SinGLU: Sinusoidal Gated Linear Units Improve Classification Accuracy of Small Vision Transformers

Anonymous authors
Paper under double-blind review

Abstract

Gated Linear Unit (GLU) variants such as SwiGLU are now widely used in modern Transformers. However, the GLU functions explored in the recent literature represent only a small fraction of the possible GLU design space. Starting from a mathematically complete enumeration of all zeroth-, first-, and second-order GLU formulas, we conduct a controlled study on ViT-Tiny across CIFAR-10, CIFAR-100, SVHN and ImageNet-64, instantiating each GLU formula with Sigmoid, Tanh and Sin activations. Under identical training recipes and equal parameter counts, our proposed first order GLU variant **SinGLU** consistently outperforms SwiGLU, the de-facto standard in contemporary Transformers. Inference latency differs by <0.1% on an NVIDIA A100 GPU, confirming cost parity. All Code and model weights will be released upon publication.

1 Introduction

Vision Transformers (Dosovitskiy et al., 2020) have seen widespread, and rapid adoption across the field of computer vision. This is owed largely to their improved performance over previous machine learning architectures like Convolutional Neural Networks (CNNs) when large volumes of data are available. However, the Transformer architecture is surprisingly simple. There are effectively only two components in a Transformer encoder. These are the Multi-headed Self Attention block (MHSA), and the two-layer Multi-layer Perceptron block (MLP). Further, the only sources of nonlinearity are the repeated Layer Normalizations (Ba et al., 2016), the SoftMax function within the MHSA, and the nonlinear first layer of the MLP.

The original Vision Transformer (Dosovitskiy et al., 2020) was proposed with a GELU (Hendrycks & Gimpel, 2016) activation as the nonlinearity in the first layer of the MLP. Recently Shazeer (2020) introduced a number of Gated Linear Unit (GLU) (Dauphin et al., 2017) type functions which outperformed GELU when used in the MLP portions of a language Transformer. Shazeer's SwiGLU function has since been used in several state-of-the-art large language and Vision Transformer models such as PaLM (Chowdhery et al., 2023), LLaMA (Touvron et al., 2023), and DINOv2 (Oquab et al., 2023). In Shazeer's paper SwiGLU was presented as a GLU-type layer with a Swish (Ramachandran et al., 2017) activation function.

The characteristics of a 'GLU-type' function are not strictly defined. Both the original GLU and Shazeer's variants use two weight matrices and bias vectors for each layer, producing two groups of hidden neurons. One group of hidden neurons has their outputs passed through a nonlinear activation function while the other does not. The outputs of the two groups are then multiplied together element-wise, giving the layer's final output. In this way the activated nonlinear neurons can be seen as 'gating' the linear neurons, hence the name 'Gated Linear Unit'.

In both GLU and Shazeer's GLU variants the weight matrices and bias vectors for the activated and non-activated neurons are distinct and different. However, if we allow the weight matrices and bias vectors to be either different or identical then we can actually view the Swish 'activation' (Ramachandran et al., 2017) inside SwiGLU as itself being a GLU-type function with a *Sigmoid* activation. In this way SwiGLU can be seen as a *second order* GLU-type function with a *Sigmoid* activation. It also becomes clear that other second order variants are possible by changing which weight matrices are identical and which are unique. Visual

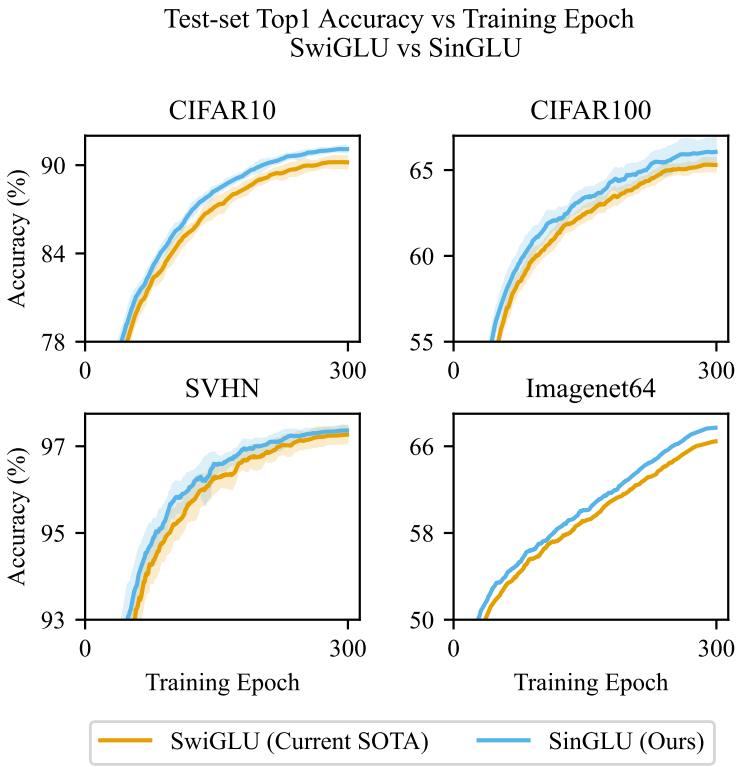

Figure 1: Test-set top-1 classification accuracy for ViT-Tiny models using either SwiGLU or our SinGLU functions in the MLP blocks. SinGLU consistently outperforms SwiGLU on all datasets tested. For CIFAR10, CIFAR100 and SVHN we show the mean of three training runs, with $\pm 1\sigma$ shown as a shaded area around the mean. For readability, a rolling mean has been applied to all plots with a window of 10 epochs.

