Bug or Feature[2]: Weight Drift, Activation Sparsity and Spikes

Egor Shvetsov * **Aleksandr Serkov** * **Shokorov Viacheslav** **Redko Dmitry** **Vladislav Goloshchapov** **Evgeny Burnaev**

Abstract

The design of modern neural architectures has converged through incremental empirical choices, yet the mechanisms governing their training dynamics remain only partially understood. We identify and analyze a negative weight drift induced by the interaction between standard losses and positively biased activation functions. We prove that under MSE or cross-entropy loss, the gradient with respect to positive pre-activations is non-negative in expectation at initialization, driving downstream weights toward negative values during early training. The drift is intrinsic to optimization rather than data, and persists across architectures and asymmetric activation functions (ReLU, GELU, SiLU). Coupled with ReLU, weight drift produces activation sparsity reaching nearly 90% in GPT-nano. We further analyze whether this natural sparsity is damaging and find that, on average, models sharply degrade above ∼70% activation sparsity. We then evaluate sparsity-inducing activation functions, which reveals activation spikes in intermediate transformer layers. Code is available at https://github.com/On-Point-RND/BugOrFeature.

1. Introduction

The design of modern neural architectures resembles an evolutionary process that converges toward stable paradigms. Incremental improvements are frequently discovered and adopted in an ad-hoc manner, yet we do not fully understand the intrinsic mechanics governing the models we employ. In this work, we identify and study a negative drift in weight distributions induced by the interaction between standard losses and positively asymmetric activation functions. This drift further negatively shifts the mean of intermediate representations. Passing these representations through the activation functions squashes them toward zero, which in turn reinforces the drift that produced them (Figure 1).

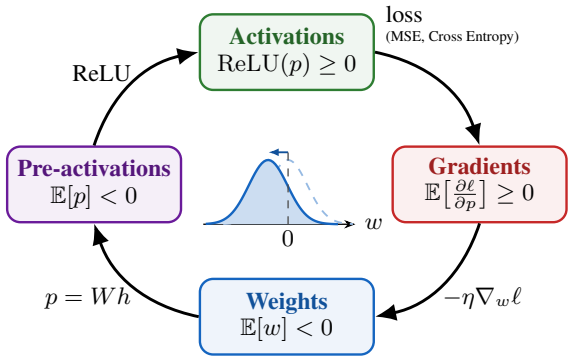


Figure 1. **Negative weight drift is a self-reinforcing loop.** Under MSE or cross-entropy loss, ReLU activations make the expected gradient with respect to pre-activations non-negative (§2). Gradient descent therefore shifts the weight distribution toward negative values (inset), which pushes pre-activations below zero and closes the loop. ReLU maps the shifted pre-activations to zero, producing activation sparsity (§4–§5).

Weight drift. We formally illustrate and empirically verify that, for positively biased activation functions combined with standard losses, gradient descent drives weights toward negative values during the early iterations of training. This shift is largest in the first iterations, when the loss is largest, and the resulting negative offset persists throughout training. The effect is intrinsic to the optimization rather than to the data, since the same drift appears when training on entirely random inputs, and it holds broadly across architectures.

Bug or Feature: emergent sparsity. In the absence of centering normalization [1] ReLU maps the negatively shifted pre-activations exactly to zero, inducing hard activation sparsity of nearly 90% in GPT-nano. With GELU and SiLU, the same drift pushes activations toward zero, so low-magnitude outputs dominate intermediate representations. In both regimes, this raises an immediate question of whether it is a **Bug or a Feature**. As a bug, uncontrolled sparsity or magnitude suppression risks degrading model performance by silencing large parts of the network. As a feature, an effect that emerges without any explicit regularization could

*Equal contribution. Correspondence to: Egor Shvetsov <geksut@gmail.com>.

Date October 14, 2026.

[1] We discuss modern architectures that do not use centering in Appendix 10.2.

provide a mechanism for computational efficiency or improved interpretability. The answer depends on whether this suppression, hard or soft, hurts model performance.

When high sparsity is harmful. To answer this question, we vary the sparsity level using two approaches, Percentile Centering and Top-K sparsity (Shu et al., 2025), and find that performance stays nearly flat at moderate activation sparsity and collapses abruptly once sparsity exceeds $\sim$70%. Moreover, we demonstrate that skip connections improve robustness to sparsity.

Sparsity-inducing activation functions. Having established that weight drift and normalization jointly govern sparsity in ReLU-based models, we ask whether alternative activation functions offer a more favorable sparsity–accuracy tradeoff. We evaluate ReLU^2 (So et al., 2022), NoisyReLU (Gulcehre et al., 2016), and SUGARB-SiLU (Horuz et al., 2025) as candidates that may simultaneously enhance sparsity and model performance. The latter two are specifically designed to overcome the zero-gradient problem of ReLU, in which neurons with negative pre-activations receive no gradient (Lu et al., 2019). We find that ReLU^2 achieves lower loss than both ReLU and GELU on GPT-nano across three seeds, but it is highly sensitive to the choice of normalization. It works well only with LayerNorm (Ba et al., 2016) and RMSNorm (Zhang and Sennrich, 2019), and its performance degrades with BatchNorm or without normalization.

Spike amplification. In GPT-nano models, we identify spikes, i.e., large positive activation values, in early layers, and find that ReLU^2 amplifies them. This raises once more the question of whether this amplification is a **Bug or a Feature**[2]. We find that amplified spikes may become damaging, whereas clipping the amplified activations mitigates it.

An efficiency bonus. Finally, the fact that the weight drift happens only during first iterations yields a practical result. Since the critical dynamics stabilize after only a few iterations, centering statistics, quantile shifts, and Top-K thresholds can all be computed as running means exclusively over these early steps enabling savings in compute time without sacrificing effectiveness.

The paper is organized as follows. §2 and §3 formally and empirically characterize weight drift. §4 analyzes controllable post-activation sparsity and its relationship to model accuracy. §5 evaluates sparsity-inducing activation functions, while §6 examines activation spikes in GPT-nano models. §7 discusses computational efficiency gains enabled by early drift stabilization, and §10 covers related work. Appendix 10.5 covers proofs, implementation details, and extended experimental results over different settings and models.

2. Formal Illustration of Negative Weight Drift

Throughout this section we restrict the formal argument to ReLU (Nair and Hinton, 2010) and demonstrate results for other activation functions empirically [2]. Consider a multilayer perceptron with randomly initialized, zero-mean weights, using ReLU activation without mean-centering normalization layers. We demonstrate that at initialization and during the first training iterations under MSE or cross-entropy loss, the gradient of the loss with respect to the pre-activations is positive in expectation. Since gradient descent applies updates in the negative direction of the gradient ($\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}}$), these consistently positive gradients drive downstream weights toward negative values. This negative weight drift in turn shifts pre-activations further below zero reinforcing the effect in a self-amplifying cycle. As training progresses and gradients diminish, the drift stabilizes. Our formal analysis applies to the early phase of training, when the properties of the random zero-mean initialization still reasonably hold.

Properties of the Effective Weight Matrix. Consider a network with L linear layers interleaved with ReLU activations. For a fixed input $\mathbf{x}$, the activation pattern of each ReLU is fixed, so each activation layer acts as a binary diagonal matrix $\mathbf{D}_l$, where $(\mathbf{D}_l)_{ii} = 1$ if the i-th neuron is active and 0 otherwise. Pick any intermediate layer l with pre-activation vector $\mathbf{p}^{(l)}$. All layers after l form the composition:

$$\mathbf{V}_{\text{eff}}^{(l)} = \mathbf{W}_L \, \mathbf{D}_{L-1} \, \mathbf{W}_{L-1} \cdots \mathbf{D}_l \, \mathbf{W}_{l+1}, \tag{1}$$

so that the network output can be written as

$$f(\mathbf{x}) = \mathbf{V}_{\text{eff}}^{(l)} \, \sigma\left(\mathbf{p}^{(l)}\right). \tag{2}$$

Theorem 2.1. *Let $\mathbf{V}_{\text{eff}}$ be as in (1) with $\mathbf{W}_L, \ldots, \mathbf{W}_{l+1}$ drawn from a zero-mean i.i.d. distribution, and $\sigma = \text{ReLU}$. Denote the rows of $\mathbf{V}_{\text{eff}}$ by $\mathbf{v}_1, \ldots, \mathbf{v}_{d_p}$. Then:*

$$\mathbb{E}[\mathbf{v}_i] = \mathbf{0}, \forall\, i, \quad \mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle] \geq 0, \forall\, i,\, j. \tag{3}$$

Proof of the first statement. Each row of $\mathbf{V}_{\text{eff}}$ can be written as $\mathbf{v}_i = [\mathbf{D}_{L-1} \mathbf{W}_{L-1} \cdots \mathbf{D}_l \mathbf{W}_{l+1}]^\top [\mathbf{W}_L]_i$, where $[\mathbf{W}_L]_i$ denotes the i-th row of $\mathbf{W}_L$. Since $\mathbb{E}[W_{ij}] = 0$ for all entries and the diagonal matrices $\mathbf{D}_k$ are fixed (determined by the input), the expectation factors through the outermost weight matrix, giving $\mathbb{E}[\mathbf{v}_i] = \mathbf{0}$. □

Proof of the second statement. For ReLU, each $\mathbf{D}_l$ is a binary diagonal matrix that selects active neurons, so $\mathbf{V}_{\text{eff}}$ is a product of random weight matrices with inactive rows

[2] The formal extension of Theorems 2.2 and 2.3 to smooth activations requires a continuous analogue of the survival-conditioning argument and is beyond the scope of this paper.

zeroed out. At each ReLU gate $\mathbf{D}_k$, the row survives only if the corresponding pre-activation is non-negative, i.e., its inner product with the layer input is ≥ 0. The property $\mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle] \geq 0$ thus reduces to a property of random vectors conditioned on ReLU survival: it suffices to show that for any two random vectors $\mathbf{a}$ and $\mathbf{b}$ drawn from a zero-mean i.i.d. distribution, conditioned on $\mathbf{a}^\top \mathbf{x} \geq 0$ and $\mathbf{b}^\top \mathbf{x} \geq 0$ for a fixed input $\mathbf{x}$, we have $\mathbb{E}[\mathbf{a}^\top \mathbf{b}] \geq 0$. Intuitively, conditioning on survival forces both vectors to share a positive component along the input direction, inducing a positive correlation. By the symmetry of the distribution we may choose coordinates so that $\mathbf{x} = [1, 0, \ldots, 0]$. The conditions $\mathbf{a}^\top \mathbf{x} \geq 0$ and $\mathbf{b}^\top \mathbf{x} \geq 0$ then reduce to $a_0 \geq 0$ and $b_0 \geq 0$, so we can write $\mathbf{a} = [|a_0|, a_1, \ldots, a_n]$ and $\mathbf{b} = [|b_0|, b_1, \ldots, b_n]$. Then:

$$\mathbb{E}[\mathbf{a}^\top \mathbf{b}] = \mathbb{E}[|a_0| \cdot |b_0|] + \mathbb{E}\left[\sum_{i \geq 1} a_i b_i\right].$$

The first term is strictly positive since $|a_0|$ and $|b_0|$ are positive random variables. The second term equals zero by the zero-mean i.i.d. assumption on the entries. Thus $\mathbb{E}[\mathbf{a}^\top \mathbf{b}] \geq 0$. □

Positive Expected Gradient under MSE and Cross-Entropy. Here we show that, at initialization, the gradient of the loss with respect to any positive pre-activation is non-negative in expectation, both for MSE regression and for softmax cross-entropy classification. We state the two results in parallel, their proofs are provided Sections A.1 and A.2, respectively.

Theorem 2.2 (MSE loss)**.** *Let $f(\mathbf{x}) = \mathbf{V}_{\text{eff}}^{(l)} \sigma(\mathbf{p}^{(l)})$ be as in (2), with $\sigma = \text{ReLU}$ and $\mathbf{V}_{\text{eff}}^{(l)}$ satisfying Theorem 2.1. Assume the network is at initialization, so that $\mathbf{V}_{\text{eff}}^{(l)}$ is independent of $\mathbf{p}^{(l)}$ and of the target $\mathbf{y}$. Consider the MSE loss $\ell(f(\mathbf{x}), \mathbf{y}) = \frac{1}{2}\|f(\mathbf{x}) - \mathbf{y}\|_2^2$. Then for any neuron i, $\mathbb{E}\left[\frac{\partial \ell}{\partial p_i^{(l)}}\right] \geq 0$, with strict inequality whenever $p_i^{(l)} > 0$, where the expectation is taken over $\mathbf{V}_{\text{eff}}^{(l)}$.*

Theorem 2.3 (Cross-entropy loss)**.** *Let $f(\mathbf{x}) = \mathbf{V}_{\text{eff}}^{(l)} \sigma(\mathbf{p}^{(l)})$ be as in (2), with $\sigma = \text{ReLU}$ and $\mathbf{V}_{\text{eff}}^{(l)} \in \mathbb{R}^{C \times d_p}$ satisfying Theorem 2.1, where C denotes the number of output classes. Assume the network is at initialization, so that $\mathbf{V}_{\text{eff}}^{(l)}$ is independent of $\mathbf{p}^{(l)}$ and of the one-hot label $\mathbf{y}$. Consider the cross-entropy loss $\ell(f(\mathbf{x}), \mathbf{y}) = -\sum_{c=1}^{C} y_c \log s_c$, where $\mathbf{s} = \text{softmax}(f(\mathbf{x}))$. Then for any neuron i, $\mathbb{E}\left[\frac{\partial \ell}{\partial p_i^{(l)}}\right] \geq 0$, with strict inequality whenever $p_i^{(l)} > 0$, where the expectation is taken over $\mathbf{V}_{\text{eff}}^{(l)}$, up to corrections of order $O(\|\mathbf{f}\|^2)$ from the softmax linearization around $\mathbf{f} = \mathbf{0}$.*

Extension to Arbitrary Depth and Locality. Theorems 2.1 and 2.2 hold for any intermediate layer l, so the positive-gradient property and the resulting negative weight drift propagates through the entire network. For each $l \in \{1, \ldots, L-1\}$, we fold all subsequent layers into $\mathbf{V}_{\text{eff}}^{(l)}$ as in (1). By Theorem 2.1, $\mathbf{V}_{\text{eff}}^{(l)}$ satisfies the zero-mean and non-negative cross-correlation conditions at initialization. By Theorem 2.2, the gradient with respect to any positive pre-activation $p_i^{(l)}$ at layer l is non-negative in expectation over the downstream weights, with strict positivity whenever $p_i^{(l)} > 0$. Consequently, weights at every layer experience a non-positive expected update, i.e. negative drift. This consequently shifts pre-activations at layer l downward, increasing the fraction of neurons falling below zero reinforcing the cycle.

Although we present the analysis for a ReLU MLP without normalization or skip connections, the argument is local, the structural property established in Theorem 2.1 depends only on the composition of subsequent linear layers and ReLU gates, and applies to any contiguous stack of such layers within a larger architecture. The argument also carries over to architectures with skip connections (Appendix A.3), and we discuss its extension beyond initial training steps in Appendix A.4.

3. Empirical Results for Negative Weight Drift

In the previous section we established that gradient descent drives weights negative in expectation during early training. We now verify this empirically across optimizers, learning rates, architectures, and activation functions.

Weight drift depends on optimizer and learning rate. First, we analyze how quickly model weights evolve during the first batch updates and what weight drift depends on. Let $w_0 \in \mathbb{R}$ denote a scalar weight at initialization and w_t its value after t gradient steps. We measure relative drift per layer using the *Z-score*=$\mathbb{E}[|w_t - w_0|/\text{std}(w_0)]$, where the expectation is taken over all weights in a layer and the absolute value captures drift in both directions symmetrically. Results in Figure 2 demonstrate that: **(1)** momentum substantially accelerates drift, **(2)** within each optimizer, higher learning rates produce faster and larger drift and **(3)** momentum-based optimizers exhibit a rapid initial surge that then plateaus, while plain SGD progresses more slowly and near-linearly. We hypothesize that momentum amplifies positive gradient bias at early accumulation steps. Additional results on training dynamics are presented in §B.1.

Drift is intrinsic to optimization, not data. To demonstrate that negative weight drift is an intrinsic property of the optimization process, we train the same MLP on random $\{X, Y\}$ pairs sampled from $\mathcal{N}(0, 1)$. As shown in Fig-

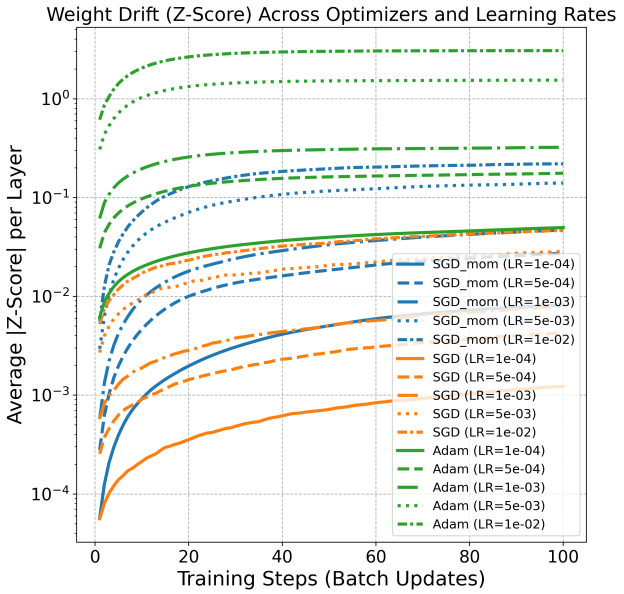


Figure 2. Weight drift measured as average absolute Z-score per layer over the first 100 training steps for an MLP trained on CIFAR-10. Momentum accelerates initial weight changes and leads to rapid convergence toward asymptotic drift levels, while plain SGD exhibits slower progression. Results are in log scale.

ure 3a, negative drift arises across all activation functions even on entirely random data. For ReLU, whose positive activation bias is strictly enforced at every layer, the drift exhibits a clear depth ordering, where earlier layers accumulate more negative weights. For SiLU (Elfwing et al., 2018) and GELU (Hendrycks and Gimpel, 2016), whose outputs are only positively *biased* rather than strictly non-negative, the drift is weaker and the last layer barely drifts, though the overall drift remains. First layers do not experience drift since their inputs have not yet passed through an activation function.

