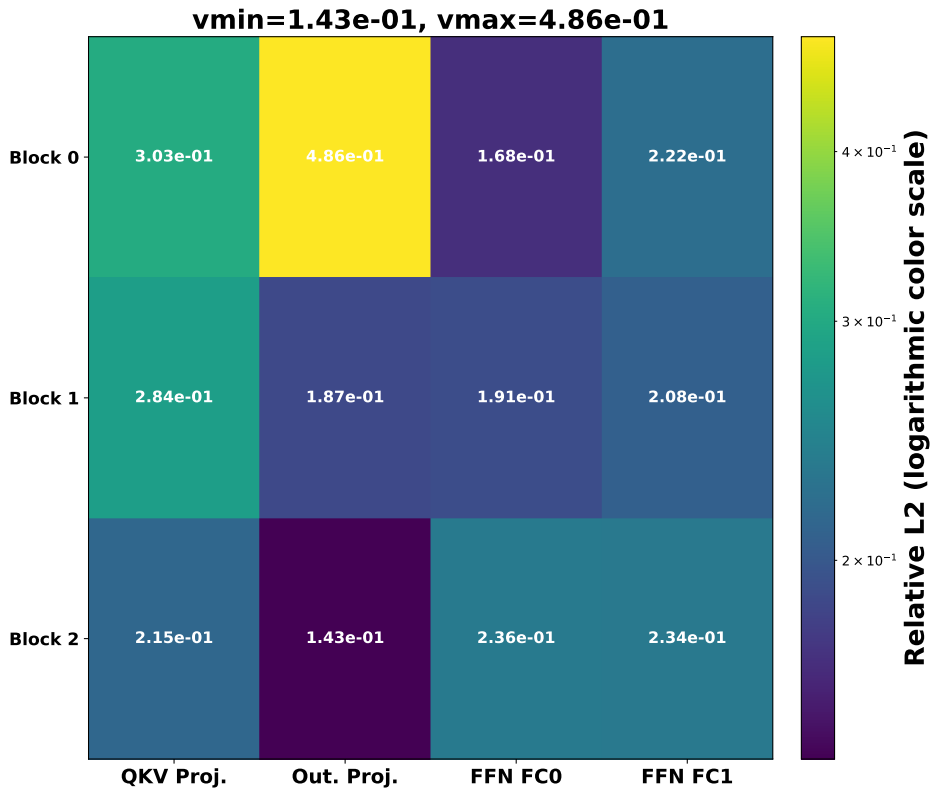


Figure 4: Relative ℓ_2 of inherited weights after LENS recovery training on single-head ViT using CIFAR-10.

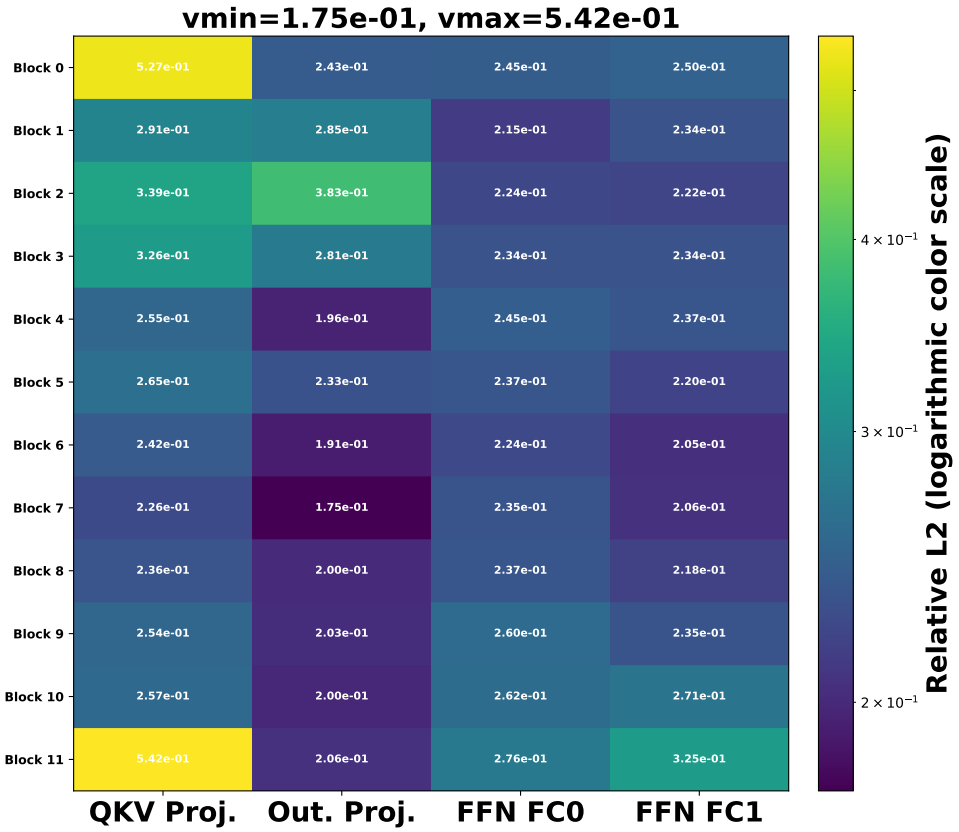


Figure 5: Relative ℓ_2 of inherited weights after LENS recovery training on DeiT-Tiny.