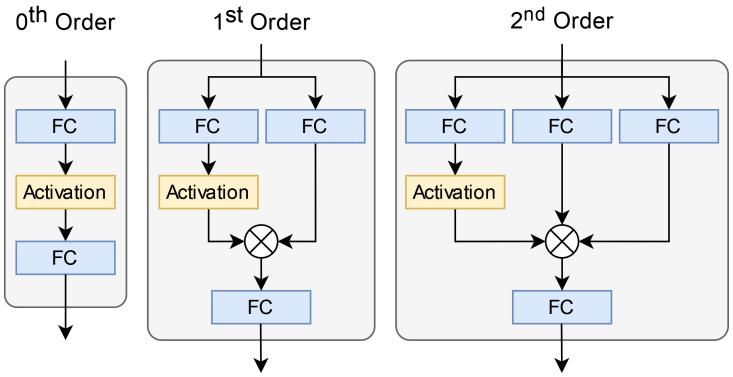

Figure 2: Block diagrams of Transformer MLP blocks with 0th, 1st, and 2nd order GLU-type functions. Here 'FC' refers to a fully connected layer of neurons, 'Activation' represents an element-wise activation function, and $\otimes$ is an element-wise multiplication.

comparisons between Transformer MLP blocks utilizing zeroth, first, and second order GLU-type functions may be seen in Figure 2.

We therefore seek to perform a broad search across the space of possible GLU-type layer functions in the hopes of discovering functions which outperform SwiGLU on common tasks. However, the space of possible

GLU-type functions is infinite, as one can always add more weight matrices and bias vectors. We therefore restrict the search space to only functions with 1 to 3 weight matrices, and only allowing one group of hidden neurons to be passed through an activation function. This reduces the number of possible 0th to 2nd order functions for a given activation to 7.

We choose the activations *Sigmoid*, *Tanh*, and *Sin* as the functions to investigate in this study. *Sigmoid* and *Tanh* were chosen because they are possibly the most ubiquitous continuous activation functions in neural networks. *Sin* was chosen because it is not a common activation function, however, for input values near zero sin closely approximates *Tanh*.

With 7 GLU-type layer functions and 3 activations, we create 21 ViT-Tiny (Wu et al., 2022) models, and assess their performance on a variety of image classification datasets.

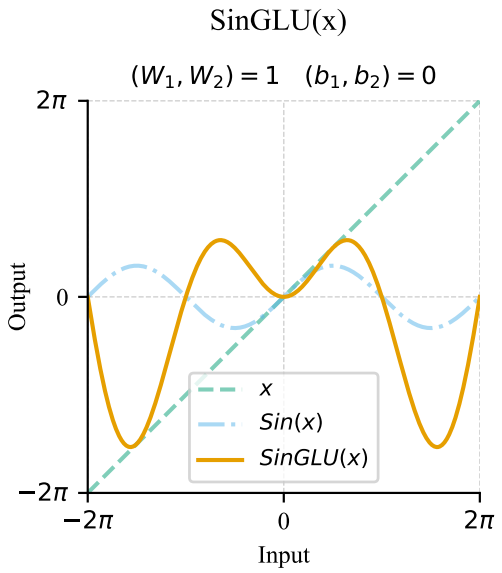

Figure 3: Input-output plot for our SinGLU layer function when all weights are 1 and all biases are 0 ie. $Sin(x) \otimes x$. Dashed green lines shows $output = input$. Dash-dotted blue shows $output = Sin(input)$. SinGLU is effectively the product of these two.

Through our experiments we identified a 1st order GLU-type layer with a *Sin* activation as the most performant when the problem space is sampled densely enough to prevent over-fitting. We choose to call this function "SinGLU" in line with the naming conventions present in the literature. However, future work is required to identify why exactly SinGLU appears to generalise better than the other functions tested. It is possible that sinusoidal activations cover a larger representable hypothesis space than approximately piecewise linear functions. Similarly, it is possible that sinusoidal functions in combination with the near-zero initialisation of weights and online normalisation increases the models' soft inductive bias towards low-frequency solutions.

SinGLU is given by the equation:

$$\text{SinGLU}(x, W_1, W_2, b_1, b_2) = \text{Sin}(xW_1 + b_1) \otimes (xW_2 + b_2) \tag{1}$$

where x is the input feature map of shape $[batch \times inputdim]$, W_1 and W_2 are learnable weight matrices of shape $[hiddendim \times inputdim]$, b_1 and b_2 are learnable bias vectors of length $[hiddendim]$, and $\otimes$ is an element-wise multiplication. The first weight and bias effectively control the frequency modulation and phase offset of the function's output. The second controls the amplitude. The input-output plot for a single neuron with weights equal to 1 and biases equal to 0 is shown in Figure 3. We show the effect of varying the parameters W_1, W_2, b_1, and b_2 in Figure 4.

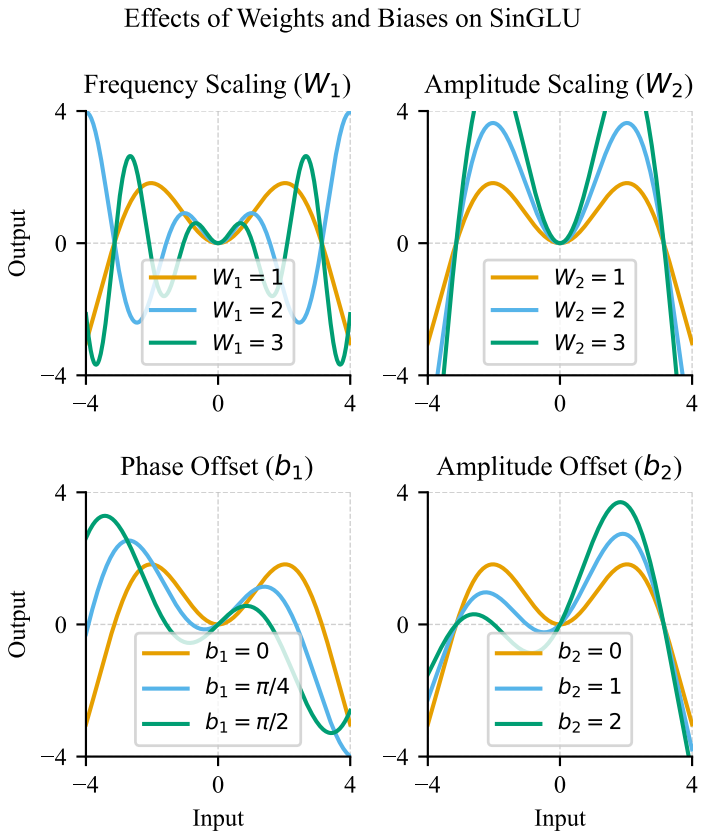

Figure 4: Input-output plots for our SinGLU layer function $SinGLU = Sin(xW_1 + b_1) \otimes (xW_2 + b_2)$ when varying the parameters W_1, W_2, b_1, and b_2 respectively. Starting with weights equal to 1 and biases equal to zero, we increase each parameter in turn while holding the others constant. (a) and (b) show the effect of increasing W_1 or W_2 respectively. (c) and (d) show the effect of increasing b_1 or b_2 respectively. W_1 and b_1 are seen to control the frequency modulation and phase offset, while W_2 and b_2 control the amplitude scale and amplitude offset.