Negative weight drift across architectures and activation functions. Theorems 2.2 and 2.3 predict a positive expected gradient at initialization, and we find that this prediction holds broadly in practice. Across four architectures, MLP, MaxViT-Tiny (Tu et al., 2022), MP-SENet (Lu et al., 2023) shown in Figure 4, and ResNet-18 (He et al., 2016) shown in Appendix Figure 8, the positive-gradient property is observed. For the MLP and ResNet-18, where we measure it, the covariance term $\mathrm{Cov}_{\mathbf{x}}(\partial \ell / \partial p_i, x)$ remains orders of magnitude smaller than the weight mean, validating the assumptions made in Appendix A.1. The results are consistent across the activation functions GELU, ReLU, and SiLU and models with skip connections. Adding a skip connection to the MLP leaves the drift trajectory qualitatively unchanged under all four tested activation functions (Appendix Figure 9). The residual architectures ResNet-18 and MaxViT also exhibit the predicted positive gradient bias.

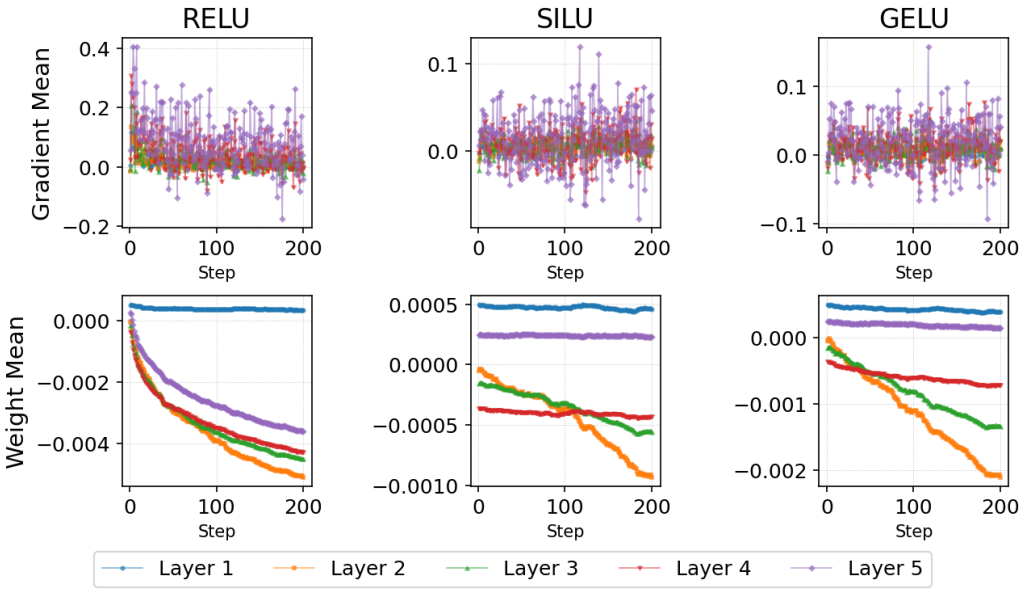
Table 1 reports accuracy and the fraction of negative pre-activation values for MLP, ResNet, ViT, and GPT across six activation functions. As a direct consequence of negative weights, the fraction of negative pre-activations is substantial in nearly every configuration typically between 60% and 80% confirming that weight drift is present throughout. The one exception is ResNet with batch normalization, where mean-centering directly disrupts the drift.

4. Post-activation Sparsity and Performance

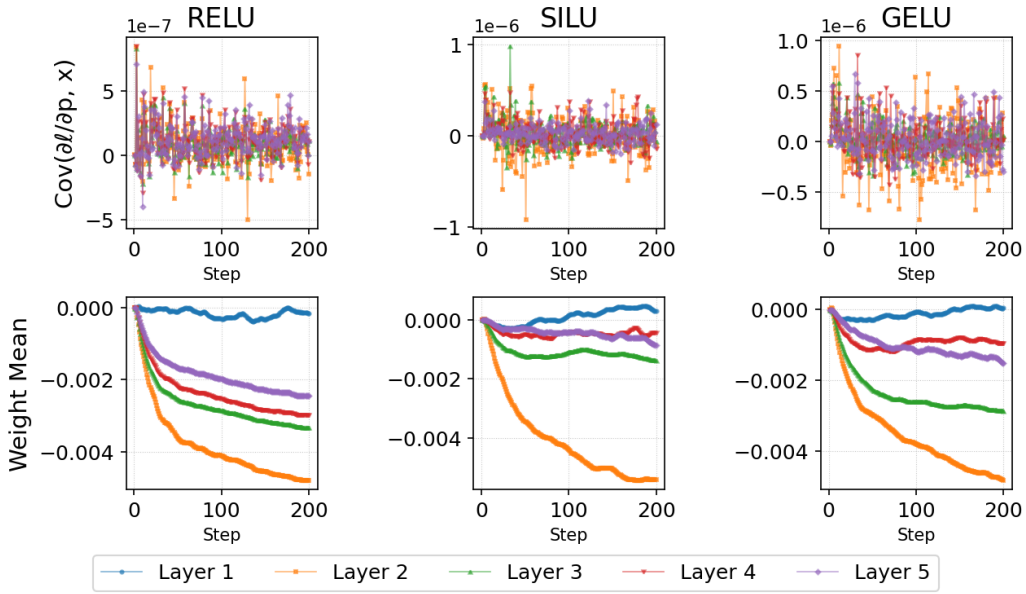
Controllable Sparsity. Weight drift naturally induces hard activation sparsity in ReLU-based models, while for smooth activations such as GELU and SiLU, it pushes pre-activations into near-zero regions, yielding predominantly low-magnitude outputs. Since this behavior emerges directly from the optimization dynamics we want to evaluate if the resulting sparsity impair model performance, or could resulting sparsity instead be beneficial? To answer this, we control post-activation sparsity levels across architectures and investigate whether explicitly enforcing higher or lower sparsity improves downstream performance. We further examine whether an optimal sparsity regime exists that outperforms the baseline naturally induced by weight drift.

Sparsification mechanisms. We consider two methods to control post-activation sparsity. One commonly used in the literature, and one we propose specifically to evaluate ReLU-induced sparsity at controlled levels. *Top-K* activation sparsity (Shu et al., 2025) retains the $k\%$ largest activations and hard-zeros the rest. We pair Top-K with GELU rather than ReLU, since ReLU already induces sparsity via weight drift, making it impossible to reliably control the sparsity lower bound. To evaluate controlled sparsity in ReLU-based models directly, we propose **Percentile Centering** (PC), which integrates into existing normalization layers with minimal architectural changes. Rather than shifting activations by the mean as in standard BatchNorm (BN) or LayerNorm, PC shifts by a target percentile q, $\hat{x} = \frac{x - Q_x(q)}{\sqrt{\sigma_x^2 + \epsilon}}$, where $Q_x(q)$ denotes the q-th percentile of the pre-activation distribution and σ_x^2 its variance. When followed by ReLU, this causes a $q\%$ fraction of activations to fall below zero, directly controlling post-activation sparsity. This is particularly convenient for architectures like ResNet, where BN is placed immediately before ReLU. In our implementation, PC maintains a running mean of the percentile estimate analogous to the running statistics in BN.

Experimental setup. We evaluate both mechanisms across four architectures: MLP, ResNet-18, MaxViT-Tiny, and GPT-nano (Karpathy, 2022). For ResNet-18, we additionally distinguish between *per-activation* sparsity, which



(a) Random inputs, MSE loss, SGD.



(b) CIFAR-10, cross-entropy loss, Adam.

Figure 3. **Negative weight drift in a five-layer MLP without normalization. (a)** Training on random $\{X, Y\}$ pairs sampled from $\mathcal{N}(0, 1)$, the top row shows the mean gradient and the bottom row the mean weight per layer. **(b)** Training on CIFAR-10, the top row shows the covariance term $\mathrm{Cov}_{\mathbf{x}}(\partial \ell / \partial p_i, x)$, which is three to four orders of magnitude smaller than the weight mean (bottom row). Trajectories are averaged across 5 runs. Weights drift negative for all three activation functions. Weights of Layer 1, whose input has not passed through an activation, do not drift, as predicted. Further details are given in §D.2.

zeros individual activation values, and *per-channel* (structured) sparsity, which zeros entire feature map channels with the latter being more relevant for practical applications. In total, we obtain $N = 79$ (model, sparsity) pairs across architectures, activation functions, and sparsification mechanisms. *Sparsity is measured exclusively at the output of activation functions. Consequently, we do not account for intermediate model components that lack activation functions, for example, attention layers in transformers.* Technical details are outlined in §D.

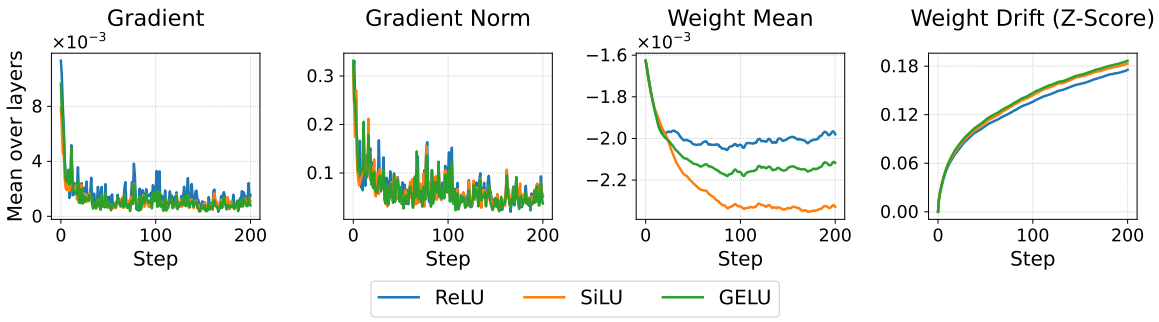


Figure 4. **Weight drift in MP-SENet.** Training dynamics under GELU, ReLU, and SiLU with the AdamW optimizer (lr $= 5 \times 10^{-4}$), averaged across all model layers. Drift patterns are qualitatively consistent with those observed in the MLP and ResNet settings.

4.1. Results for controlled post-activation sparsity experiments

While raw numerical results are presented in Tables 6 and 7 to enable comparison across architectures, we normalized performance metrics by the maximum value observed for each architecture–modification pair. Since we report accuracy for all models except GPT-nano (where we use loss), we inverted the loss values for GPT-nano so that higher values consistently indicate better performance. Visual inspection of the scaled metrics in Figure 5 and Appendix Figure 11 (each architecture separately), reveals that performance remains largely stable across moderate sparsity levels and degrades sharply only beyond a high threshold. We further fit a three-parameter power law via nonlinear least squares: $\hat{a}(s) = A - B \cdot s^N$, where $s \in [0, 1]$ denotes sparsity, and A, B, N are free parameters. The fitted coefficient $A = 0.978$ corresponds to the predicted performance at zero sparsity, confirming near-complete retention of accuracy when no activations are zeroed. $B = 0.635$ represents the maximum potential drop. The exponent $N = 16.72$ sets the sharpness of the transition, s^N remains negligible for $s \lesssim 0.7$ and grows rapidly thereafter so accuracy degrades. Together, these values quantify the qualitative "cliff" visible in Figure 5, with performance preserved across a wide plateau before abruptly degrading at high sparsity.

The position of this cliff is primarily determined by model architecture, with skip connections improving robustness. For example, a plain MLP collapses at 85% sparsity while MLP with skip connections allows the model to maintain 74.0% accuracy at that same level. Transformers, especially GPT-nano, exhibit extreme resilience, with loss remaining nearly flat up to $s \approx 0.91$. This robustness is partially attributed to skip connections, however, further investigation is required. In ResNet-18, channel-wise sparsity degrades more monotonically than unstructured Top-K sparsity, meaning that unstructured sparsity causes less model degradation.

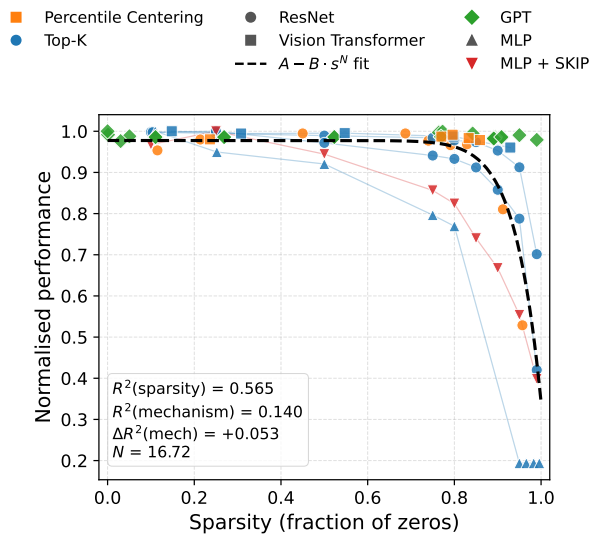


Figure 5. Scaled performance versus post-activation sparsity. The dashed curve denotes the fitted power-law decay model, with sparsity level is the dominant predictor of accuracy $R^2 = 0.565$, while the choice of mechanism (TOP-K vs. PC) contributes only a marginal incremental gain $\Delta R^2 = +0.053$.

> Empirical results reveal an **accuracy cliff** with a small preceding monotonic decline. Performance remains nearly flat for moderate sparsity ($s \lesssim 0.7$) before collapsing abruptly once a critical threshold is reached. MLP and ResNet-18 with structured sparsity exhibit more monotonic preceding degradation.

Finally, we observe that the specific *mechanism* of sparsification Top-K vs. Percentile Centering is of secondary importance, adding a mechanism indicator to the power law model marginally improves R^2 from 0.565 to 0.618, the *amount* of sparsity remains the dominant determinant of predictive accuracy.

Table 1. Accuracy (or test loss for GPT) and fraction of negative pre-activations across architectures and activation functions. For MaxViT-Tiny we replace only default GELU activations with the specified function. Best per-row sparsity and performance are in **bold**. Details on statistics measurement are provided in § D.1. **Format: Accuracy / Negative values.**

Model	ReLU	GELU	SiLU	NoisyReLU	SUGARBSiLU	ReLU^2
MLP	46.6% / 70.8%	**51.1**% / 63.6%	49.8% / 61.0%	49.1% / 70.4%	40.8% / **72.5**%	10.0% / 8.0%
MLP + RMSNorm	44.7% / 68.4%	48.4% / 64.5%	**49.4**% / 62.3%	48.4% / 66.8%	48.6% / **70.8**%	48.0% / 65.3%
MLP + LayerNorm	40.9% / **68.6**%	42.3% / 67.2%	43.9% / 65.4%	41.8% / 66.4%	41.0% / 58.5%	**45.3**% / 62.5%
ResNet (BN)	93.8% / 43.6%	**94.1**% / 43.0%	93.3% / 43.6%	93.5% / 43.1%	92.1% / **55.5**%	10.0% / 0.0%
MaxViT-Tiny	69.6% / 79.8%	**70.3**% / 70.5%	69.4% / 70.2%	68.5% / 79.7%	51.5% / **82.9**%	62.4% / 77.1%
GPT (loss↓)	3.290 / 89.0%	3.260 / **89.3**%	–	3.287 / 89.2%	3.291 / 89.1%	**3.251** / 84.2%
Average:	59.1% / 66.2%	**61.2**% / 61.8%	**61.2**% / 60.5%	60.3% / 65.2%	54.8% / **68.0**%	51.9% / 68.3%

5. Sparsity and Activation Functions

§3 established that weight drift naturally induces sparsity in ReLU-based models. *We now ask whether alternative activation functions can achieve a more favorable sparsity–accuracy tradeoff, either by inducing sparsity through different mechanisms or by recovering ReLU-style sparsity post-training.* We evaluate five activation functions including GELU and ReLU baselines across four architectures (MLP, ResNet-18, ViT, GPT-nano).

Candidate activation functions. *(1) NoisyReLU* (Gulcehre et al., 2016) injects input-dependent noise into negative pre-activations during training, maintaining gradient flow through otherwise dead neurons (Lu et al., 2019) while reverting to standard ReLU at inference. *(2) SUGARBSiLU* (Horuz et al., 2025) applies ReLU in the forward pass but substitutes a smooth surrogate gradient (B-SiLU) in the backward pass, ensuring nonzero gradient signal for negative pre-activations. *(3) ReLU^2*, originally discovered through neural architecture search (So et al., 2022), has since been shown to achieve a strong sparsity–accuracy tradeoff in LLMs (Zhang et al., 2024). Implementation details for all three are provided in §D.7.