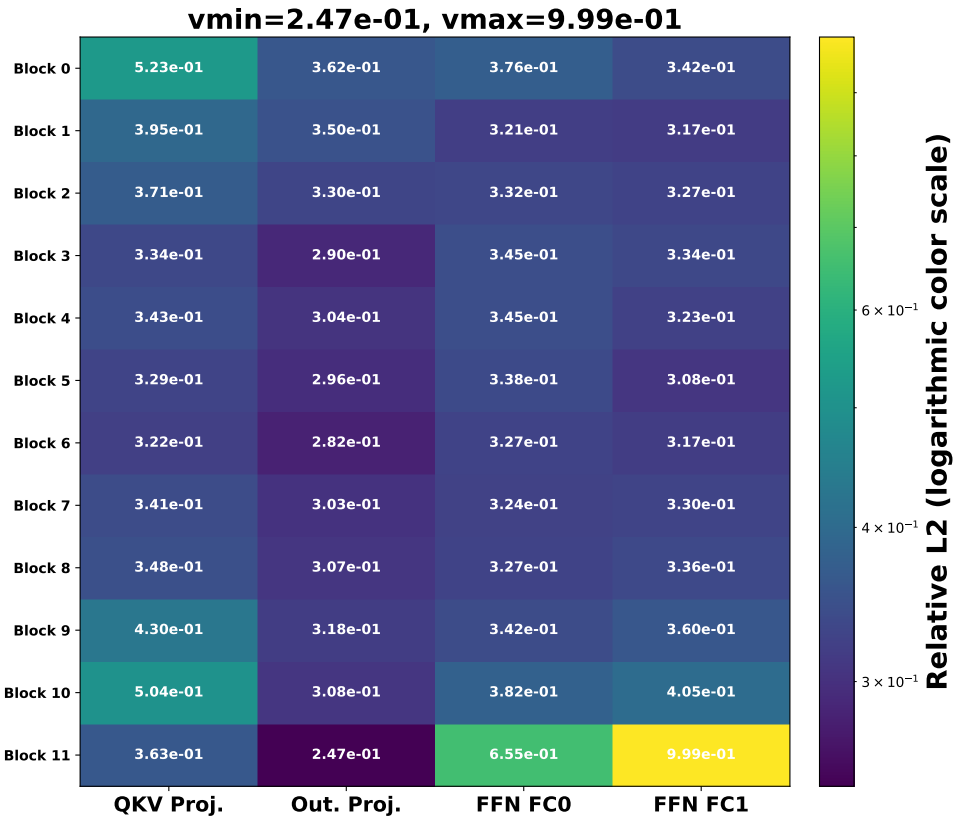


Figure 6: Relative ℓ_2 of inherited weights after LENS recovery training on DeiT-Small.

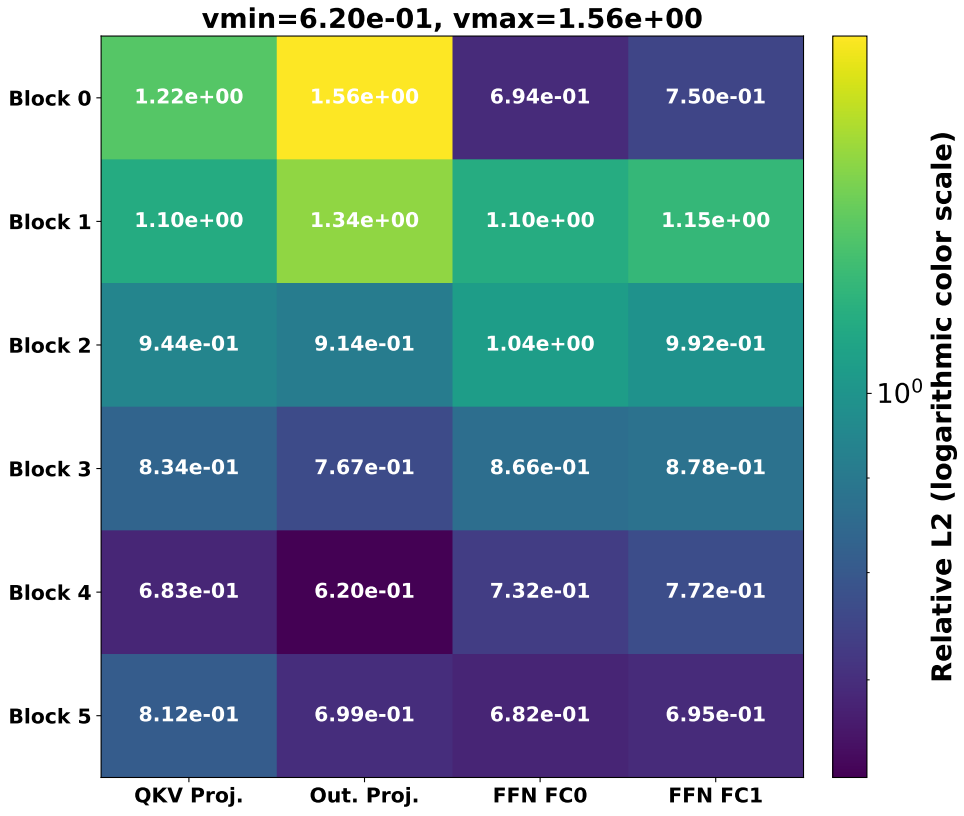


Figure 7: Relative ℓ_2 of inherited weights after LENS recovery training on Pythia-70M.