The main contributions of this work are:

- Systematic enumeration of all 0th to 2nd order GLU-type layer functions for any give activation.
- Discovery of several novel layer functions which outperformed SwiGLU.
- Evidence that periodic activations may be used in Vision Transformers without introducing training difficulties, given modern training recipes.

2 Background and Related Work

2.1 Vision Transformers

The Vision Transformer (ViT) was introduced by Dosovitskiy et al. (2020). It built on the Transformer architecture proposed by Vaswani et al. (2017) for natural language processing. The Transformer encoder used in ViT consists of two blocks which are augmented with skip connections. These blocks are a channel-wise MLP (Multi-Layer Perceptron), and a sequence-wise Multi-Headed Self Attention (MHSA).

Analysis by Park & Kim (2022) has shown that the MLP and MHSA blocks perform opposite, complementary roles. The MLP acts as a high-pass filter and amplifier, increasing the variance of feature maps. The MHSA

block acts as a low pass filter, aggregating feature maps and reducing variance. This causes the MHSA to have a smoothing effect on the loss landscapes of ViTs, reducing the hessian maximum eigenvalues, on sufficiently large datasets. Park has also shown that the MHSA block makes ViTs far less likely to overfit on small datasets. However, it is important to note that the lack of strong inductive biases in ViTs can result in a less convex loss landscapes for small datasets. Tuli et al. (2021) have shown that the aggregating property of the MHSA substantially increases the network's bias towards shape rather than texture, when compared to ResNets (He et al., 2016). This bias aligns more closely with human perception and improves error consistency and robustness to domain shift, as measured by accuracy on Stylised Image Net (SIN) (Geirhos et al., 2018).

2.2 Gated Linear Units

The original gated linear unit (GLU) was introduced by Dauphin et al. (2017) for use in convolutional language modeling. The output of a GLU layer is the Hadamard, or element-wise product of the outputs of two fully connected groups of neurons, one of which has a *Sigmoid* activation function. Given an input feature map x, *Sigmoid* function σ, learnable weight matrices W_1, W_2, and learnable bias vectors b_1, b_2 the output of a GLU layer may be defined as:

$$GLU(x, W_1, W_2, b_1, b_2) = (xW_1 + b_1) \otimes (xW_2 + b_2) \tag{2}$$

Shazeer (2020) later introduced GLU variants of several popular activation functions, by replacing the *Sigmoid* with other activation functions such as Swish (Ramachandran et al., 2017) and RELU (Fukushima, 1969).

2.3 Online Normalization

We suspect that online normalization techniques may be beneficial when training neurons with periodic activation functions. Online normalization standardizes the data distribution before applying a learned linear affine.

Batch Normalization is an online normalization technique proposed by Ioffe & Szegedy (2015). It normalizes data batch-wise to have a mean of 0 and a variance of 1. A linear affine is then applied using learnable weight and bias vectors with dimensions equal to the number of feature channels. Batch Normalization was initially proposed as a means of reducing internal covariate shift. However, Santurkar et al. (2018) has suggested that the improved training performance may instead be due smoothing of the network loss landscape. This in turn results in more stable gradients and improved robustness to training hyperparameters.

Instance Normalization (Ulyanov et al., 2016) is an alternative technique first proposed for style transfer tasks. Similar to Batch Normalization, the input is normalized to have a mean of 0 and a variance of 1, with a weight and bias then applied. However, instead of computing the mean and variance cross the batch, Instance Normalization computes these values across the height and width of a single channel of each data instance separately. This is beneficial when drawing batch elements from several distributions, or when batch statistics are not reliable, for example when some elements in the batch have been zero-padded.

Group Normalization (Wu & He, 2018) is similar to batch or instance norm. However, instead of normalizing across the batch or across a single data instance, group norm first divides the feature channels into groups and then normalizes across these groups. This method is advantageous when the mini-batch size is small, as it can provide a balance between the two previous techniques.

Layer-Normalization (Ba et al., 2016) is the technique used in Transformers. Rather than computing the mean and variance across the batch as in Batch Norm, or across groups of features as in Group Norm, Layer-Norm acts across all feature channels of a single data instance. This is equivalent to performing Group Normalization with a single group. This is particularly fitting for Transformer architectures because the fully connected linear layers in the MHSA and MLP layers all act across the channel dimension. However, there is some debate about whether Layer-Norm should be applied at the start or the end of the MLP and MHSA blocks. Layer-Norm applied at the start of the blocks is typically called Pre-Layer-Normalization, as