ReLUfication (Mirzadeh et al., 2023; Chua et al., 2024) while not an activation function it allows one to reach the same goal, therefore we also consider this procedure. Rather than training with a sparsity-inducing activation function from scratch, models trained with a smooth activation (e.g., GELU) are converted to ReLU variants via brief fine-tuning. The motivation is that smooth activations avoid the dying-neuron problem during training, while post-hoc conversion recovers inference-time sparsity.

5.1. Results for sparsity and activation functions

Results are presented in Table 1. GELU is the strongest general-purpose baseline. It achieves the highest accuracy on four of five classification settings and the second-best loss on GPT-nano, however, it does not produce sparsity. ReLU^2 is normalization-sensitive but excels on GPT. It collapses entirely on MLP without normalization and ResNet with BN, but recovers when used with RMSNorm or LayerNorm. SUGARBSiLU produces the highest sparsity at the cost of stability. We attribute this to persistently noisy gradient trajectories throughout training (Figure 9 in Appendix), suggesting the surrogate gradient introduces optimization instability. NoisyReLU matches ReLU's sparsity with comparable accuracy. NoisyReLU achieves average sparsity close to ReLU's and competitive accuracy. The margins are small but consistent, suggesting NoisyReLU is a viable drop-in replacement when gradient flow through dead neurons is desired.

Table 2. ReLUfication results. Models trained with GELU are converted to ReLU variants via one epoch of fine-tuning. Format: **Accuracy / Negative pre-activations / Post-activation sparsity**.

Model	Activation	Acc / Neg. Val / Spar.
MLP	GELU (baseline)	51.49% / 0.632 / 0.026
	ReLU (1 epoch FT)	51.55% / 0.704 / 0.704
ResNet	GELU (baseline)	94.03% / 0.583 / 0.080
	ReLU (1 epoch FT)	93.05% / 0.577 / 0.578
	NoisyReLU (1 epoch FT)	92.31% / 0.553 / 0.555
MaxViT-Tiny	GELU (baseline)	70.30% / 0.705 / 0.015
	ReLU (1 epoch FT)	70.15% / 0.740 / 0.740

ReLUfication yields 55–74% post-activation sparsity, while accuracy drops by less than 2 percentage points. The fraction of negative pre-activations changes little after conversion, and the resulting sparsity closely matches it. ReLUfication therefore barely changes the weights, weight drift has already pushed activations below zero such that GELU outputs values close to zero, and ReLU simply sets these low-magnitude activations to zero.

ReLUfication is the practical recommendation for the sparsity–accuracy tradeoff. Rather than training from scratch with ReLU or other sparsity-inducing activations, light fine-tuning from a GELU baseline creates high post-activation sparsity with an accuracy loss of less than 2 percentage points, and none for some models, sidestepping the optimization difficulties of training with ReLU.

6. Spike Amplification with Squared Activation Functions

Our layer-wise analysis reveals the presence of spikes—large positive activation values in early GPT-nano layers—regardless of the choice of activation function. We visualize these spikes in Figure 6 (right) and quantify them in Table 3. We measure spikes as input range $r = \max - \min$ at different layers. Since the down-projection input follows the activation function, its minimum is close to zero and r is essentially the maximum activation for that layer. While prior work has shown that training GPT-like models with ReLU^2 can improve performance (So et al., 2022), we show that it also amplifies spikes, which may degrade model performance. We further show that clipping ReLU^2 activations removes the spikes without degrading performance.

Origin of spikes: In our architecture, the MLP block takes the following form:

```
def forward(self, x):
    x = self.up_projection(x)
    x = self.activation(x)
    x = self.down_projection(x)
    return x
```

As shown in Figure 6, there are no spikes in the inputs to the **up-projection** (left), they emerge in the values entering the **down-projection** (right), precisely after the activation is applied. Since there is no gating interaction in our MLP block, the spike must originate in the up-projection itself, where a small subset of neurons produces anomalously large pre-activation values, which are further amplified by the nonlinearity. We perform an extended statistical analysis of weights and activations over all 23 runs in §B.3, which further supports this conclusion, and discuss possible mechanisms behind the spikes. Profiling six pretrained models shows that distinct early-layer spikes appear only in language models, but not in vision models, even with similar architectures (Appendix C).

Figure 6 reveals that spike ranges shrink when normalization with centering or quantile shift is applied. The same pattern holds for input ranges to the up-projection, which are also smaller under squared activations and under models that use centering or quantile shift. This points to an intricate interaction between normalization layers and activation functions, on top of the weight-drift mechanism discussed in §3.

Squared activations amplify spikes. As shown in Table 3, baseline activations (ReLU, GELU) exhibit moderate spikes at Layer-2. ReLU^2 amplifies this spike by a factor of 4 to 25 across three seeds, reaching values above 1000 in the run with the largest spike. Exact values for all seeds are given in Appendix Table 8.

Amplification can be avoided with simple clipping, so we test whether it would degrade performance or can be used safely. We clip ReLU^2 activations at thresholds of 15 and 50. We find that $\text{ReLU}^2_{\text{clip50}}$ removes the spikes without degrading performance, and it reaches the same loss in all three seeds, whereas the loss of unclipped ReLU^2 varies from seed to seed (Appendix Table 8). $\text{ReLU}^2_{\text{clip15}}$, however, degrades performance relative to $\text{ReLU}^2_{\text{clip50}}$, suggesting that aggressive clipping suppresses informative large activations.

7. From Weight Drift to Computational Efficiency

Dynamic computation of percentiles and centering statistics at every forward pass introduces non-negligible overhead. Quantile algorithms are poorly parallelizable on GPUs (Cederman and Tsigas, 2010; Satish et al., 2009), and even standard LayerNorm is a measurable bottleneck (Kanavalau et al., 2026; Ryabykin et al., 2026). Fortunately, the dynamic computation is not strictly necessary throughout training. In §3 we demonstrated that weights stabilize after the initial iterations, and we exploit this property by replacing the dynamic statistics with fixed values once the network has settled.

Concretely, during a warm-up phase we track centering and percentile statistics with an EMA scheme analogous to the running statistics in BatchNorm (Ioffe and Szegedy, 2015): $\bar{v}_i = \gamma \bar{v}_{i-1} + (1-\gamma) v_i$ with $\gamma = 0.9999$. After T_{warm} steps we apply *Accumulation Stop* (AS), freezing the buffer and eliminating dynamic computation entirely. We evaluate this approach on DiT-S/2 and MaxViT, both of which operate on fixed-size inputs. We note that this approach is less directly applicable to autoregressive models such as GPT, where activation statistics vary with sequence position and context length, which may make frozen thresholds less reliable across generation lengths.

Hardware and training configuration. DiT-S/2 experiments are conducted on Nvidia H200 (141 GB) GPUs using ImageNet-1K images resized to 256×256. MaxViT experiments use Nvidia A100 (80 GB) GPUs. All models are trained in bfloat16 mixed precision with percentile and

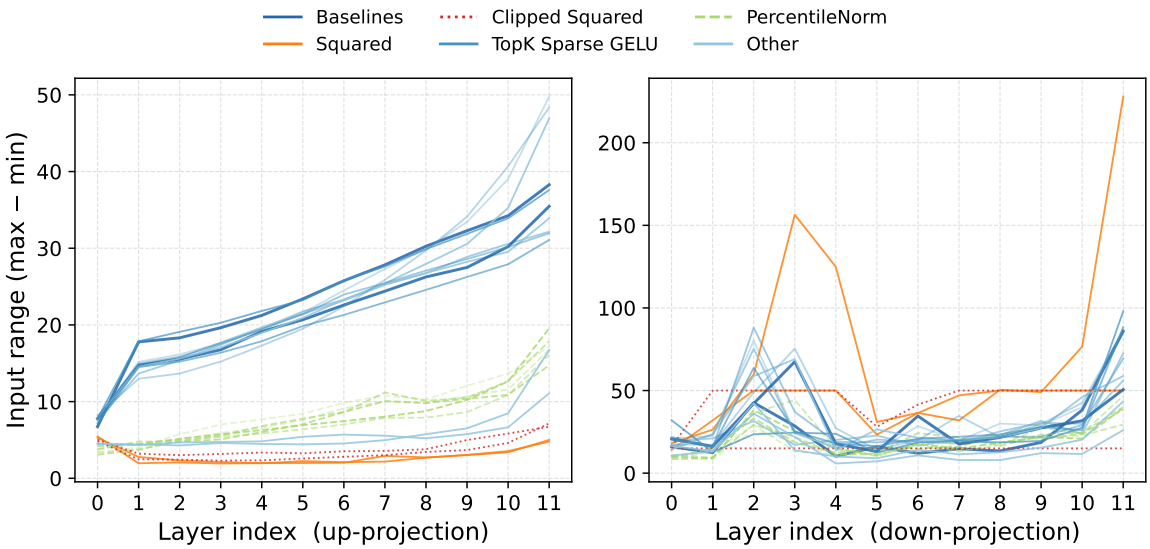


Figure 6. Input ranges into up- and down-projections across layer indices, aggregated over 21 runs for different activation functions, normalization and sparsification strategies.

Table 3. Loss and per-layer down-projection input range $r = \max - \min$ at different GPT-nano layers. ReLU^2 shows the biggest of three seeds spikes, all seeds are reported in Appendix Table 8.

	ReLU	**GELU**	**NReLU**	**SUGARBSiLU**	**ReLU^2**	**GELU^2**	**ReLU$^2_{\text{clip15}}$**	**ReLU$^2_{\text{clip50}}$**
Test loss ↓:	3.290	3.260	3.287	3.291	3.251	**3.233**	3.242	3.236
Layer 1	16.2	12.4	23.3	23.7	**108.5**	26.3	15.0	50.0
Layer 2	42.2	42.7	75.3	58.7	**1055.0**	59.8	15.0	50.0
Layer 3	67.5	28.5	24.2	69.0	**260.9**	156.4	15.0	50.0
Layer 4	17.8	10.2	15.3	13.7	49.8	124.9	15.0	50.0
Layer 5	13.1	16.0	20.3	26.5	65.3	31.0	15.0	28.1

centering thresholds computed channel-wise and averaged over the batch. The baseline uses PyTorch's optimized LayerNorm and BatchNorm kernels; our AS implementation uses plain PyTorch without custom CUDA kernels, so all reported throughput gains are conservative lower bounds.

Warm-up and measurement protocol. The warm-up phase runs for $T_{\text{warm}} = 50{,}000$ steps for DiT-S/2. For MaxViT-Tiny, warm-up and throughput measurement follow the four-phase protocol described in §D.5. For DiT-S/2, throughput is measured as batches per second averaged over 1,000 steps after warm-up, with GPU synchronization performed before measurement.

7.1. Results for computational efficiency

Throughput results. Dynamic computation reduces throughput by ∼25–30% during warm-up, once statistics are frozen, throughput fully recovers to near-baseline (optimized LN) levels with our naive PyTorch implementation, suggesting that an optimized kernel could be faster (Table 4).

Table 4. Training throughput (batches/sec, ↑) for DiT-S/2 (H200) and MaxViT-Tiny (A100) before and after Accumulation Stop ($T_{\text{warm}} = 50{,}000$ steps, non-optimized implementation).

Activation	**Norm**	**Accum.**	**Fixed**
DiT-S/2 — baseline: 3.42 (GELU + PyTorch LN)			
Percentile GELU	LN	2.47	3.43
GELU	Percentile LN	2.47	3.41
ReLU	Percentile LN	2.58	3.53
MaxViT — baseline: 1.59 (GELU + PyTorch BN)			
GELU	BN	1.55	**2.09**
Percentile GELU	BN	1.34	**2.13**
ReLU	Percentile BN	1.14	**1.73**

Quality preservation. Table 5 confirms that model quality is preserved after freezing statistics. For DiT-S/2 we report FID and IS computed on 50,000 generated samples using the EMA model with Classifier-Free Guidance

($s = 1.5$) and 250 sampling steps. Interestingly, we observe an improvement when DiT is trained with Percentile Shift instead of standard LayerNorm, with FID dropping from 49.40 to 48.21. Given the 50% shift, the only difference between Percentile LN and standard LN is that the former uses quantile-based shifting while the latter uses mean centering. One detail worth noting is that quantile computation in PyTorch propagates gradients only through a single quantile value. While we do not yet have a full explanation for this result, the direction warrants its own investigation. For MaxViT-Tiny we report top-1 accuracy on the ImageNet-1K validation set, all metrics slightly degrade within 2% relative change of the corresponding baseline from 70.30% to 69.75%.

Table 5. Model quality when Accumulation Stop is applied. DiT-S/2 reports FID↓ / IS↑ (**baseline: 49.40 / 29.85**.), MaxViT reports top-1 accuracy↑ (baseline: 70.30%).

Activation	Norm	Score
DiT-S/2		
Percentile GELU (50%)	LN	52.33 / 28.87
GELU	Percentile Shift (50%)	**48.21 / 31.41**
ReLU	Percentile Shift (50%)	51.44 / 29.00
MaxViT		
GELU	BN	69.75%
Percentile GELU (50%)	BN	69.96%
ReLU	Percentile BN (50%)	69.74%

Weight drift stability across the AS boundary. Appendix Figure 10 plots the average |Z-score| of weight drift before and after the AS boundary. Prior to AS, all configurations converge to ≈ 0.40 by step 100 regardless of activation or normalization choice. After the freeze, AS and No-AS trajectories remain closely aligned with no discontinuity, confirming that the transition to fixed EMA statistics preserves training stability.

8. Limitations and Discussion

We summarize the main limitations of our work below and each is discussed in greater detail in Appendix E. **(1)** Our theoretical results assume zero-mean i.i.d. weights and hold strictly only at initialization, however, we extend the discussion to drift after initialization in Appendix A.4. **(2)** The formal proof covers ReLU while its extension to smooth activations is supported empirically rather than formally. **(3)** Language modeling is evaluated only on FineWeb with GPT-nano (124M parameters), clipping thresholds for squared activations are selected empirically and appear architecture-dependent. **(4)** Accumulation Stop is evaluated on architectures with fixed-size inputs. and reported throughput gains use a naive PyTorch implementation against an optimized baseline. **(5)** The strong robustness of transformer architectures (ViT, GPT-nano) to aggressive sparsification is not explained by our analysis, and skip connections alone do not fully account for it. **(6)** Squared activation functions yield clear gains on autoregressive GPT-nano but fail to outperform plain GELU on MaxViT, suggesting a modality- or objective-specific interaction that we do not characterize. **(7)** Likewise, the improved FID and IS scores observed for DiT-S/2 under Percentile LayerNorm with GELU point to a beneficial interaction between centering at a non-zero percentile and generative training that we leave for future investigation.

9. Conclusion

We described negative weight drift, a systematic shift of weights toward negative values caused by the interaction between standard losses and positively biased activation functions. The drift emerges in the first iterations of training, persists even on random data, and appears across architectures that lack mean-centering normalization. Because the drift pushes pre-activations below zero, it determines how many activations are zeroed or suppressed, and it should therefore be accounted for when designing new architectures, especially as modern models replace mean-centering normalization with RMSNorm, which does not counteract it.

The sparsity induced by drift is a feature rather than a bug, since performance is nearly unaffected up to $\sim 70\%$ post-activation sparsity. Sparsity levels can be controlled through the choice of activation function and post-training procedures such as ReLUfication. However, some activation functions, such as ReLU^2, may amplify large activation values, which may degrade performance. Both weight drift and spikes depend on normalization, since mean-centering suppresses drift and centering or percentile shifts reduce spike magnitude.

The common theme of these findings is the activations themselves. While this paper mainly focuses on weight drift and its effect on sparsity, these results show that drift, activation functions and normalization should be considered together, as they jointly shape the activations of a network and contribute to overall performance.

10. Related Work

10.1. Weight drift and internal covariate shift

The concept of *internal covariate shift* was introduced by Ioffe and Szegedy (2015) to describe the continuously changing distribution of layer inputs during training, as upstream weights update, the activations they produce shift, forcing downstream layers to perpetually adapt to a moving target. Batch Normalization (BN) was proposed as a remedy by normalizing layer inputs to zero mean and unit

variance, with subsequent work establishing additional benefits beyond distributional stability (Santurkar et al., 2018). Our work touches the same phenomenon but from a different angle. Rather than studying how changing activations force downstream weights to adapt, we study how positively biased activation functions *cause* weights to drift systematically negative. The most important connection is that while mean-centering normalization was developed to overcome covariate shift, it also incidentally suppresses weight drift by removing the positive pre-activation bias that drives it. However, modern architectures increasingly adopt normalization layers without centering such as RMSNorm (Zhang and Sennrich, 2019) meaning weight drift is likely to occur in these settings.

10.2. Normalization layers and centering in modern architectures

Most contemporary large language models replace LayerNorm (Ba et al., 2016) with RMSNorm (Zhang and Sennrich, 2019), which rescales activations by their root-mean-square but does not subtract the mean. This includes the LLaMA family (Touvron et al., 2023) and most other open-weight LLMs released after 2023 (Mistral, Gemma, Qwen, DeepSeek). The original motivation was efficiency, Zhang and Sennrich (2019) reported that the centering step contributed little to model quality while accounting for noticeable computational cost. Vision and generative-image architectures present a more heterogeneous picture. Classical ViTs (Tu et al., 2022) retain LayerNorm, and U-Net diffusion backbones rely on GroupNorm, both of which center activations. The original Diffusion Transformer (Peebles and Xie, 2023) and its successors use adaptive LayerNorm (adaLN) for timestep and class conditioning, again preserving centering on the main residual path. Moreover, recent large-scale diffusion transformers have begun adopting RMSNorm in the attention pathway. Stable Diffusion 3 and SD 3.5 (Esser et al., 2024) and the FLUX family (Black Forest Labs, 2024) apply RMSNorm to query and key projections to stabilize mixed-precision training, while retaining adaLN elsewhere.