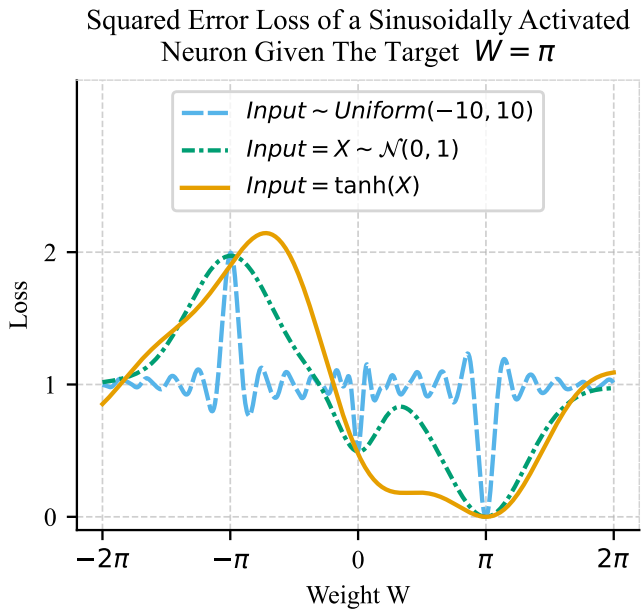

Figure 5: Loss landscapes of a single sinusoidally activated neuron with the function $Output = Sin(W \times Input)$ and the target $W = \pi$. Here the input $X \in R^{10,000}$ is drawn from either a uniform distribution representing an unnormalized input, or a normal distribution. Also shown is the loss landscape if the normalized input is squashed with a Tanh function as is empirically observed in Layer Norm (Zhu et al., 2025).

opposed to Post-Layer-Normalization which is applied at the end of the block (Xiong et al., 2020). In this paper we investigate changes to models that use Pre-Layer-Normalization.

Recent work by Zhu et al. (2025) has shown that the presence of large outliers in the network activations causes Layer Norm to act as a saturating Tanh like function. This effect is more pronounced deeper in the network, and further into training, where large outlier values are more likely to develop.

In summary, while the specific mechanics of these online normalization techniques vary, their shared objective is to stabilize training by bounding and standardizing the distribution of hidden activations. We suspect that the use of online normalisation before the SinGLU layer is likely essential, however future work is required to evaluate this. Normalization likely prevents the pre-activations from widely traversing multiple periods of the sine function, effectively smoothing the local loss landscape and allowing the gating mechanism in SinGLU to converge stably without the need for specialized initialization schemes.

2.4 The Viability of Periodic Activation Functions in Transformers

Periodic activation functions have seen limited adoption in machine learning outside specific areas such as Fourier Neural Networks (Ngom & Marin, 2021) and Implicit Neural Representations (Sitzmann et al., 2020). Previous work (Parascandolo et al., 2016) has suggested that this lack of widespread use is due to the training difficulties introduced by the non-convex loss landscapes of neurons with periodic activations. Periodic functions are non-quasi-convex and non-monotonic. As such, neurons with periodic activations have loss landscapes with infinitely many minima. This can be seen in Figure 5 which shows the loss landscapes of a single sinusoidally activated neuron with inputs sampled from either uniform or normal distributions.

In Figure 5 A global minimum can be seen in both plots at $W = \pi$, when the target function $sin(\pi x)$ has been matched exactly. A global-maximum is also seen at $W = -\pi$, when the neuron's output has the same frequency as the target but is exactly 180 degrees out of phase. It may also be seen that when sampling inputs from large ranges or with large standard deviations (blue lines in Figure 5) a significant local-minimum

develops around the point where the weight is zero. This is problematic because under typical initialization schemes such as Kaiming (He et al., 2015), Xavier (Glorot & Bengio, 2010), or truncated-normal, most weights at training-start will be near-zero. As such, the target $W = \pi$ will be unlearnable. However, reducing the range or the standard deviation of inputs significantly smooths the loss landscapes, removing the local minima around $W = 0$, making the target learnable (red lines in Figure 5).

This analysis suggests that sinusoidally activated neurons are likely to benefit from online normalization of their inputs, L2 regularization or weight decay to discourage the development of large-value target weights, small batch sizes, and data augmentation techniques which are known to smooth loss landscapes such as label smoothing (Müller et al., 2019). We believe that Vision Transformers utilizing typical training recipes meet these criteria.

3 Methods

3.1 GLU-Type Layer Functions

Shazeer (2020) defined SwiGLU as:

$$SwiGLU(x, W, V, b, c, \beta) = Swish_\beta(xW + b) \otimes (xV + c) \tag{3}$$

Where x is the input feature map, W and V are weight matrices, b and c are bias vectors, $\otimes$ is a Hadamard or element-wise product, and β is scalar parameter which is set to 1 in all of Shazeer's experiments. Setting β to 1 makes Swish equivalent to the SiLU activation of Hendrycks & Gimpel (2016), which they defined as:

$$SiLU(x) = \sigma(x) \otimes x \tag{4}$$

Where σ is a *Sigmoid* function. Therefore, we may rewrite the SwiGLU equation as:

$$SwiGLU(x, W, V, b, c) = \sigma(xW + b) \otimes (xW + b) \otimes (xV + c) \tag{5}$$

We can simplify this notation further if we let:

$$x_n = xW_n + b_n \tag{6}$$

The output Z of a SwiGLU layer can then be written as:

$$Z = \sigma(x_1) \otimes x_1 \otimes x_2 \tag{7}$$

If we determine the order of a GLU-type layer by the number of x_n terms, we can consider SwiGLU to be a second-order layer function with a *Sigmoid* activation. Furthermore, it is evident that SwiGLU represents just one of several potential second-order layers with a *Sigmoid* nonlinearity.

3.2 Enumerating all Possible 0th to 2nd Order GLU-type Layers

We seek to test as wide a range of layer functions as possible in order to identify the most performant. As discussed earlier the space of all possible GLU-type layer functions is infinite. To restrict our search to an extensive but tractable range we explore one to three x_n terms, allow the activation ϕ to act on only one x_n term, and allow only Hadamard product operations between x_n terms. This range encompasses regular non-GLU layer functions, the original GLU, and the second order function SwiGLU. However, this still leaves a wide range of functions unexplored, which should be investigated in future work.