10.3. Emergent activation sparsity

Li et al. documented that activation sparsity emerges in trained transformers without explicit regularization and persists on random data, implicating the optimizer rather than the data distribution as the causal factor. Our work can be seen as both an extension and a deepening of this finding. We show that weight drift is governed by the local interaction between the activation function, the normalization layer, and the loss function. We identify *negative weight drift* as the precise mechanistic cause of activation sparsity. On the theoretical side, our proof of gradient positivity extends that of Li et al. (2022) in two directions, we establish the result for *any* intermediate layer rather than only the final two, and we operate under strictly weaker assumptions by permitting non-negative off-diagonal weight correlations rather than requiring them to be zero. Empirically, we go beyond characterizing sparsity as a phenomenon by evaluating whether naturally arising sparsity hurts model performance or not.

10.4. Practical side of sparse activations

Pruning is most effective when the values to be pruned are already near zero, which is naturally the case for the intermediate representations of many LLMs (Liu et al., 2023; Li et al., 2022). This has motivated a line of work exploiting activation sparsity to accelerate the decoding stage via faster *sparse vector–dense matrix* multiplications, achieving up to $2\times$ speedup using specialized kernels (Song et al., 2024a;b; Liu et al., 2024; Lee et al., 2024). These gains are most pronounced at batch size 1 and diminish as batch size increases (Shrestha et al., 2025), since the speedup arises by skipping rows of the weight matrix that correspond to zero activations an advantage that erodes under batched computation. Many of these methods additionally require a predictive mechanism to prefetch the relevant weight indices into memory ahead of time (Liu et al., 2023). Our work complements this line by identifying the optimization dynamics responsible for producing the sparsity these methods depend on, and by characterizing the sparsity–accuracy tradeoff that determines how aggressively it can be exploited without degrading model quality.

10.5. Activation spikes at intermediate layers

Sun et al. studied massive activation spikes in SwiGLU-based transformers, attributing them to a directional quadratic amplification mechanism arising from the interaction between gate and up projections. We independently observed qualitatively similar spikes in standard non-gated MLP blocks, a finding we arrived at separately while studying the effects of squared nonlinearities on GPT-nano. The fact that spikes arise in both gated and non-gated architectures suggests that intermediate-layer spiking is a general property of transformer language models rather than a SwiGLU-specific artifact as suggested by Sun et al. (2026). This is further supported by Yusupov et al. (2025), who observe consistent early-layer spikes in geometric properties of internal representations including Maximum Explainable Variance, Effective Rank, and Intrinsic Dimensionality across multiple models. We contribute three further observations: (1) normalization reduces but does not eliminate the spikes, (2) the phenomenon originates in the learned weight matrices themselves and (3) controlling spike amplitude stabilizes training. Regarding the origin of spikes, Barbero et al. (2025) argue that LLMs attend to the first token to avoid over-mixing of token representations, so the first or "special" tokens acquire a functional role. This of-

fers a possible explanation for why spikes concentrate on specific tokens in early layers. Our profiling of pretrained models presented in Appendix C is consistent with this view. We observe that distinct early-layer spikes appear in GPT-2 and BERT but not in vision models.

References

Dong Shu, Xuansheng Wu, Haiyan Zhao, Daking Rai, Ziyu Yao, Ninghao Liu, and Mengnan Du. A survey on sparse autoencoders: Interpreting the internal mechanisms of large language models. *arXiv preprint arXiv:2503.05613*, 2025.

David R So, Wojciech Manke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. Primer: Searching for efficient transformers for language modeling, 2022. *URL https://arxiv. org/abs/2109.08668*, 2022.

Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. In *International conference on machine learning*, pages 3059–3068. PMLR, 2016.

Coşku Can Horuz, Geoffrey Kasenbacher, Saya Higuchi, Sebastian Kairat, Jendrik Stoltz, Moritz Pesl, Bernhard A Moser, Christoph Linse, Thomas Martinetz, and Sebastian Otte. The resurrection of the relu. *arXiv preprint arXiv:2505.22074*, 2025.

Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

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

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107: 3–11, 2018.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. Maxvit: Multi-axis vision transformer. In *European conference on computer vision*, pages 459–479. Springer, 2022.

Ye-Xin Lu, Yang Ai, and Zhen-Hua Ling. Mp-senet: A speech enhancement model with parallel denoising of magnitude and phase spectra. In *Interspeech 2023*, pages 3834–3838. ISCA, August 2023. doi: 10.21437/interspeech.2023-1441. URL http://dx.doi.org/10.21437/Interspeech.2023-1441.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Andrej Karpathy. nanoGPT, 2022. URL https://github.com/karpathy/nanoGPT.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. Relu2 wins: Discovering efficient activation functions for sparse llms. *arXiv preprint arXiv:2402.03804*, 2024.

Seyed Iman Mirzadeh, Keivan Alizadeh-Vahid, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Vui Seng Chua, Yujie Pan, and Nilesh Jain. Post-training statistical calibration for higher activation sparsity. *arXiv preprint arXiv:2412.07174*, 2024.

Daniel Cederman and Philippas Tsigas. Gpu-quicksort: A practical quicksort algorithm for graphics processors. *Journal of Experimental Algorithmics (JEA)*, 14:1–4, 2010.

Nadathur Satish, Mark Harris, and Michael Garland. Designing efficient sorting algorithms for manycore gpus. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–10. IEEE, 2009.

Andrei Kanavalau, Carmen Alonso, and Sanjay Lall. Gated removal of normalization in transformers enables stable training and efficient inference, 02 2026.

Alexey Ryabykin, Irina Zhelavskaya, Egor Shvetsov, Alexey Rukhovich, Nikita Okhotnikov, Artem Khrapov, Evgeny Burnaev, and Vladimir Mikhailovich Kryzhanovskiy. Damn: Deleting and migrating normalization layers from transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2883–2892, 2026.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/905056c1ac1dad141560467e0a99e1cf-Abstract.html.

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.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

Black Forest Labs. FLUX.1. https://github.com/black-forest-labs/flux, 2024.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. *arXiv preprint arXiv:2210.06313*, 2022.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR, 2023.

Yixin Song, Haotong Xie, Zhengyan Zhang, Bo Wen, Li Ma, Zeyu Mi, and Haibo Chen. Turbo sparse: Achieving llm sota performance with minimal activated parameters. *arXiv preprint arXiv:2406.05955*, 2024a.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 590–606, 2024b.

James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. Training-free activation sparsity in large language models. *arXiv preprint arXiv:2408.14690*, 2024.

Donghyun Lee, Je-Yong Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. Cats: Contextually-aware thresholding for sparsity in large language models. *arXiv preprint arXiv:2404.08763*, 2024.

Susav Shrestha, Brad Settlemyer, Nikoli Dryden, and Narasimha Reddy. Polar sparsity: High throughput batched llm inferencing with scalable contextual sparsity. *arXiv preprint arXiv:2505.14884*, 2025.

Shangwen Sun, Alfredo Canziani, Yann LeCun, and Jiachen Zhu. The spike, the sparse and the sink: Anatomy of massive activations and attention sinks. *arXiv preprint arXiv:2603.05498*, 2026.

Viacheslav Yusupov, Danil Maksimov, Ameliia Alaeva, Anna Vasileva, Anna Antipina, Tatyana Zaitseva, Alina Ermilova, Evgeny Burnaev, and Egor Shvetsov. From internal representations to text quality: A geometric approach to llm evaluation. *arXiv preprint arXiv:2509.25359*, 2025.

Federico Barbero, Álvaro Arroyo, Xiangming Gu, Christos Perivolaropoulos, Michael Bronstein, Petar Veličković, and Razvan Pascanu. Why do LLMs attend to the first token? *arXiv preprint arXiv:2504.02732*, 2025.

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and Maosong Sun. Prosparse: Introducing and enhancing intrinsic activation sparsity within large language models. *arXiv preprint arXiv:2402.13516*, 2024c.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *International Conference on Learning Representations*, 2024.

Appendix

A Theory **14**
- A.1 Theorem & proof: positive expected gradient for MSE loss . . . 14
- A.2 Theorem & proof: positive expected gradient for cross-entropy loss . . . 14
- A.3 Extension to skip connections . . . 15
- A.4 Weight drift beyond initialization . . . 16

B Extended Results **16**
- B.1 Additional experiments on weight drift . . . 16
  - B.1.1 Effect of Top-K sparsity on weight drift . . 16
  - B.1.2 Cross-architecture validation . . . 16
  - B.1.3 MLP with and without skip connections . 16
  - B.1.4 Weight drift when normalization statistics are frozen . . . 17
- B.2 Controllable sparsity . . . 18
- B.3 GPT-nano . . . 19
- B.4 Squared activation functions in ViT . . . 19

C Activation Spikes in Pretrained Models **21**

D Technical Details **21**
- D.1 Shared dataset and training configuration . . . 21
- D.2 MLP experiments . . . 22
- D.3 ResNet-18 experiments . . . 22
- D.4 DiT-S/2 configuration . . . 22
- D.5 MaxViT-T configuration . . . 23
- D.6 GPT-nano training details . . . 24
- D.7 Activation function implementations . . . 24

E Extended Limitations and Discussion **25**
- E.1 Limitations . . . 25
- E.2 Additional discussion . . . 26

A. Theory

A.1. Theorem & proof: positive expected gradient for MSE loss

Theorem A.1 (MSE loss)**.** *Let $f(\mathbf{x}) = \mathbf{V}_{\text{eff}}^{(l)}\,\sigma(\mathbf{p}^{(l)})$ be as in (2), with $\sigma = \text{ReLU}$ and $\mathbf{V}_{\text{eff}}^{(l)}$ satisfying Theorem 2.1. Assume the network is at initialization, so that $\mathbf{V}_{\text{eff}}^{(l)}$ is independent of $\mathbf{p}^{(l)}$ and of the target $\mathbf{y}$. Consider the MSE loss $\ell(f(\mathbf{x}), \mathbf{y}) = \frac{1}{2}\|f(\mathbf{x}) - \mathbf{y}\|_2^2$. Then for any neuron i,*

$$\mathbb{E}\left[\frac{\partial \ell}{\partial p_i^{(l)}}\right] \geq 0, \tag{4}$$

with strict inequality whenever $p_i^{(l)} > 0$, where the expectation is taken over $\mathbf{V}_{\text{eff}}^{(l)}$.

Proof. Let $i, j \in \{1, \ldots, d_p\}$ index the neurons of layer l, and let $\mathbf{v}_j$ denote the j-th column of $\mathbf{V}_{\text{eff}}^{(l)}$. If $p_i^{(l)} \leq 0$ then $\sigma'(p_i) = 0$ and $\partial \ell / \partial p_i = 0$, so the bound holds. We assume $p_i^{(l)} > 0$ for the remainder, in which case $\sigma'(p_i) = 1$. Writing the matrix–vector product columnwise gives $f(\mathbf{x}) = \sum_{j=1}^{d_p} \mathbf{v}_j\,\sigma(p_j)$, and only the $j = i$ term depends on p_i, so $\partial f / \partial p_i = \mathbf{v}_i\,\sigma'(p_i)$. Applying the chain rule to the MSE loss and substituting,

$$\frac{\partial \ell}{\partial p_i} = \sigma'(p_i) \sum_{j=1}^{d_p} \langle \mathbf{v}_i, \mathbf{v}_j \rangle\,\sigma(p_j) \;-\; \sigma'(p_i)\,\langle \mathbf{v}_i, \mathbf{y} \rangle. \tag{5}$$

By taking the expectation over $\mathbf{V}_{\text{eff}}^{(l)}$, since $\mathbf{y}$ and the p_j are independent of $\mathbf{V}_{\text{eff}}^{(l)}$ at initialization. The label term therefore vanishes,

$$\mathbb{E}[\langle \mathbf{v}_i, \mathbf{y} \rangle] = \langle \mathbb{E}[\mathbf{v}_i], \mathbf{y} \rangle = 0, \tag{6}$$

using $\mathbb{E}[\mathbf{v}_i] = \mathbf{0}$ from Theorem 2.1, leaving

$$\mathbb{E}\left[\frac{\partial \ell}{\partial p_i}\right] = \sum_{j=1}^{d_p} \mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle]\;\sigma(p_j). \tag{7}$$

Each $\sigma(p_j) \geq 0$ since $\sigma = \text{ReLU}$, and $\mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle] \geq 0$ by Theorem 2.1, so every summand is non-negative. The $j = i$ term is strictly positive, since $\mathbb{E}[\|\mathbf{v}_i\|^2] > 0$ and $\sigma(p_i) = p_i > 0$. Thus $\mathbb{E}[\partial \ell / \partial p_i] \geq 0$, with strict inequality whenever $p_i^{(l)} > 0$. □

Remark A.2. When averaging over inputs, a covariance term $\text{Cov}_{\mathbf{x}}(\partial \ell / \partial p_i,\; x_k^{(l-1)})$ arises. However, we further empirically demonstrate that this term oscillates around zero at early iterations, results are present in Figure 3b. Here, $\mathbf{x}$ denotes the input data vector (or layer input $\mathbf{x}^{(l-1)}$), while $\mathbf{p}^{(l)}$ is the pre-activation at layer l, given by $\mathbf{p}^{(l)} = \mathbf{W}^{(l)}\mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}$. The covariance is taken over the empirical/data distribution of $\mathbf{x}$.

A.2. Theorem & proof: positive expected gradient for cross-entropy loss

Theorem A.3 (Cross-entropy loss)**.** *Let $f(\mathbf{x}) = \mathbf{V}_{\text{eff}}^{(l)}\,\sigma(\mathbf{p}^{(l)})$ be as in (2), with $\sigma = \text{ReLU}$ and $\mathbf{V}_{\text{eff}}^{(l)} \in \mathbb{R}^{C \times d_p}$ satisfying Theorem 2.1, where C denotes the number of output classes. Assume the network is at initialization, so that $\mathbf{V}_{\text{eff}}^{(l)}$ is independent of $\mathbf{p}^{(l)}$ and of the one-hot label $\mathbf{y}$. Consider the cross-entropy loss $\ell(f(\mathbf{x}), \mathbf{y}) = -\sum_{c=1}^{C} y_c \log s_c$, where $\mathbf{s} = \text{softmax}(f(\mathbf{x}))$. Then for any neuron i,*

$$\mathbb{E}\left[\frac{\partial \ell}{\partial p_i^{(l)}}\right] \geq 0, \tag{8}$$

(up to higher-order corrections from the softmax linearization), with strict inequality whenever $p_i^{(l)} > 0$, where the expectation is taken over $\mathbf{V}_{\text{eff}}^{(l)}$.

Proof. If $p_i^{(l)} \leq 0$ then $\sigma'(p_i) = 0$ and the bound holds trivially, so assume $p_i^{(l)} > 0$ and $\sigma'(p_i) = 1$. Define

$\mathbf{s} = \mathrm{softmax}(f(\mathbf{x}))$ as the predicted class distribution. The softmax–cross-entropy gradient with respect to the logits is $\partial \ell / \partial f_c = s_c - y_c$. Using the column-wise expansion $f(\mathbf{x}) = \sum_{j=1}^{d_p} \mathbf{v}_j \, \sigma(p_j)$, only the $j = i$ term depends on p_i, so $\partial f / \partial p_i = \mathbf{v}_i$, and the chain rule gives

$$\frac{\partial \ell}{\partial p_i} = \langle \mathbf{v}_i, \mathbf{s} \rangle - \langle \mathbf{v}_i, \mathbf{y} \rangle. \tag{9}$$

Label term. Since $\mathbf{y}$ is independent of $\mathbf{V}_{\text{eff}}^{(l)}$ at initialization,

$$\mathbb{E}[\langle \mathbf{v}_i, \mathbf{y} \rangle] = \langle \mathbb{E}[\mathbf{v}_i], \mathbf{y} \rangle = 0, \tag{10}$$

by Theorem 2.1, exactly as in the MSE case.

Softmax term. The simplex constraint $\mathbf{1}^\top \mathbf{s} = 1$ means $\mathbf{s}$ is not zero-mean, so we cannot apply Theorem 2.1 directly. Decompose $\mathbf{s} = \frac{1}{C}\mathbf{1} + \tilde{\mathbf{s}}$, where $\tilde{\mathbf{s}} = \mathbf{P}\mathbf{s}$ is the centered softmax and $\mathbf{P} = \mathbf{I} - \frac{1}{C}\mathbf{1}\mathbf{1}^\top$ is the centering projection. The constant component vanishes: $\mathbb{E}[\langle \mathbf{v}_i, \frac{1}{C}\mathbf{1} \rangle] = 0$, leaving

$$\mathbb{E}\left[\frac{\partial \ell}{\partial p_i}\right] = \mathbb{E}[\langle \mathbf{v}_i, \tilde{\mathbf{s}} \rangle]\,. \tag{11}$$

We linearize the softmax at the origin. Using $\mathbf{J}_{\mathbf{s}}(\mathbf{0}) = \frac{1}{C}\,\mathbf{P}$,

$$\tilde{\mathbf{s}} = \tfrac{1}{C}\,\mathbf{P}\mathbf{f} + O(\|\mathbf{f}\|^2), \tag{12}$$

and substituting $\mathbf{f} = \sum_j \mathbf{v}_j \, \sigma(p_j)$,

$$\mathbb{E}[\langle \mathbf{v}_i, \tilde{\mathbf{s}} \rangle] = \frac{1}{C} \sum_{j=1}^{d_p} \sigma(p_j)\, \mathbb{E}\big[\mathbf{v}_i^\top \mathbf{P}\, \mathbf{v}_j\big] \;+\; O(\|\mathbf{f}\|^2). \tag{13}$$

Evaluating the projected inner product. Expanding $\mathbf{P} = \mathbf{I} - \frac{1}{C}\mathbf{1}\mathbf{1}^\top$ gives

$$\mathbb{E}\big[\mathbf{v}_i^\top \mathbf{P}\, \mathbf{v}_j\big] = \mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle] - \tfrac{1}{C}\,\mathbb{E}\big[(\mathbf{v}_i^\top \mathbf{1})(\mathbf{v}_j^\top \mathbf{1})\big]\,, \tag{14}$$

so it suffices to evaluate the second term. Write $\mathbf{V}_{\text{eff}}^{(l)} = \mathbf{W}_L\, M$, where $M = \mathbf{D}_{L-1}\mathbf{W}_{L-1} \cdots \mathbf{W}_{l+1}$ is independent of $\mathbf{W}_L$ at initialization. The rows of $\mathbf{W}_L$ are i.i.d. zero-mean, so for any $c \neq c'$ and any i, j,

$$\begin{aligned} &\mathbb{E}[(\mathbf{v}_i)_c\, (\mathbf{v}_j)_{c'}] \\ &= \mathbb{E}_M\big[\mathbb{E}_{\mathbf{W}_L}\big[(\mathbf{W}_L)_{c,:} M_{:,i} \cdot (\mathbf{W}_L)_{c',:} M_{:,j} \,\big|\, M\big]\big] = 0, \end{aligned} \tag{15}$$

using independence and zero-mean of rows of $\mathbf{W}_L$. By the same i.i.d. row structure, $\mathbb{E}[(\mathbf{v}_i)_c (\mathbf{v}_j)_c]$ is constant in c, hence equals $\frac{1}{C}\,\mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle]$. Therefore

$$\begin{aligned} \mathbb{E}\big[(\mathbf{v}_i^\top \mathbf{1})(\mathbf{v}_j^\top \mathbf{1})\big] &= \sum_{c,c'} \mathbb{E}[(\mathbf{v}_i)_c (\mathbf{v}_j)_{c'}] = \\ &\sum_{c} \mathbb{E}[(\mathbf{v}_i)_c (\mathbf{v}_j)_c] = \mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle], \end{aligned} \tag{16}$$

and consequently

$$\begin{aligned} \mathbb{E}\big[\mathbf{v}_i^\top \mathbf{P}\, \mathbf{v}_j\big] = \mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle] - \tfrac{1}{C}\,\mathbb{E}\big[(\mathbf{v}_i^\top \mathbf{1})(\mathbf{v}_j^\top \mathbf{1})\big] = \\ \tfrac{C-1}{C}\,\mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle] \;\geq\; 0, \end{aligned} \tag{17}$$

by Theorem 2.1.

Sign analysis. Substituting (17) into (13),

$$\mathbb{E}\left[\frac{\partial \ell}{\partial p_i}\right] = \frac{C-1}{C^2} \sum_{j=1}^{d_p} \mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle]\, \sigma(p_j) \;+\; O(\|\mathbf{f}\|^2). \tag{18}$$

To leading order, every summand is non-negative (by Theorem 2.1 and ReLU non-negativity) $\frac{C-1}{C^2}\,\mathbb{E}[\|\mathbf{v}_i\|^2]\, p_i^{(l)} \geq 0$. □

Remark A.4. The CE bound parallels Theorem 2.2 with two structural differences. First, the centering projection $\mathbf{P}$ replaces the identity, contributing the factor $(C-1)/C$ that reflects the simplex constraint $\mathbf{1}^\top \mathbf{s} = 1$. Second, the overall coefficient $1/C^2$ reflects the softmax derivative scale at the origin. The leading non-vanishing correction to (12) that survives expectation is $O(\|\mathbf{f}\|^4)$: the cubic term in $\mathbf{V}_{\text{eff}}$ has zero expectation by sign-flip symmetry of $\mathbf{W}_L$ (which leaves the gates $\mathbf{D}_l, \ldots, \mathbf{D}_{L-1}$ invariant since they depend only on $\mathbf{W}_1, \ldots, \mathbf{W}_{L-1}$). The same negative-drift consequence then follows: the gradient with respect to weights $W_{i,k}^{(l)}$ factorizes as $(\partial \ell / \partial p_i)\, \sigma(p_k^{(l-1)})$, both factors non-negative in expectation, so the expected weight update is non-positive.

A.3. Extension to skip connections

Theorem 2.1 does not use the fact that the layers after l form a single chain. It requires only that the map from $\sigma(\mathbf{p}^{(l)})$ to the output is built from zero-mean weights and ReLU gates and ends in the network's last weight matrix $\mathbf{W}_L$. A skip connection adds parallel routes without changing either property, so $\mathbb{E}[\mathbf{v}_i] = \mathbf{0}$ and $\mathbb{E}[\langle \mathbf{v}_i, \mathbf{v}_j \rangle] \geq 0$ carry over unchanged. The part of the output that bypasses layer l, which we denote $\mathbf{r}$, does not depend on $\mathbf{p}^{(l)}$ and simply shifts the effective target from $\mathbf{y}$ to $\mathbf{y} - \mathbf{r}$. It is not independent of $\mathbf{v}_i$, since both pass through $\mathbf{W}_L$, but the resulting inner product $\langle \mathbf{v}_i, \mathbf{r} \rangle$ has the same form as the cross-correlation term that the theorem already bounds.

Empirical support. Skip connections do not change the structure of the argument, and we verify this empirically. Figure 9 compares an MLP with and without a skip connection, and the drift trajectory is qualitatively unchanged. ResNet-18 (Figure 8) and MaxViT are skip-connection architectures and both exhibit the predicted positive gradient bias, and the MLP with a skip connection appears throughout Tables 6 and 7. However, skip connections change the

magnitude of the effect. In Figure 9, adding a skip connection reduces ReLU sparsity from $\sim$0.6 to $\sim$0.4 while increasing gradient magnitudes, and Tables 6 and 7 show that skip connections substantially improve tolerance to imposed sparsity.

A.4. Weight drift beyond initialization

Our theorems hold strictly at initialization. At $t > 0$, the weights are no longer zero-mean and are no longer independent of each other or of the pre-activations. The main insight is that the gradient magnitude decreases once the model predicts correctly, while most of the drift happens while it still predicts incorrectly.

Assuming the covariance term stays orders of magnitude smaller than the weight mean, as we measure in Figure 3b, we write the weights after t steps as $\mathbf{W}_t = \mathbf{W}_0 + \mathbf{\Delta}_t$. The analysis at initialization still applies approximately as long as the weights stay close to their initial values, $\|\mathbf{\Delta}_t\| \ll \sigma(\mathbf{W}_0)$. This ratio is the Z-score in Figure 2. The term at risk is then the label term $-\langle \mathbf{v}_i, \mathbf{y} \rangle$ in Eq. (5). Once downstream weights align with the labels, it reduces the positive gradient. However, gradients are largest early in training, when the loss is large, so main weights shift happens in the first iterations. By the time predictions align with the labels, gradient magnitudes have decayed and the weights can no longer change substantially. This is consistent with the knee and plateau we observe in the drift trajectories (§3). *The drift could only be avoided if the zero-mean initialization already lay close to a good solution, which is not the case in practice.*

B. Extended Results

B.1. Additional experiments on weight drift

This section complements the empirical analysis of Section 3 with three sets of additional experiments: the effect of Top-K sparsity on gradient bias and weight drift, cross-architecture validation of the drift phenomenon, and the stability of drift dynamics when normalization statistics are frozen via *Accumulation Stop*.

B.1.1. Effect of Top-K sparsity on weight drift

Figure 7 shows gradient mean and weight mean trajectories for a three-layer MLP with GELU activations trained under varying levels of Top-K sparsity on random data. As K increases and more activations are retained, the positive gradient bias and corresponding negative weight drift become progressively more pronounced. At $K = 0.10$, where only 10% of activations are retained the gradient mean collapses toward zero across all layers and weight means remain nearly flat throughout training. This indicates that extreme sparsity starves the network of informative gradient signal too few activations survive to produce consistent updates resulting in severely slow convergence. This observation suggests a practical lower bound on usable sparsity.

B.1.2. Cross-architecture validation

Figures 8 and 4 confirm that negative weight drift is not an artifact of the MLP setting but arises consistently across diverse architectures and optimizers. In ResNet-18 (Figure 8), trained with Adam at lr$= 10^{-3}$, weight drift accumulates monotonically across GELU, ReLU, and SiLU, with the covariance term $\mathrm{Cov}(\partial \ell / \partial p, x)$ remaining orders of magnitude smaller than the weight mean. MaxViT-Tiny (Tu et al., 2022) (Figure **??**), trained with AdamW at lr$= 3 \times 10^{-3}$, exhibits the same qualitative pattern: positive gradient bias during early training, monotonically accumulating negative weight drift, and rapid stabilization once gradient magnitudes diminish.

MP-SENet (Lu et al., 2023) (Figure 4), trained with AdamW at lr$= 5 \times 10^{-4}$, presents a partial exception. While ReLU follows the expected pattern of negative weight drift, GELU and SiLU exhibit a positive drift in weight means despite showing positive gradients throughout training.

Taken together, these results confirm that negative weight drift is a robust and broadly observed phenomenon, while acknowledging that architecture-specific factors can influence its magnitude and direction in certain settings.

B.1.3. MLP with and without skip connections

Figure 9 tracks four metrics over 1000 training steps of an MLP trained on real data with Adam (lr$= 10^{-3}$), comparing GELU, SiLU, ReLU, NoisyReLU, and SUGARBSiLU, both without (top) and with (bottom) a skip connection. No normalization layers were used in either configuration, so all observed dynamics arise purely from the interaction between the activation function and the optimizer.

Gradient magnitude and variance. All activations exhibit a sharp transient in both gradient magnitude and standard deviation during the first $\sim$100 steps, followed by rapid decay to a stable regime. NoisyReLU settles at a notably higher gradient variance than the other activations, reflecting the stochastic nature of its outputs, SUGARBSiLU exhibits persistently noisy gradients throughout training, which may hinder stable convergence. Adding a skip connection increases gradient magnitude across all activations, indicating that the residual path sustains stronger gradient flow to earlier layers. *Weight drift.* The Z-score of weight magnitudes increases monotonically for all activations, confirming that negative weight drift accumulates continuously on real data, consistent with the random-data experiments of §2. The drift trajectory exhibits a knee at approximately the same

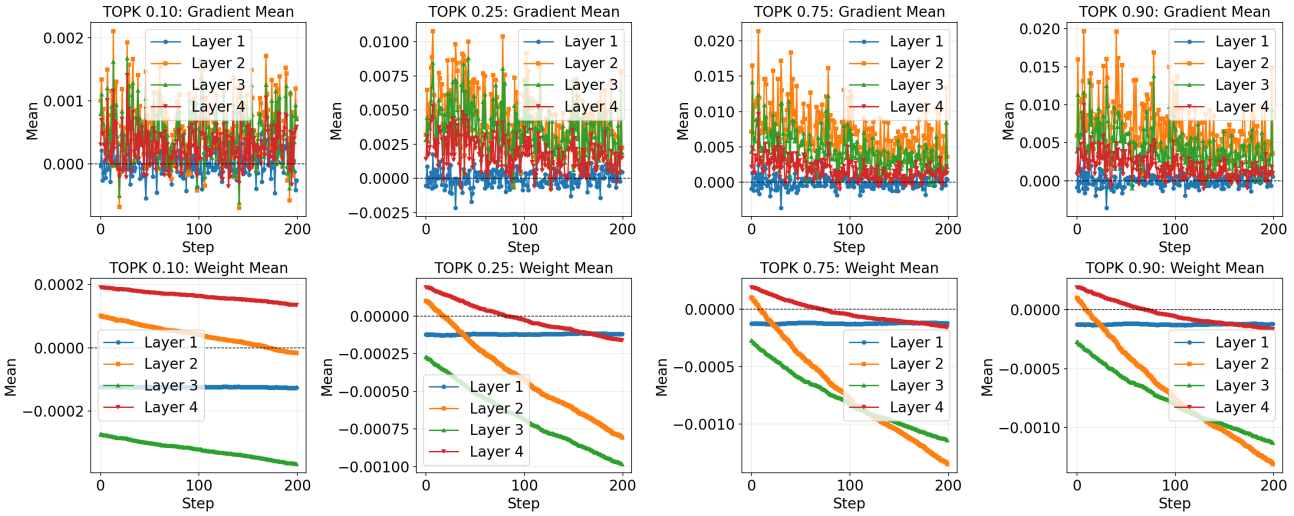


Figure 7. **Effect of Top-K sparsity on weight drift.** A three-layer MLP with GELU activations is trained on random $\{X, Y\}$ pairs sampled from $\mathcal{N}(0, 1)$ under four Top-K retention levels ($K \in \{0.10, 0.25, 0.75, 0.90\}$). Top row: gradient mean per layer. Bottom row: weight mean per layer. Trajectories are averaged across 20 runs (hidden dimension 128, SGD with lr$= 0.001$). As K increases, gradient bias and weight drift grow more pronounced. At $K = 0.10$, gradient signal collapses and weight means remain flat, indicating that extreme sparsity prevents consistent weight updates.

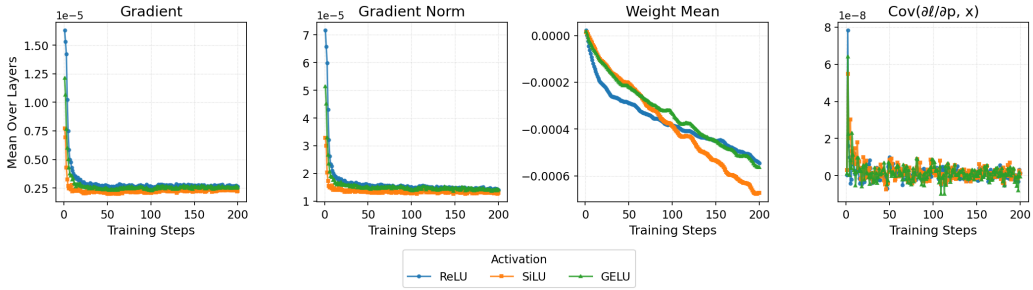


Figure 8. **Weight drift in ResNet-18.** Training dynamics under GELU, ReLU, and SiLU with Adam optimizer (lr$= 10^{-3}$). Four metrics are tracked: (1) average gradient mean, (2) gradient norm, (3) weight mean, and (4) covariance term $\mathrm{Cov}(\partial \ell / \partial p,\, x)$.

step as the gradient transient, drift accumulates rapidly in early training and then slows as the gradient bias diminishes. ReLU and NoisyReLU produce the most pronounced drift, while GELU and SiLU drift more slowly, reflecting their less extreme positive output bias. The skip connection does not qualitatively alter the drift trajectory, suggesting that weight drift is driven primarily by the activation bias rather than by gradient magnitude.

Sparsity. Activation sparsity increases organically throughout training without any explicit regularization. In the plain MLP, ReLU and NoisyReLU reach the highest sparse-neuron fractions ($\sim$0.6), consistent with ReLU's hard thresholding at zero. As weights become more negative, pre-activations shift downward, causing more units to fall below the activation threshold. Adding a skip connection substantially disrupts this mechanism, with ReLU and NoisyReLU sparsity dropping from $\sim$0.6 to $\sim$0.4, at the cost of denser, higher-magnitude gradient flow.

B.1.4. Weight drift when normalization statistics are frozen

Figure 10 shows that freezing normalization statistics with Accumulation Stop (§7) does not disturb weight drift.