Under these constraints the list of functionally unique 0th to 2nd order GLU-type layers with activation function ϕ is:

$$
\begin{array}{ll}
\underline{Order} & \underline{LayerFunction} \\
0^{th} & Z_1 = \phi(x_1) \\
1^{st} & Z_2 = \phi(x_1) \otimes x_1 \\
1^{st} & Z_3 = \phi(x_1) \otimes x_2 \\
2^{nd} & Z_4 = \phi(x_1) \otimes x_1 \otimes x_1 \\
2^{nd} & Z_5 = \phi(x_1) \otimes x_2 \otimes x_2 \\
2^{nd} & Z_6 = \phi(x_1) \otimes x_1 \otimes x_2 \\
2^{nd} & Z_7 = \phi(x_1) \otimes x_2 \otimes x_3
\end{array}
\tag{8}
$$

When $\phi = Sigmoid$, $Z_1 = Sigmoid$, $Z_2 = Swish_{\beta=1}$, $Z_3 = GLU$, and $Z_6 = SwiGLU$. However, as far as we are aware, $Z_4 Z_5$ and Z_7 have not yet been explored. To the best of our knowledge Z_4 through Z_7 have never been explored for $Tanh$ or Sin activation functions either.

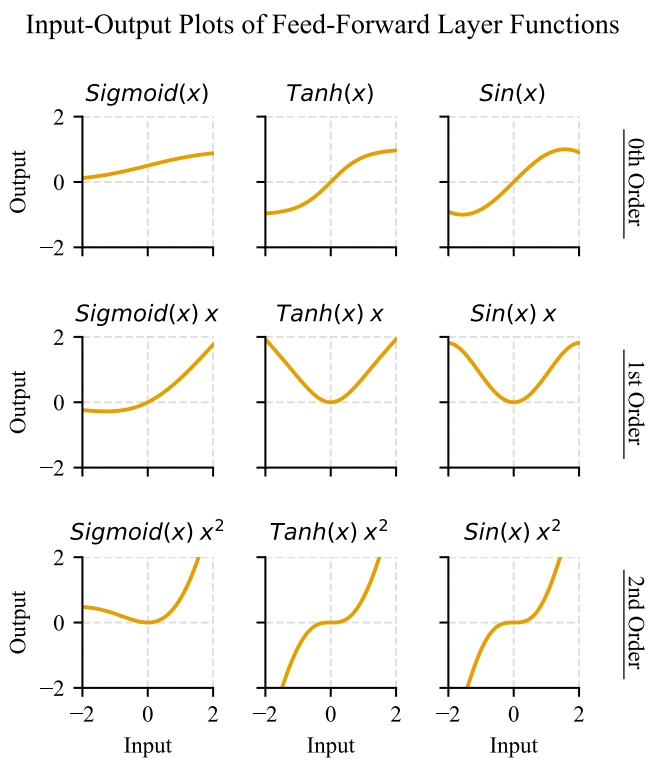

Figure 6: Input-output plots of 0th, 1st, and 2nd order GLU-type layers with *Sigmoid*, *Tanh*, and *Sin* activation functions with all weights equal to 1 and no biases. Plots are arranged by function order row-wise and by activation function column-wise. It can be seen that for all function orders *Tanh* and *Sin* activated layers have very similar input output plots near zero.

3.3 Importance of Function Order In GLU-type Layers

We suspect that the relative differences in performance of different layer functions is likely to be primarily driven by the shapes of their input-output responses. We illustrate these differences in Figure 6. First order GLU-type functions with *Sigmoid* activations (SiLU, GLU) have outputs which tend towards linearity as inputs increase in magnitude. Second order functions with *Sigmoid* (SwiGLU, etc.) are approximately parabolic for large positive inputs. This is also true of GLU-type functions with *Tanh* activations. However, unlike *Sigmoid*, *Tanh* activated functions do not tend towards zero as inputs tend towards negative infinity. Amplitudes for *Sin* activated functions also increase linearly for first order functions and parabolically for second order functions.

Table 1: ViT–Tiny model architecture

Parameter	Value
Width	192
Depth	12
MLP ratio (base)	4
MHSA heads	3
Parameters	5.5 million

3.4 Architecture

We implement 21 ViT-Tiny (Wu et al., 2022) models with the layer functions in the MLP blocks. Vision transformers are typically thought to require vast amounts of training data to suppress large negative eigenvalues in the Hessian and stabilise training (Park & Kim, 2022). Despite this, ViT-Tiny (a ViT model with substantially reduced parameters) has been shown to work well on small image datasets (Gani et al., 2022). Further, pairing a small model with small datasets allows us to efficiently evaluate a wide range of models under limited compute.

In order to ensure a fair comparison between layer functions we reduce the size of the hidden dimension of the MLP block depending on the number of unique weight matrices in the layer function. As in Shazeer's implementation we reduce the hidden dimension to:

$$HiddenDim_{new} = \frac{2 \times HiddenDim}{n_{max} + 1} \tag{9}$$

Where n_{max} is the number of unique weight matrices, i.e. 1 for Z_2, or 3 for Z_7 in equation 8. This ensures the parameter count is constant between functions. To further reduce runtime and memory usage we combine the activation and Hadamard product operations via a fused kernel generated at run time using PyTorch's JIT compiler.

3.5 Datasets

The datasets used in this study are CIFAR10 (Krizhevsky et al., 2009), CIFAR100 (Krizhevsky et al., 2009), SVHN (Netzer et al., 2011), and Imagenet64 (Chrabaszcz et al., 2017). These datasets have been widely used as benchmarks for small vision transformers (Park & Kim, 2022), (Wu et al., 2022), (Gani et al., 2022).

CIFAR-10 consists of 60,000 colour images, each of size 32x32 pixels. These images span 10 object classes such as airplanes, cars, and birds. The dataset is divided into 50,000 training images and 10,000 test images. This gives 5,000 training images and 1,000 test images per class. CIFAR-10 serves as a widely recognized benchmark for machine learning and image classification models.

CIFAR-100 also contains 60,000 colour images of size 32x32 pixels. However, these images are distributed across 100 classes. As such, there are only 500 training images and 100 test images per class. This makes CIFAR-100 useful in testing a model's capacity to learn generalizable functions when the problem space is sparsely sampled.

SVHN is a dataset of over 600,000 real-world images of house numbers taken from Google Street View. The images are 32x32 pixels and have one number at the center of each image. However almost all images show other numbers to the sides of the centered number. This adds an interesting extra challenge as the model must learn to ignore the numbers to the sides.

Imagenet64 is a variant of the Imagenet1k dataset that has been resized to 64x64 pixels. Imagenet1k has over a million labelled images covering 1,000 distinct classes of natural objects such as cats, dogs, cars etc. Imagenet1k is considered a standard benchmark for machine learning image classification.

Table 2: Training hyperparameters

Parameter	Value
Patch size	2×2 or 4×4
Image padding	4 pixels
Color jitter	0
Batch size	96
Epochs	300
Warm-up epochs	5
Weight decay	0.05
Optimizer	AdamW
Learning rate	1.25×10^{-4}
LR scheduler	cosine annealing
LR $T_{\max}$	300
Auto-augment policy	`rand-m9-n2-mstd1.0`
Label smoothing	1.0
Mixup α	0.8
CutMix α	1.0
Mixup probability	1.0

3.6 Training Regimes

We test each of the 21 models on the CIFAR10, CIFAR100, and SVHN datasets. Identifying SinGLU as the most performant function in these tests we then compare SinGLU with the current standard in state-of-the-art models (SwiGLU) on Imagenet64. As the purpose of this study is not to set new top scores for the various benchmarks but to compare relative performance between different layer functions, we choose to use the same hyperparameters for CIFAR10, CIFAR100, and SVHN as these datasets are of comparable sizes and all have images of 32x32 pixels. The hyperparameters are taken from Park & Kim (2022), and are reported in Table 2. The patch size used in all datasets other than Imagenet64 was 2x2. However, with Imagenet64 we double the patch size to 4x4 to account for the doubling in image size to 64x64.

4 Results and Discussion

In this section we present the Top-1 Accuracy of all 21 models for the four datasets. Top-1 accuracy is the proportion of times the model's most confident prediction (the class with the highest predicted probability) matches the true class label. Formally, we can say that given a set of data instances N, with each instance i having a true class label y_i, and a model prediction $\hat{y}_i$ representing the class with the highest confidence:

$$\text{Top-1 Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \begin{cases} 1, & \hat{y}_i = y_i \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

Due to the scaling by $\frac{1}{N}$ the Top-1 Accuracy function necessarily returns a value between 0 and 1. In the tables following we scale the values by a factor of 100 to give the accuracy as a percentage.

4.1 CIFAR10

In Table 3 we report the final Top-1 Accuracy for all models on the CIFAR10 dataset. Each model was trained and tested three times with different random seeds, and shown are the mean of those 3 runs. Highlighted with $\star$ are the results for SwiGLU. Highlighted with $*$ are the results for SinGLU, the function that we have identified as generally the most performant. All results greater than SwiGLU have been presented in bold. The very best results are both shown in bold and underlined. It may be seen that for CIFAR10 SinGLU outperforms SwiGLU and all other functions tested by a significant margin.

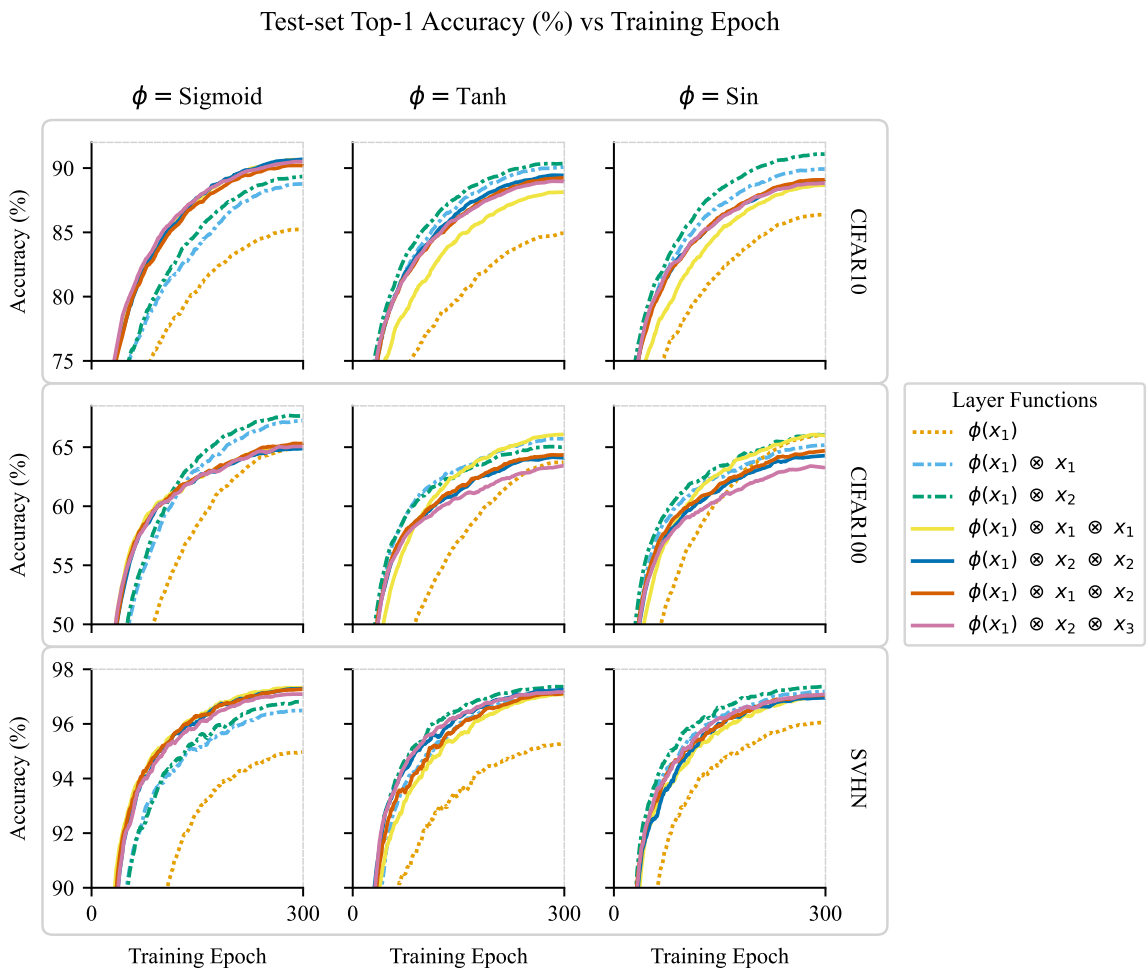

Figure 7: Plots of Top-1 Accuracy vs Training Epoch for all datasets

Table 3: Final top-1 accuracy (%) on CIFAR-10 for ViT–Tiny layers. Daggers (†) highlight the results for SinGLU; stars (⋆) highlight SwiGLU. In bold are all the results better than SwiGLU. Bold and underlined is the best result overall (SinGLU).