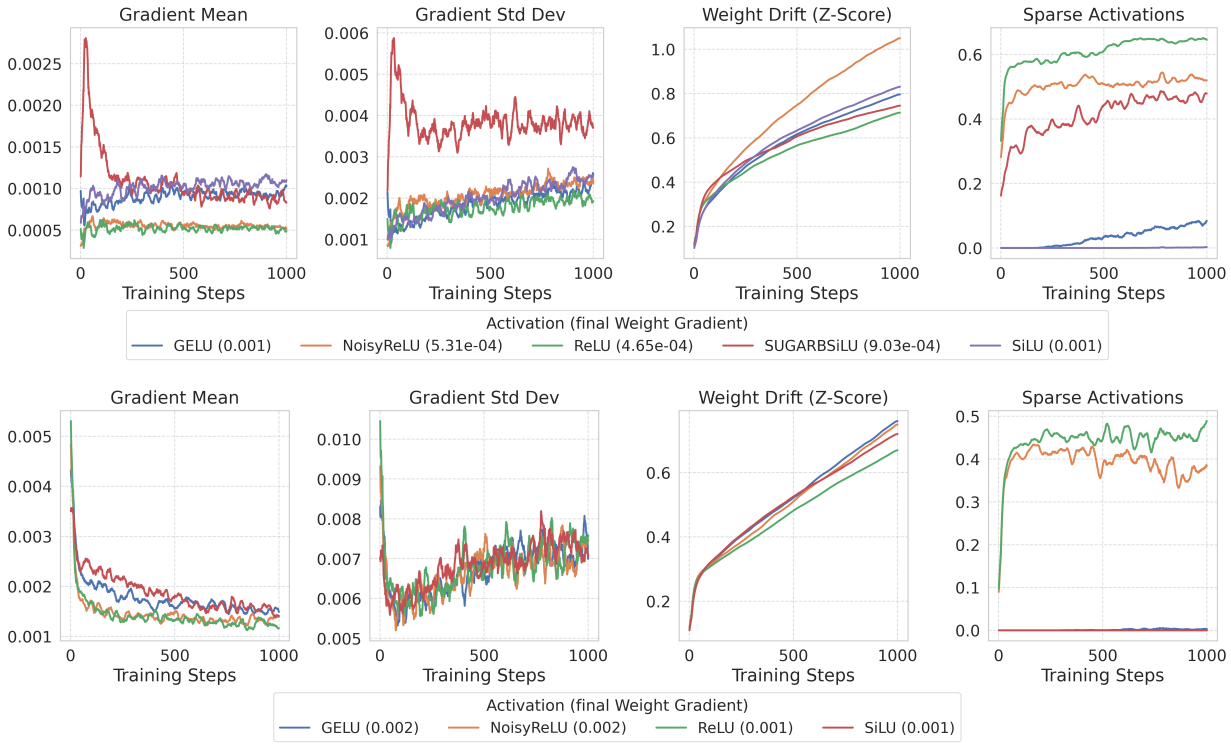


Figure 9. **Training dynamics of an MLP without (top) and with (bottom) skip connection** (see Appendix D.2 for architecture details), under GELU, NoisyReLU, ReLU, SiLU, and SUGARBSiLU with Adam optimizer (lr= 10^{-3}). Four metrics are tracked over 1000 training steps: (1) average gradient magnitude, (2) gradient standard deviation, (3) weight drift (Z-score), and (4) fraction of sparse activations. In the plain MLP, ReLU and NoisyReLU induce the highest sparsity ($\sim$60%) and most pronounced drift; GELU and SiLU remain near-zero sparsity. Adding a skip connection reduces sparsity to $\sim$40% for ReLU and NoisyReLU while substantially increasing gradient magnitudes across all activations. Final average gradient magnitudes are reported in parentheses in the legend.

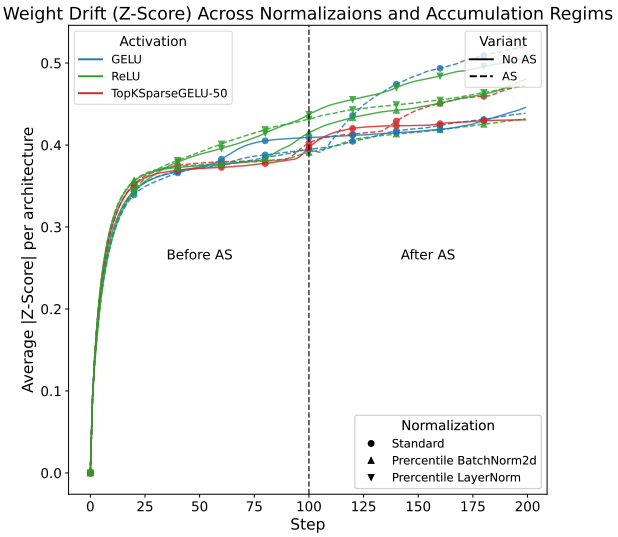


Figure 10. **Weight drift before and after Accumulation Stop.** Average weight drift (mean |Z-score|) for GELU, ReLU, and TopKSparseGELU-50 under Standard, PercentileBN, and PercentileLN normalization strategies. The dashed vertical line marks the AS boundary at step 100. Solid lines: without AS. Dashed lines: with frozen EMA statistics. Trajectories remain continuous and stable after the boundary across all configurations.

B.2. Controllable sparsity

Tables 6 and 7 report accuracy (or test loss for GPT), negative pre-activation fractions, and post-activation sparsity across architectures and sparsity levels under both Percentile Centering and Top-K pruning.

Percentile Centering and Top-K are consistent. Both methods produce closely matching sparsity levels and negative value fractions at equivalent percentile targets, confirming their practical equivalence and supporting the conclusion of §4 that sparsity level rather than mechanism is the dominant predictor of performance.

Architecture determines sparsity tolerance. Figure 11 shows the sparsity–accuracy tradeoff separately for each architecture. ResNet retains 94.3% accuracy at 75% per-activation sparsity and degrades only to 90.4% at 90%, while structured (per-channel) sparsity incurs a heavier penalty, dropping to 74.7% at 95% due to the loss of entire feature channels. ViT accuracy varies by less than 2% across all evaluated levels under both methods, consistent with attention-based aggregation being inherently tolerant to sparse MLP activations. GPT test loss remains nearly flat

from 11% to 77% sparsity (increasing by less than 0.002 nats) and degrades only marginally to 3.286 at 90%, suggesting the accuracy cliff for transformer language models lies well above the range evaluated here. By contrast, the plain MLP collapses to near-chance performance ($\sim$10%) beyond 80% sparsity, while adding skip connections allows it to retain 32.7% accuracy even at 90%, confirming that residual paths provide a bypass mechanism under aggressive sparsification.

Negative value fractions confirm weight drift. Pre-activation negative fractions are substantial across all models even before any sparsity is explicitly induced, corroborating the drift findings of §2. ViT exhibits particularly high negative fractions ($\sim$0.77–0.86) at low sparsity levels, suggesting weight drift is especially pronounced in attention-based architectures. The fact that per-channel and per-activation ResNet results share similar negative fractions at equivalent sparsity levels confirms that the negative bias originates from weight drift rather than from the sparsification mechanism.

Sparsification in generative models. Figure 12 extends the sparsity analysis to image generation with DiT-S/2 (Peebles and Xie, 2023) on ImageNet. All configurations converge to a plausible overall composition within 50K steps and improve progressively to 300K steps. Differences emerge primarily in high-frequency detail. The baseline ReLU configuration produces noticeably blurred backgrounds and lacks fine-grained structure, while GELU with Top-K sparsity consistently preserves sharp structural details and textural realism. This suggests that Top-K masking of a smooth non-monotonic activation such as GELU better protects the high-frequency spatial gradients needed for clear visual boundaries, offering a practical recommendation for sparsity-aware generative model design.

B.3. GPT-nano

Tables 9–10 collectively support the hypothesis that activation spikes originate in the MLP block rather than in attention layers. Weight standard deviations remain stable across all layer types, including in spike layers L1–L4 , ruling out weight instability as a cause. The down-projection consistently receives large, one-sided inputs (max $\approx$ 39–42, min $\approx$ 0) in both the full model and spike layers, a direct consequence of the activation function zeroing negatives and amplifying large positives. By contrast, attention layers exhibit symmetric, moderate input ranges, arguing against any causal role of the attention mechanism or Softmax in spike formation.

Origin of spikes. Down-projection weights in spike layers are not larger than elsewhere in the network (max $|W| = 2.299$ vs. 2.858 overall, Table 10), and up-projection inputs are symmetric and moderate ($\approx \pm 7$). The asymmetry therefore appears exactly where the activation function acts, which indicates that spikes arise because a few token representations align with particular rows of the up-projection. Our measurements do not explain why this alignment concentrates on specific tokens and layers. Sun et al. (2026) attribute spikes in SwiGLU models to quadratic amplification from the gate–up product, but our non-gated MLP shows that a gate is not required. Barbero et al. (2025) argue that the first or "special" tokens take on a functional role that limits over-mixing, which is consistent with the concentration of spikes in early layers.

Seed variability of squared activations. Table 8 reports two additional runs of ReLU^2 alongside the first run from Table 3. Spikes are present in all three runs, but their magnitude varies considerably: the Layer 2 range drops from 1055.0 in the first run to 248.7 and 178.3 in the additional runs. Even the smaller values remain 4–6$\times$ above ReLU and GELU. Test loss also varies, from 3.251 to 3.233 (mean 3.240). In Run 2 it coincides with GELU^2 (3.233) despite a pronounced spike. ReLU$^2_{\text{clip50}}$, in contrast, reaches 3.236 with negligible variance across three runs. We conclude that spike amplification by squared activations is consistent across seeds but not always damaging. Clipping removes it and makes training outcomes more predictable without degrading loss. The advantage of squared over non-squared activations (Figure 13) holds across all runs.

B.4. Squared activation functions in ViT

To test whether the observed effects extend beyond GPT-like models, we evaluate squared activations on a larger Vision Transformer (MaxViT). Table 11 reports aggregated accuracy and sparsity metrics. Unlike in GPT-nano, squared activations do not improve over their base counterparts. ReLU^2 reaches 62.38% compared to 69.57% for ReLU, and GELU^2 reaches 60.65% compared to 70.30% for GELU. Clipping at 50 recovers most of this gap (69.25% for ReLU$^2_{\text{clip50}}$ and 69.63% for GELU$^2_{\text{clip50}}$), but no squared or clipped variant outperforms plain GELU. Pretrained vision models show no distinct early-layer spikes (§C), although MaxViT-T develops large values in later stages (up to 17.31). Squaring can amplify such values, which may explain why clipping helps. Squared activations may therefore benefit mainly autoregressive language models, or vision models may require a different clipping threshold.

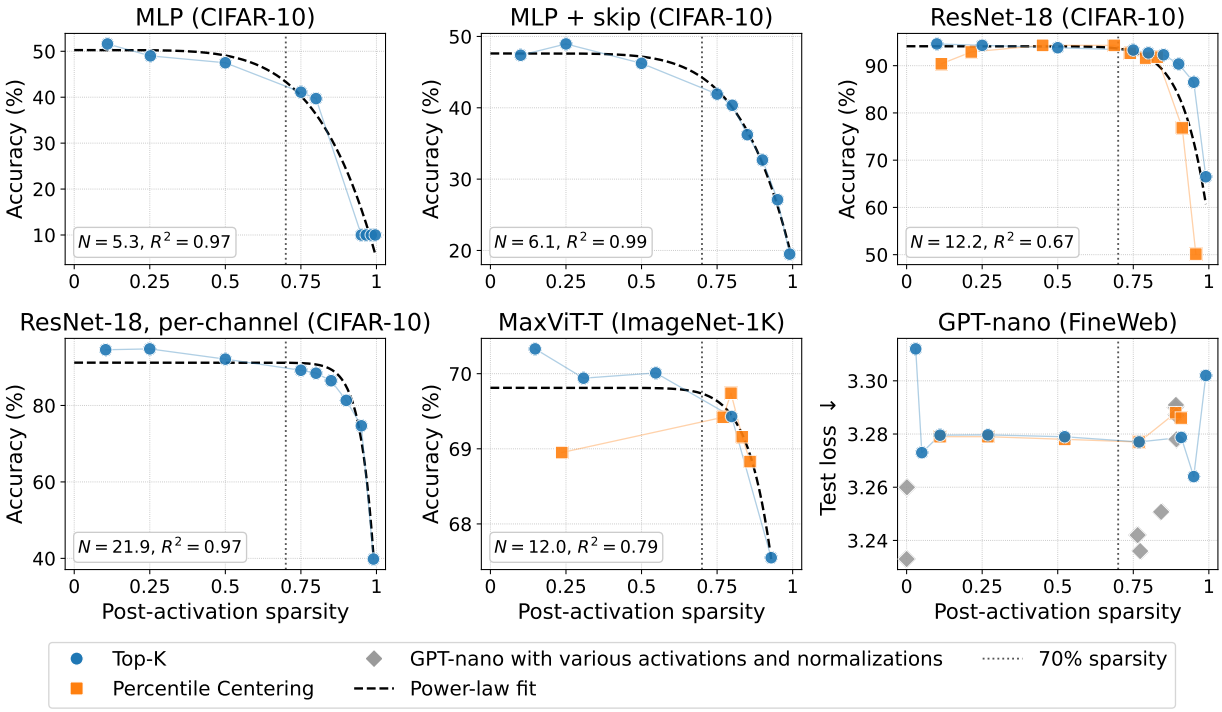


Figure 11. **Sparsity–accuracy tradeoff per architecture.** The same 79 configurations as in Figure 5, shown as raw accuracy (test loss for GPT-nano) rather than as a pooled normalized score. In the classification models, Top-K is applied to GELU and Percentile Centering to ReLU. Dashed curves show a per-panel fit $A - B\,s^N$. No fit is shown for GPT-nano, whose loss stays flat up to $s \approx 0.9$. Gray diamonds are GPT-nano runs with other activation functions or normalization layers at their natural sparsity. The dotted line marks 70% sparsity.

Table 6. Results when sparsity is induced with percentile shift or TOP-K. Negative values and sparsity for GPT are computed only after down-projection and before up-projection layers correspondingly. Results are presented as: (**Accuracy or test loss for GPT**) / (**Negative Values before activation function**) / (**Sparsity after activation function**).

Model	**P10**	**P25**	**P50**	**P75**	**P90**
	*Percentile Centering - **ReLU** is used to induce sparsity*				
MLP	45% / 0.123 / 0.121	48.3% / 0.273 / 0.272	47.0% / 0.502 / 0.500	43.3% / 0.747 / 0.750	24.6% / 0.889 / 0.880
MLP + SKIP	41.75 / 0.137 / 0.132	44.64 / 0.269 / 0.271	45.11 / 0.503 / 0.504	47.58 / 0.737 / 0.737	41.12 / 0.876 / 0.873
ResNet	90.4% / 0.115 / 0.115	92.9% / 0.215 / 0.214	94.3% / 0.451 / 0.450	94.3% / 0.687 / 0.687	91.8% / 0.849 / 0.829
MaxViT	68.95% / 0.761 / 0.236	69.42% / 0.770 / 0.770	69.74% / 0.796 / 0.796	69.16% / 0.833 / 0.833	68.83% / 0.859 / 0.859

	*TOP-K Pruning - **GELU** is used for all models*				
Model	**K90**	**K75**	**K50**	**K25**	**K10**
MLP	51.6% / 0.610 / 0.110	49.0% / 0.626 / 0.252	47.5% / 0.661 / 0.500	41.1% / 0.595 / 0.750	10.0% / 0.500 / 0.966
MLP + SKIP	47.37 / 0.480 / 0.100	48.95 / 0.485 / 0.250	46.23 / 0.501 / 0.500	41.92 / 0.545 / 0.750	32.68 / 0.603 / 0.900
ResNet	94.6% / 0.469 / 0.100	94.3% / 0.471 / 0.250	93.8% / 0.447 / 0.500	93.3% / 0.436 / 0.750	90.35% / 0.370 / 0.900
MaxViT	70.33% / 0.788 / 0.148	69.94% / 0.771 / 0.308	70.01% / 0.771 / 0.547	69.43% / 0.772 / 0.798	67.55% / 0.786 / 0.929
GPT	3.279 / 0.1106 / 0.1106	3.279 / 0.2689 / 0.2689	3.278 / 0.5231 / 0.5231	3.277 / 0.7692 / 0.7692	3.286 / 0.9087 / 0.9087

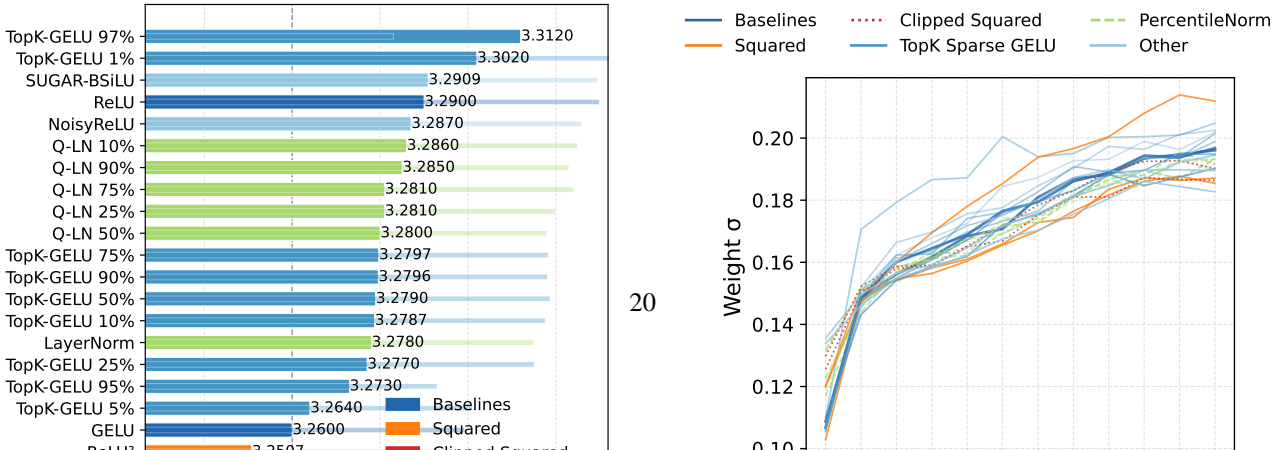


Table 7. Extended results when sparsity is induced with Percentile Centering or TOP-K across additional sparsity levels. Results are presented as: (**Accuracy**) / (**Negative Values before activation function**) / (**Sparsity after activation function**).