	Activation ϕ		
Layer function	Sigmoid	Tanh	Sin
$\phi(x_1)$	85.24	84.92	86.40
$\phi(x_1) \otimes x_1$	88.78	90.08	89.92
$\phi(x_1) \otimes x_2$	89.35	**90.36**	†**<u>91.09</u>**†
$\phi(x_1) \otimes x_1 \otimes x_1$	**90.62**	88.12	88.68
$\phi(x_1) \otimes x_2 \otimes x_2$	**90.67**	89.43	88.95
$\phi(x_1) \otimes x_1 \otimes x_2$	⋆90.19⋆	89.24	89.07
$\phi(x_1) \otimes x_2 \otimes x_3$	**90.49**	88.98	88.82

From Figure 7 it may be seen that, for the activation functions *Sigmoid*, *Tanh*, and *Sin*, model training curves group by GLU-type function order. However, it is not the case that second order functions outperform first or zeroth order functions in all cases. Instead, the best performing function order changes with activation. For *Sigmoid* activations, second order functions consistently outperform first order, which outperform zeroth. However, for *Tanh* and *Sin* activations first order functions outperform second, which outperform zeroth.

4.2 CIFAR100

Table 4: Final top-1 accuracy (%) on CIFAR-100 for ViT–Tiny layers. Daggers (†) highlight the results for SinGLU; stars (⋆) highlight SwiGLU. In bold are all the results better than SwiGLU. Bold and underlined is the best result overall.

	Activation ϕ		
Layer function	Sigmoid	Tanh	Sin
$\phi(x_1)$	65.00	**63.75**	**66.00**
$\phi(x_1) \otimes x_1$	**67.28**	**65.73**	65.20
$\phi(x_1) \otimes x_2$	**<u>67.63</u>**	65.07	†**66.06**†
$\phi(x_1) \otimes x_1 \otimes x_1$	65.21	**66.10**	**66.01**
$\phi(x_1) \otimes x_2 \otimes x_2$	64.87	64.10	64.32
$\phi(x_1) \otimes x_1 \otimes x_2$	*65.31*	64.37	64.71
$\phi(x_1) \otimes x_2 \otimes x_3$	65.07	63.42	63.32

In Table 4 we report the final Top-1 Accuracy for all 21 models on the CIFAR100 dataset. As with CIFAR10, all models were trained and tested 3 times and we present here the mean. Highlighted with $\star$ are the results for SwiGLU. Highlighted with $*$ are the results for SinGLU. Presented in bold are all results higher than SwiGLU. The very best results are shown in bold and underlined. Unlike CIFAR10, the most performant function on CIFAR100 was the original GLU: $\sigma(x_1) \otimes x_2$.

In Figure 7 we present the training curves for CIFAR100. These results are inconsistent with CIFAR10, and by a significant degree. In particular 1st and 2nd order functions perform poorly for all activations. We suspect that these results indicate that more complex functions generalize poorly when the problem space is sparsely sampled. We explore this further in Section 4.6. If we are correct and the poor results for higher order functions is due to increased propensity for overfitting this could likely be overcome by stronger regularization. For example by increasing the weight decay or using dropout.

4.3 Street View House Numbers

Table 5: Final top-1 accuracy (%) on SVHN. Daggers (†) highlight the results for SinGLU; stars (⋆) highlight SwiGLU. In bold are all the results better than SwiGLU. Bold and underlined is the best result overall (SinGLU).

	Activation ϕ		
Layer function	Sigmoid	Tanh	Sin
$\phi(x_1)$	94.97	95.28	96.07
$\phi(x_1) \otimes x_1$	96.49	97.18	97.18
$\phi(x_1) \otimes x_2$	96.82	**<u>97.36</u>**	†**<u>97.36</u>**†
$\phi(x_1) \otimes x_1 \otimes x_1$	**97.33**	97.07	97.00
$\phi(x_1) \otimes x_2 \otimes x_2$	**97.29**	97.25	96.96
$\phi(x_1) \otimes x_1 \otimes x_2$	*97.27*	97.13	97.05
$\phi(x_1) \otimes x_2 \otimes x_3$	97.09	97.17	97.08

In Table 5 we report the final Top-1 Accuracy for all 21 models on the SVHN dataset. As before, we present here the mean of three training and testing runs. Highlighted with $\star$ are the results for SwiGLU. Highlighted with $*$ are the results for SinGLU. Presented in bold are all results higher than SwiGLU. The best results are shown in bold and underlined. As with CIFAR10 the most performant model was the 1st order GLU variant with *Sin* activation which we have chosen to call SinGLU. It should be noted that SVHN has the same number of classes as CIFAR10, and slightly more images per class. It is also an arguably easier task meaning that the problem space is more densely sampled than CIFAR10.

Figure 7 shows the training curves for SVHN. These curves can be seen to be similar to those of CIFAR10. As before, the curves generally couple based on GLU-type function order. Similarly we can see that, for

Sigmoid activations, second order functions consistently outperform first order, which outperforms zeroth. Again, as with CIFAR10, for *Tanh* and *Sin* activations first order functions are consistently best, followed by second order, then zeroth.

4.4 Imagenet64

Table 6: Final top-1 accuracy (↑) on ImageNet64. The best result is shown in bold and underlined.