Model	Q10	Q25	Q50	Q75	Q80	Q85	Q90	Q95
	*Percentile Centering with **ReLU** is used to induce sparsity*							
ResNet	90.4% / 0.115 / 0.115	92.9% / 0.215 / 0.214	94.3% / 0.451 / 0.450	94.3% / 0.687 / 0.687	92.6% / 0.740 / 0.740	91.6% / 0.791 / 0.791	91.8% / 0.849 / 0.829	76.8% / 0.912 / 0.912

	*TOP-K Pruning – **GELU** is used for all models and BN is applied in ResNet-18*							
Model	**K90**	**K75**	**K50**	**K25**	**K20**	**K15**	**K10**	**K5**
ResNet	94.6% / 0.469 / 0.100	94.3% / 0.471 / 0.250	93.8% / 0.447 / 0.500	93.3% / 0.436 / 0.750	92.7% / 0.406 / 0.800	92.3% / 0.388 / 0.850	90.4% / 0.370 / 0.900	86.5% / 0.367 / 0.950
ResNet Struct.	94.6% / 0.600 / 0.104	94.8% / 0.578 / 0.250	92.1% / 0.543 / 0.500	89.2% / 0.511 / 0.750	88.4% / 0.506 / 0.800	86.5% / 0.519 / 0.850	81.3% / 0.514 / 0.900	74.7% / 0.512 / 0.950
MLP	51.6% / 0.610 / 0.110	49.0% / 0.626 / 0.252	47.5% / 0.661 / 0.500	41.1% / 0.595 / 0.750	39.7% / 0.490 / 0.800	10.0% / 0.550 / 0.950	10.0% / 0.500 / 0.966	10.0% / 0.501 / 0.983
MLP + SKIP	47.4% / 0.480 / 0.100	49.0% / 0.250 / 0.250	46.2% / 0.501 / 0.500	41.9% / 0.545 / 0.750	40.4% / 0.563 / 0.800	36.2% / 0.587 / 0.850	32.7% / 0.603 / 0.900	27.1% / 0.576 / 0.950

Table 8. Loss and per-layer down-projection input range (max−min) for three ReLU^2 seeds on GPT-nano, compared to single runs of ReLU and GELU. ReLU$^2_{\text{clip50}}$ reaches a test loss of 3.236 with negligible variance across three seeds.

	ReLU	GELU	ReLU^2 Run 1	ReLU^2 Run 2	ReLU^2 Run 3
Loss ↓	3.290	3.260	3.251	3.233	3.236
Layer 1	16.2	12.4	108.5	33.0	29.5
Layer 2	42.2	42.7	1055.0	248.7	178.3
Layer 3	67.5	28.5	260.9	96.3	164.5
Layer 4	17.8	10.2	49.8	131.3	59.4
Layer 5	13.1	16.0	65.3	24.5	32.0

C. Activation Spikes in Pretrained Models

To test whether activation spikes are specific to our GPT-nano setup, to transformers in general, or to a particular modality, we extend the spike analysis to six publicly available pretrained models. They span different architectures (transformers, a hybrid convolution–attention model, and CNNs), modalities (text and images), and training objectives. For each model we measure the down-projection input range (max−min) per layer, as in Table 3 (Tables 12–14).

Findings. Distinct spikes in early-to-middle layers appear only in the language models. Both GPT-2 and BERT-base show a pronounced spike at layer 2 (62.95 and 51.73, compared to ∼3–17 in neighboring layers), and BERT-base shows a second, larger spike at layer 10 (114.46). ViT-B/16 shows only a moderate peak at layer 5 (15.43). MaxViT-T shows high values that grow in later stages (up to 17.31), and these could be amplified by squared activations, which is consistent with the benefit of clipping we observe for MaxViT (Table 11). ResNet-50 shows even higher values (up to 39.80 in layer3, block 4), whereas ResNet-18 remains small throughout. Overall, these results suggest that early-layer spikes are a property of language models rather than of the transformer MLP block itself. This supports the hypothesis that they originate from tokens with a "functional" role, such as the first or special tokens (Barbero et al., 2025).

D. Technical Details

This appendix provides full implementation details for all experiments in the paper. §D.2 covers MLP experiments, §D.3 ResNet-18, §D.4 DiT-S/2, §D.5 MaxViT-T, and §D.6 GPT-nano. Shared dataset configurations are described in §D.1. Activation statistics measurement protocols are defined in §D.1.

D.1. Shared dataset and training configuration

CIFAR-10. Unless otherwise specified, classification experiments use CIFAR-10 with per-channel normalization ($\mu = [0.4914, 0.4822, 0.4465]$, $\sigma = [0.2470, 0.2435, 0.2616]$), a training batch size of 128, and evaluation on an unaugmented test set with batch size 100. All linear and convolutional layers omit bias terms.

ImageNet-1K. Generative and large-scale classification experiments use ImageNet-1K. Images are center-cropped and resized to 256×256 (DiT) or 224×224 (MaxViT) with standard per-channel normalization.

Activation statistics. Two statistics are measured on a fixed test batch throughout all experiments:

- **Negative percentage**: fraction of pre-activation values (linear layer outputs) that are negative, $\text{NegPct}_l = \frac{1}{N_l} \sum_i \mathbb{I}(x_{l,i} < 0)$.
- **Sparsity**: fraction of post-activation outputs with magnitude below $\epsilon = 10^{-7}$, averaged across the full test set.

Normalization variants. Where normalization is varied, three strategies are evaluated in pre-norm configuration (Norm → Linear → Act):

- **No normalization**: baseline without any normalization.
- **RMSNorm** (Zhang and Sennrich, 2019): normalizes by root mean square without centering, $\text{RMSNorm}(x) = x \,/\, \sqrt{\frac{1}{D} \sum_i x_i^2 + \epsilon} \odot w$, where $w \in \mathbb{R}^D$ is a learned scale and $\epsilon = 10^{-6}$.

Table 9. Weight standard deviations averaged over 23 runs, for all layers and spike layers (L1–L4). All layer types show lower and tighter STDs in spike layers, indicating weight stability precisely where activation spikes occur.

	All Layers			Layers 1–4		
Layer Type	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**
`attn.out_proj`	0.1919	0.0208	0.3073	0.1668	0.0208	0.2349
`mlp.up_projection`	0.1616	0.0208	0.1819	0.1662	0.0208	0.1811
`mlp.down_projection`	0.1583	0.0104	0.2017	0.1468	0.0104	0.1727
`attn.qkv_projection`	0.1481	0.0208	0.2284	0.1353	0.0208	0.1797

Table 10. Average maximum and minimum weight and input values over 23 runs, for all layers and spike layers (L1–L4). The down-projection consistently receives large, one-sided inputs (max ≈39–42, min ≈0), while attention layers exhibit symmetric and moderate ranges.

	All Layers				Layers 1–4			
Layer Type	**Max W**	**Min W**	**Max In**	**Min In**	**Max W**	**Min W**	**Max In**	**Min In**
`attn.out_proj`	1.139	−1.143	19.33	−19.23	0.934	−0.961	15.60	−15.67
`attn.qkv_projection`	1.189	−1.185	9.05	−7.87	0.930	−0.925	7.59	−6.47
`mlp.down_projection`	**2.858**	**−2.781**	**39.01**	≈0	**2.299**	**−1.957**	**42.00**	≈0
`mlp.up_projection`	1.245	−1.264	7.47	−6.65	1.136	−1.136	5.30	−4.86

- **LayerNorm** (Ba et al., 2016): centers and scales activations, $\text{LayerNorm}(x) = (x-\mu_x) \,/\, \sqrt{\sigma_x^2+\epsilon} \odot w + b$, with learned affine parameters $w, b \in \mathbb{R}^D$.

Each normalization variant is evaluated with six activation functions: ReLU, GELU, SiLU, ReLU^2, NoisyReLU, and SUGARBSiLU. Higher-order and clipped variants (GELU^2, ReLU$^2_{\text{clip15}}$, ReLU$^2_{\text{clip50}}$) are additionally evaluated in GPT-nano experiments (§D.6).

D.2. MLP experiments

We use a 5-layer fully connected MLP without normalization layers: an input projection followed by 5 repeated blocks of the form $\text{Linear}(d, d) \to [\text{Act}]$, and a final linear layer.

CIFAR-10 (cross-entropy). Input dimension 3072, hidden dimension $d = 1024$, output dimension 10. Trained for 5 epochs with Adam (lr $= 10^{-3}$, weight decay 10^{-4}). Trajectories averaged across 5 runs (Figure 3b).

Random data (MSE). To isolate the effect of activation functions on weight drift independently of the data distribution, models are trained on random $\{X, Y\}$ pairs sampled from $\mathcal{N}(0, 1)$ with MSE loss. Hidden dimension $d = 128$, trained for 5 epochs with SGD (lr $= 0.01$). Trajectories averaged across 5 runs (Figure 3a).

D.3. ResNet-18 experiments

All ResNet-18 experiments use the standard architecture with BasicBlock units and a 64–128–256–512 channel progression, trained on CIFAR-10 following Section D.1.

Activation function comparison. Six activation functions (ReLU, GELU, SiLU, ReLU^2, NoisyReLU, SUGARBSiLU) are evaluated with standard Batch Normalization. Training uses SGD with Nesterov momentum (0.9), weight decay 5×10^{-4}, and a learning rate schedule of 5-epoch linear warmup (from 10^{-4} to 10^{-3}) followed by cosine annealing.

Top-K sparsified GELU. Explicit sparsity is applied via a Top-K mask after GELU:

$$\text{TopK-GELU}(x) = \text{GELU}(x) \odot M,$$
$$M_{i,j} = \mathbb{I}(|\text{GELU}(x)_{i,j}| \in \text{top}_K(|\text{GELU}(x)|)) \,.$$

Five retention levels are evaluated: $K \in \{0.01, 0.05, 0.10, 0.15, 0.20\}$. Training uses SGD with Nesterov momentum (0.9), weight decay 5×10^{-4}, initial learning rate 5×10^{-2}, and 5-epoch linear warmup followed by cosine annealing, for 50 epochs.

D.4. DiT-S/2 configuration

Architecture. We adopt DiT-Small (DiT-S/2) (Peebles and Xie, 2023), consisting of 12 transformer blocks ($D = 384$, 6 heads) with adaptive LayerNorm (adaLN-Zero) for

Table 11. Aggregated results on MaxViT (accuracy ↑ / sparsity).

	ReLU	GELU	NoisyReLU	SUGARBSiLU	ReLU^2	$\text{ReLU}^2_{\text{clip50}}$	GELU^2	$\text{GELU}^2_{\text{clip50}}$
Acc / Sparse	69.57 / 0.798	70.30 / 0.010	68.46 / 0.172	51.49 / 0.829	62.38 / 0.756	69.25 / 0.669	60.65 / 0.017	69.63 / 0.003

Table 12. Down-projection input range by layer for language models. Spikes in bold.

Layer	GPT-2	BERT-base
0	10.91	9.00
1	16.98	9.78
2	**62.95**	**51.73**
3	11.05	8.21
4	6.92	10.99
5	3.90	11.29
6	3.16	8.25
7	4.92	6.60
8	7.01	6.73
9	6.22	28.10
10	5.18	**114.46**
11	8.70	7.45

timestep and class conditioning. All linear layers omit bias terms.

VAE latent space. The VAE encoder from Stable Diffusion v1.5 projects $256 \times 256 \times 3$ RGB images into a $32 \times 32 \times 4$ latent space. Latents are scaled by 0.18215 during training and generation.

Diffusion training and sampling. The model is trained for ∼300K steps (batch size 256) to minimize the ℓ_2 denoising objective:

$$\mathcal{L} = \mathbb{E}_{t,x_0,\epsilon}\left[\|\hat{\epsilon}_\theta(x_t, t, y) - \epsilon\|_2^2\right], \tag{19}$$

where $t \sim \text{Uniform}(0, 249)$ and noise is added via a squared cosine schedule. Generation uses the EMA model with Classifier-Free Guidance ($s = 1.5$) and 250 sampling steps.

Accumulation Stop for normalization statistics. To decouple training dynamics from fluctuating batch statistics while maintaining efficiency, we implement an Accumulation Stop (AS) mechanism using an Exponential Moving Average (EMA):

$$\bar{v}_i = \gamma \bar{v}_{i-1} + (1-\gamma)v_i, \quad \gamma = 0.9999. \tag{20}$$

After $K = 50{,}000$ steps, the running statistic $\bar{v}_K$ is frozen for all subsequent iterations. Note that Percentile BatchNorm with AS effectively functions as static Percentile Batch Symmetry after step K, analogously to standard BatchNorm with AS transitioning to static normalization.

Table 13. Down-projection input range by layer for ViT-B/16 and MaxViT-T. For MaxViT-T, each MLP sublayer follows either window or grid attention.

Layer	ViT-B/16	MaxViT-T	MaxViT sublayer
0	5.83	2.22	Stage 1, Block 1, Window attn
1	4.39	2.71	Stage 1, Block 1, Grid attn
2	3.16	2.54	Stage 1, Block 2, Window attn
3	3.50	2.91	Stage 1, Block 2, Grid attn
4	4.75	3.34	Stage 2, Block 1, Window attn
5	**15.43**	2.66	Stage 2, Block 1, Grid attn
6	4.26	3.23	Stage 2, Block 2, Window attn
7	4.03	3.76	Stage 2, Block 2, Grid attn
8	3.08	5.12	Stage 3, Block 1, Window attn
9	3.92	3.43	Stage 3, Block 1, Grid attn
10	7.27	5.79	Stage 3, Block 2, Window attn
11	2.10	5.17	Stage 3, Block 2, Grid attn
12	–	8.77	Stage 3, Block 3, Window attn
13	–	10.07	Stage 3, Block 3, Grid attn
14	–	11.18	Stage 4, Block 1, Window attn
15	–	10.90	Stage 4, Block 1, Grid attn
16	–	15.11	Stage 4, Block 2, Window attn
17	–	15.11	Stage 4, Block 2, Grid attn
18	–	11.56	Stage 4, Block 3, Window attn
19	–	5.53	Stage 4, Block 3, Grid attn
20	–	12.99	Stage 4, Block 4, Window attn
21	–	**17.31**	Stage 4, Block 4, Grid attn

D.5. MaxViT-T configuration

Architecture and training. MaxViT-T (Tu et al., 2022) uses a 2–2–5–2 block structure with interleaved local and global attention. Training runs for 60 epochs with AdamW (lr $= 6 \times 10^{-4}$, weight decay 0.1) under a OneCycleLR schedule. Data augmentation follows the original RandAugment protocol ($N = 2$, $M = 9$).

Latency benchmarking protocol. To isolate the per-forward-pass overhead of different activation and normalization configurations, we employ a four-phase measurement protocol (100 steps each):

1. **Warm-up**: GPU synchronization and cache stabilization.
2. **Active Train**: standard training with dynamic normalization statistic updates.
3. **Converged Train**: training with Accumulation Stop (frozen EMA statistics).
4. **Inference**: evaluation mode, no gradients.

Table 14. Down-projection input range by block for ResNet-18 and ResNet-50.

Index	ResNet-18	Stage	ResNet-50	Stage
0	2.08	layer1, block 1	17.78	layer1, block 1
1	3.02	layer1, block 2	18.75	layer1, block 2
2	3.12	layer2, block 1	12.34	layer1, block 3
3	1.94	layer2, block 2	22.87	layer2, block 1
4	1.57	layer3, block 1	8.29	layer2, block 2
5	1.11	layer3, block 2	11.48	layer2, block 3
6	1.06	layer4, block 1	8.74	layer2, block 4
7	0.68	layer4, block 2	7.81	layer3, block 1
8	–	–	6.09	layer3, block 2
9	–	–	11.76	layer3, block 3
10	–	–	**39.80**	layer3, block 4
11	–	–	29.05	layer3, block 5
12	–	–	18.52	layer3, block 6
13	–	–	11.43	layer4, block 1
14	–	–	17.70	layer4, block 2
15	–	–	14.33	layer4, block 3

All throughput figures are measured in bfloat16 mixed precision with statistics computed channel-wise and averaged over the batch. The baseline uses PyTorch's optimized kernel; our AS implementation uses plain PyTorch, so reported throughput recovery represents a conservative lower bound.

D.6. GPT-nano training details

Architecture. We train a GPT-nano model (124M parameters) based on the `nanoGPT` implementation (Karpathy, 2022), with 12 layers, 12 attention heads, embedding dimension 768, and vocabulary size 50,257.

Training configuration. The model is trained on FineWeb using the Muon optimizer for linear layers and Adam for all other parameters. Key hyperparameters: batch size 16, sequence length 2048, 16 gradient accumulation steps (totaling 7,168 effective iterations), zero weight decay, optimizer betas $[0.9, 0.95]$. The learning rate schedule consists of 2,000 warmup and 2,028 warmdown iterations.

Evaluation. Test evaluation is performed every 1,024 iterations on 1,048,576 tokens with an evaluation batch size of 32. No hyperparameters are selected on the test split. Training time per model is approximately 8 GPU hours. A total of 23 configurations are evaluated, covering activation functions (ReLU, GELU, NoisyReLU, SUGARBSiLU, ReLU^2, GELU^2, ReLU$^2_{\text{clip15}}$, ReLU$^2_{\text{clip50}}$), sparsification strategies (Top-K GELU at multiple retention levels), and normalization variants (LayerNorm, QuantileLayerNorm at multiple percentiles).

D.7. Activation function implementations

All custom activation functions are implemented as `nn.Module` subclasses and are drop-in replacements for standard PyTorch activations. We describe each below alongside the key implementation decisions.

ReLU^2. ReLU^2 is defined as $f(x) = \max(0, x)^2$ and is implemented as a simple composition of ReLU and a squaring operation:

```
class ReLUSquared(nn.Module):
    def forward(self, x):
        return F.relu(x).square()
```

No learnable parameters are introduced. The squaring is applied *after* the ReLU threshold, so the zero-output region is preserved exactly and sparsity behavior is identical to ReLU at the activation boundary.

NoisyReLU. NoisyReLU (Gulcehre et al., 2016) addresses the dying neuron problem by injecting input-dependent noise into negative pre-activations during training, while reverting to standard ReLU at inference to preserve sparsity. Formally:

$$f(x) = \alpha \cdot \text{ReLU}(x) + (1 - \alpha) \cdot x + \epsilon(x), \tag{21}$$

where $\alpha \in [0, 1]$ interpolates between ReLU and the identity, and the noise term is:

$$\epsilon(x) = \mathbb{I}[x \le 0] \cdot \sigma \cdot |\mathcal{N}(0, 1)|,$$
$$\sigma = c \cdot (\text{sigmoid}(v \cdot \delta) - 0.5)^2 .$$

with $\delta = \max(0, -x)$ the magnitude of the negative input and v a learnable scalar parameter. The noise magnitude σ is therefore input-dependent and learned, vanishing as $x \to 0^-$ and growing for more negative inputs. At test time the function reduces exactly to ReLU:

```
class NoisyReLU(nn.Module):
    def __init__(self, alpha=1.0, c=1.0):
        super().__init__()
        self.alpha = alpha
        self.c = c
        self.p = nn.Parameter(torch.randn
    (1))
    def forward(self, x):
        if not self.training:
            return F.relu(x)
        delta = torch.where(
            x < 0, -x, torch.zeros_like(x))
        sigma = self.c * (
            torch.sigmoid(self.p * delta) -
     0.5
        ).square()
        noise = torch.where(
            x < 0,
            sigma * torch.abs(torch.
    randn_like(x)),
            torch.zeros_like(x))
        return (self.alpha * F.relu(x)
                + (1 - self.alpha) * x +
    noise)
```

In all experiments we use $\alpha = 1.0$ and $c = 1.0$ with half-normal noise, i.e. $\epsilon = |\mathcal{N}(0, 1)|$, so the identity branch vanishes and only the noise term acts on negative pre-activations.

SUGARBSiLU. SUGARBSiLU (Horuz et al., 2025) is a surrogate gradient method that decouples the forward and backward passes. The forward pass applies standard ReLU, preserving hard sparsity at every step, while the backward pass substitutes the zero gradient of dead neurons with the derivative of B-SiLU — a smooth surrogate that maintains non-zero gradient signal for negative pre-activations:

$$f_{\text{fwd}}(x) = \text{ReLU}(x), \tag{22}$$

$$\left.\frac{\partial f}{\partial x}\right|_{\text{bwd}} = \sigma(x) + (x + \alpha) \cdot \sigma(x)(1 - \sigma(x)), \tag{23}$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid function and $\alpha = 1.67$ is a fixed shift parameter. This is implemented via a custom `torch.autograd.Function` that applies ReLU in `forward()` and the B-SiLU derivative in `backward()`, requiring no changes to the model architecture or inference pipeline:

```
ALPHA = 1.67

class SUGARBSiLUFunction(torch.autograd.
    Function):
    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)
        return F.relu(x)

    @staticmethod
    def backward(ctx, grad_output):
        x, = ctx.saved_tensors
        sigma = torch.sigmoid(x)
        surrogate = sigma + (x + ALPHA) *
    sigma * (1.0 - sigma)
        return grad_output * surrogate

class SUGARBSiLU(nn.Module):
    def forward(self, x):
        return SUGARBSiLUFunction.apply(x)
```

The surrogate gradient is strictly positive for all x, ensuring that no neuron becomes permanently dead during training regardless of how negative its pre-activation becomes. At inference, the `forward()` path is used exclusively, so the model produces the same sparse activations as a standard ReLU network.

E. Extended Limitations and Discussion

This appendix expands on the limitations and open questions summarized in §8. We address each of the six points in turn, following the numbering used in the main text.

E.1. Limitations

(1) Theoretical assumptions hold strictly only at initialization. Theorems 2.1, 2.2, and 2.3 rely on three conditions: zero-mean i.i.d. weights, independence between $V_{\text{eff}}^{(l)}$ and the pre-activations $p^{(l)}$, and (in the cross-entropy case) softmax linearization around $f = 0$ with $\mathcal{O}(\|f\|^2)$ corrections. As training progresses, weights deviate from the i.i.d. regime, downstream layers become correlated with upstream activations through shared gradient updates, and the linearization error grows as $\|f\|$ moves away from zero. We argue empirically that the drift is largest precisely during the first iterations when these assumptions are best satisfied, and that the resulting negative offset persists once the gradient bias diminishes. The same argument shows that violations of the assumptions weaken the bound quantitatively but do not flip its sign in the early phase where drift accumulates fastest. Appendix A.4 extends this argument beyond initialization.

(2) Formal proof covers ReLU only. The argument in Theorem 2.1 relies on the survival-conditioning property of ReLU gates: each D_l is a binary diagonal matrix that selects active neurons, and conditioning on survival induces a positive correlation along the input direction. Smooth activations such as GELU and SiLU do not admit a clean binary gating decomposition, so extending the proof requires a continuous analogue of survival conditioning, which is beyond the scope of this paper. Empirically, the same drift pattern is observed across GELU, SiLU, NoisyReLU, and SUGARBSiLU except one model.

(3) Empirical scope and clipping thresholds. Classification experiments are conducted on CIFAR-10 and ImageNet-1K, and language modeling uses a single dataset (FineWeb) with a small autoregressive model (GPT-nano, 124M parameters). Whether the ∼70% sparsity cliff and the benefit of clipped squared activations transfer unchanged to frontier-scale LLMs (1B+ parameters, multi-trillion-token training) remains open. The profiling of pretrained GPT-2 and BERT (Appendix C) and the sparsity levels reported for 7B-scale models in prior work are consistent with our findings, but direct training experiments at larger scale remain future work. The cliff itself is fitted from N=79 configurations and is consistent across the architectures we tested, but a denser sweep at intermediate scales would help characterize how the cliff position depends on capacity, depth, and training duration. Clipping thresholds (15 and 50) for ReLU^2 were selected empirically on GPT-nano based on the spike magnitudes observed. The ViT results suggest that the optimal threshold might be architecture-dependent, a fixed numerical threshold tuned on one architecture does not transfer directly. Adaptive or layer-wise clipping schedules, possibly tied to running activation statistics, are a natural extension

we did not explore.

(4) Throughput gains are conservative lower bounds. Accumulation Stop is evaluated on architectures with fixed-size inputs (DiT-S/2, MaxViT), where the warm-up statistics provide a stable and representative summary of the full training distribution. Its applicability to autoregressive models is more limited. Reported throughput in Table 4 compares a naive PyTorch implementation of percentile centering against PyTorch's optimized LayerNorm/BatchNorm kernels. Because our implementation is unoptimized while the baseline is fused, the recovery to near-baseline throughput already represents a conservative lower bound. A fused CUDA kernel for percentile centering, analogous to existing fused LayerNorm kernels, would likely yield further improvement and could make **Accumulation Stop a net throughput win** even during the warm-up phase.

(5) Transformer robustness to sparsification. ViT and GPT-nano tolerate sparsity levels far beyond the cliff observed in MLPs, with GPT-nano test loss remaining nearly flat up to $s \approx 0.91$. Comparable levels of sparsity in text transformers have been reported in prior work that exploits rather than explains the phenomenon (Song et al., 2024c; Liu et al., 2024; Song et al., 2024a; Lee et al., 2024). Robustness to sparsity is not limited to transformers. MaxViT accuracy drops only from 70% to 67% as activation sparsity increases from 10% to 90%, and ResNet drops from 94% to 90% over the same range. Skip connections improve sparsity tolerance in MLPs (from collapse at 85% to 32.7% accuracy retention at 90%), but this gap alone does not account for the much greater robustness of attention-based architectures. Several factors likely contribute, including attention as a content-based aggregation mechanism that can route around sparsified MLP outputs, overparameterization of the feed-forward blocks relative to the attention pathway, and the residual structure that makes the MLP block an additive correction rather than a serial bottleneck. The residual path preserves the signal, so the next layer effectively receives $\text{sparse}(\mathbf{x}) + \text{residual}$, and updating only $\sim$10% of units in each block appears to be sufficient. This hypothesis could be tested by increasing the number of residual blocks at a fixed sparsity level. A complementary direction is to track geometric properties of intermediate representations, such as effective rank or intrinsic dimensionality per layer, across sparsity levels. A sharp drop in intrinsic dimensionality coinciding with the accuracy cliff would indicate that sparsity directly limits representational capacity. Note that a representation of dimension N can in principle retain rank N even under very high sparsity, for example with a single active unit per position across different inputs. If dimensionality is instead preserved despite the accuracy drop, the cause is more likely poor gradient dynamics, such as too few active units to sustain a useful learning signal. Disentangling these factors requires controlled ablations beyond what this paper covers.

(6) DiT-S/2 improvements with Percentile LayerNorm. On DiT-S/2, GELU with Percentile LayerNorm at the 50th percentile achieves FID 48.21 and IS 31.41, both improving over the centered LayerNorm baseline (FID 49.40, IS 29.85). We do not characterize this effect mechanistically. There is one possible explanation - gradient flow via quantile and mean are entirely different, in the case of quintile gradient flows only through one quantile value.

E.2. Additional discussion

This subsection collects further discussion points that arose during the review process.

Scope: mean-centering normalization. Mean-centering normalization (BatchNorm, LayerNorm) suppresses weight drift by removing the positive pre-activation bias that drives it. ResNet with BatchNorm is the one configuration in Table 1 without substantial negative pre-activations. Our findings therefore apply most directly to architectures without centering. This class is not small: it includes most recent LLMs, which use RMSNorm (Appendix 10.2). Moreover, activation sparsity has mainly been studied and exploited in transformer LLMs (Liu et al., 2023; Song et al., 2024c), whereas we show that the mechanism producing it extends beyond transformers (MLP, ResNet, MP-SENet).

Does the sparsity cliff shift with scale? Activation sparsity in large models is consistent with our GPT-nano results. LLMs at the 7B scale can be pushed to $\sim$90% sparsity: Song et al. (2024c) report 89.3% on LLaMA2-7B, and Song et al. (2024a) report $\sim$90% on Mistral-7B. More broadly, prior work reports sparsity of 50–90% in models such as Llama-2, Llama-3, Mistral 7B–70B, and MiniCPM-1B (Liu et al., 2023; Song et al., 2024c; Liu et al., 2024; Song et al., 2024a; Lee et al., 2024). Mixture-of-Experts models can be read as structured activation sparsity, and the numbers coincide again: Mixtral (Jiang et al., 2024) activates 2 of 8 experts per layer, i.e., 75% sparsity, and recent MoEs route more aggressively still. We therefore do not observe a clear progression of sparsity tolerance with model size. We expect the position of the cliff to depend on the task and objective, the architecture (e.g., the presence of skip connections), and the amount of training data at least as much as on capacity. For example, it is difficult to imagine that diffusion-based image generation models tolerate similarly high sparsity.

Why the first layer does not drift. The expected update to $W_{ik}^{(l)}$ factorizes as $\mathbb{E}[(\partial \ell / \partial p_i^{(l)}) \cdot x_k^{(l-1)}]$ (Remark in Appendix A.2). The first factor is non-negative in expectation by Theorems 2.2 and 2.3, so the sign of the expected

update is determined by the sign of the layer input. For the first layer, the input is the data, which is often mean-centered. The product then has no consistent sign and no drift accumulates. For every subsequent layer, the input is a post-activation and hence non-negative (ReLU) or positively biased (GELU, SiLU). The expected update is then non-positive and drift accumulates. In a transformer, each MLP block has a single hidden layer, but its down-projection receives a post-activation input and therefore satisfies exactly this condition, and the block is repeated in every layer.

Would a drift-free activation perform better? Symmetric, zero-centered activations such as tanh do not induce a negative drift, yet they perform considerably worse in practice. More importantly, our sparsity analysis suggests that negatively shifted weights do not hurt performance on their own. The main cause of degradation is a high sparsity ratio. Percentile Centering lets us shift pre-activations back into the positive range, and we observe no performance improvement from doing so (§4). We therefore have no evidence that a drift-free activation function would help, although the question remains open.

Does clipping also help GELU^2? Squared activations other than ReLU^2 also produce large values, and clipping helps them as well. On MaxViT, GELU$^2_{\text{clip50}}$ reaches 69.63% accuracy compared to 60.65% for unclipped GELU^2, and ReLU$^2_{\text{clip50}}$ reaches 69.25% compared to 62.38% for ReLU^2 (Table 11).

Spikes and special tokens in vision transformers. If early-layer spikes in language models originate from tokens with a functional role, a natural question is whether similar spikes localize on the explicit functional tokens of vision transformers: the [CLS] token, or register tokens, which Darcet et al. (2024) introduce to absorb the high-norm artifacts that otherwise appear in background patches. Profiling MLP inputs per token in ViTs with and without registers would directly test this hypothesis. We leave this to future work.

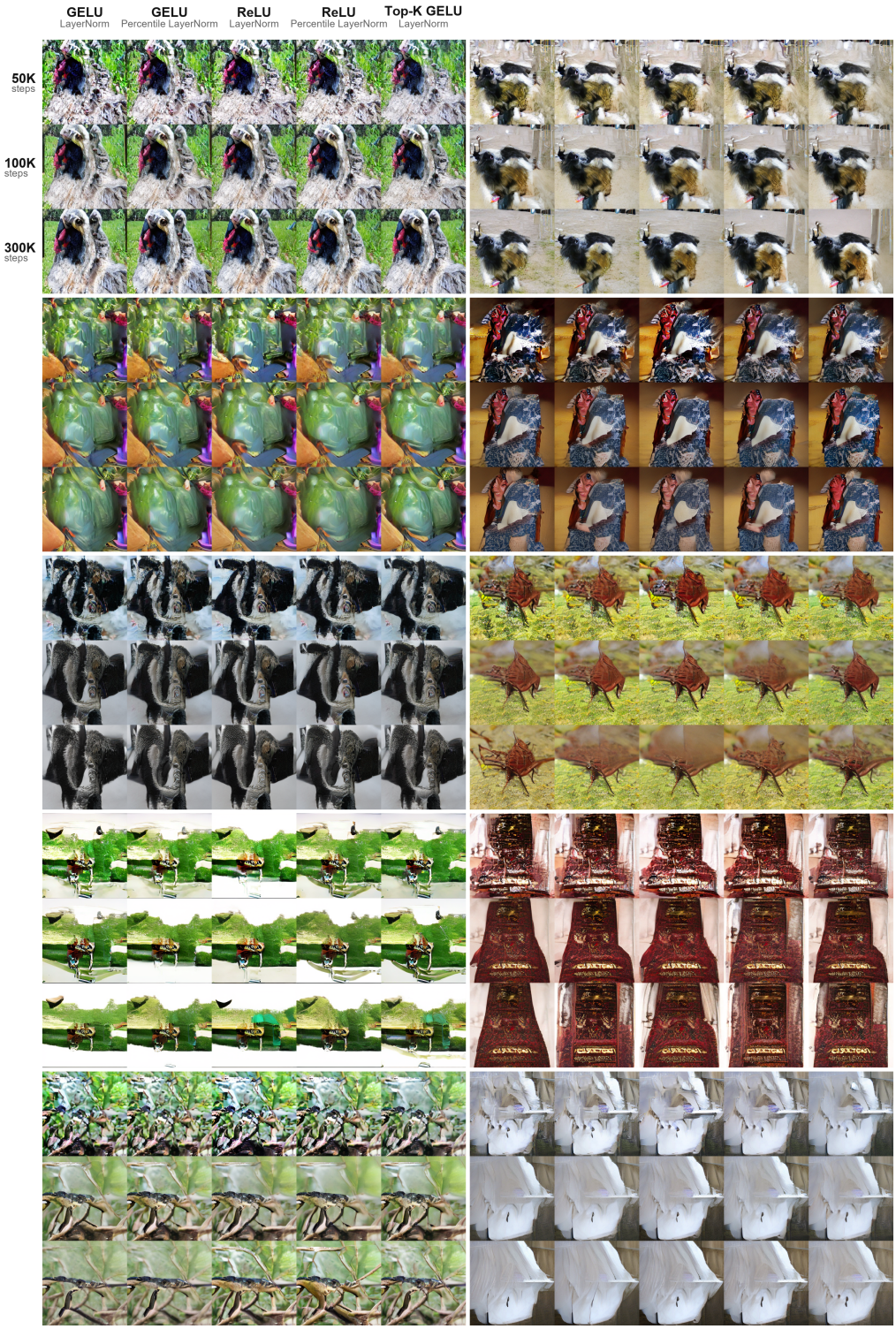


Figure 12. **DiT-S/2 generated samples across configurations at 50K, 100K, and 300K training steps.** Columns correspond to GELU + LayerNorm, GELU + Percentile LayerNorm, ReLU + LayerNorm, ReLU + Percentile LayerNorm, and Top-K GELU + LayerNorm. All configurations produce coherent compositions by 50K steps. High-frequency differences become apparent at later checkpoints: the baseline ReLU configuration produces blurred backgrounds and lacks fine detail, while Top-K GELU preserves complex structural features such as grasshopper limbs and environmental textures.