Layer function	**Results**
SwiGLU: $\sigma(x_1) \otimes x_1 \otimes x_2$	66.47
SinGLU: $\sin(x_1) \otimes x_2$	**67.73**

Shown in Table 6 are the Top-1 accuracies on Imagenet64 for SwiGLU, and SinGLU. As on all previous datasets SinGLU again outperforms SwiGLU by a statistically significant margin. This result is significant because it shows that the general improvement of SinGLU over SwiGLU is unlikely to be due to the smaller datasets used in our previous experiments.

4.5 Computation Time Comparisons

Table 7: Computation time per forward pass (milliseconds) for ViT–Tiny layers. Daggers (†) highlight the results for SinGLU; stars (⋆) highlight SwiGLU.

	Activation ϕ		
Layer function	Sigmoid	Tanh	Sin
$\phi(x_1)$	30.31	30.30	30.37
$\phi(x_1) \otimes x_1$	32.39	32.38	32.39
$\phi(x_1) \otimes x_2$	31.98	32.01	$^{\dagger}32.03^{\dagger}$
$\phi(x_1) \otimes x_1 \otimes x_1$	32.50	32.49	32.51
$\phi(x_1) \otimes x_2 \otimes x_2$	32.06	32.09	32.11
$\phi(x_1) \otimes x_1 \otimes x_2$	$^{\star}32.06^{\star}$	32.09	32.10
$\phi(x_1) \otimes x_2 \otimes x_3$	32.30	32.34	32.36

As previously discussed, we maintain the parameter count between models by resizing the MLP hidden layers based on the number of weight matrices. However, this does not in it's self guarantee equal computation time between models. We therefore benchmark the time taken to compute a forward pass.

We create models with patch size 2 and 10 output logits, in line with our training regime for all datasets except Imagenet64. We then generate 10 tensors of Gaussian noise of size $[128 \times 3 \times 32 \times 32]$. For each tensor we randomly initialize the models and peform 20 forward passes on an Nvidia A100 GPU, recording the time taken for the final 10. This allows a warm-up period for the timings to stabilize. For each model we take the mean across all forward passes and present these results in Table 7. We highlight the time taken for the SinGLU model with daggers and the SwiGLU model with stars.

It may be seen that there is less than 1% difference between activations (*Sigmoid*, *Tanh*, *Sin*) for any given layer function. However, there is a significant difference between 0th order and 1st order functions (maximum difference of 6.42% for *Sigmoid*, 6.42% for *Tanh*, and 6.24% for *Sin*). There is a more negligable difference between 1st and 2nd order functions (maximum difference of 1.60% for *Sigmoid*, 1.48% for *Tanh*, and 1.45% for *Sin*). It is likely that the difference in computation time between functions could be minimized by writing custom Cuda kernels which fuse operations. We leave this as an avenue for future work. It should be noted that there is less than 0.1% difference in computation time for SwiGLU and SinGLU models.

4.6 Overfitting

In Section 4.2 we suggested that the poor performance of first and second order GLU-type functions on the CIFAR-100 dataset could be due to the sparsely sampled problem space. In other words, we suggest that

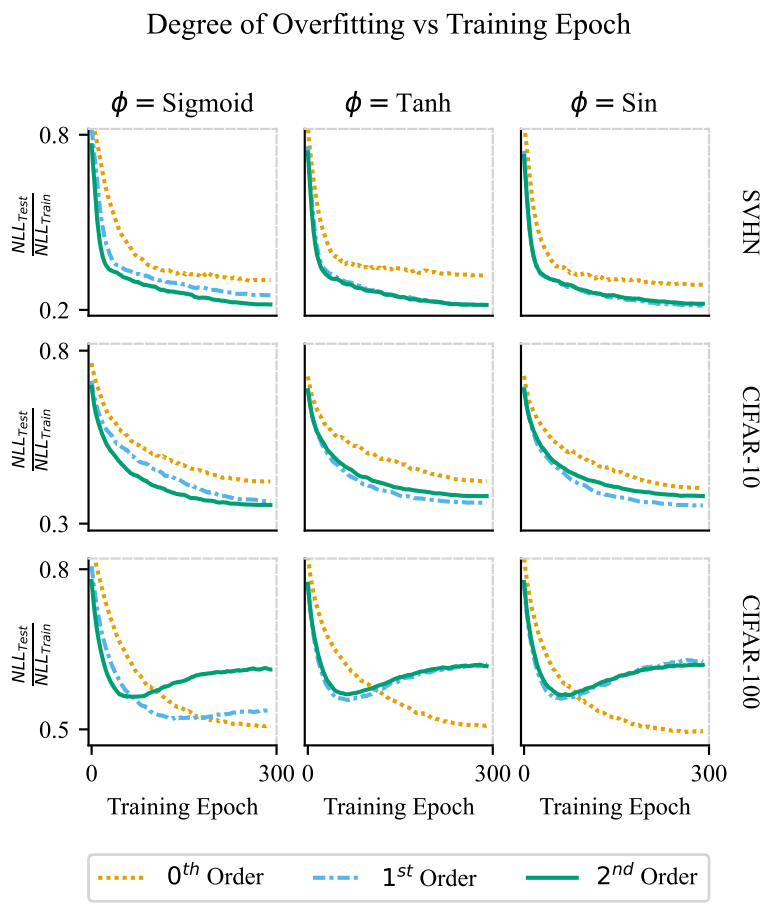

Figure 8: Plots of NLL_{Test}/NLL_{Train} vs Training Epoch for the datasets SVHN, CIFAR10, and CIFAR100. Given we previously saw tight correlation of performance for GLU-Type functions of the same order we chose to present here the mean across all functions of a given order and activation. For readability the curves have been smoothed using a rolling mean with a window of ten epochs. Downward trends indicate the models are generalizing better over time, while upward trends indicate the models are becoming increasingly overfit. Downward trends are seen for all models except 1st and 2nd order functions on CIFAR100.

more expressive, complex functions could be more prone to overfitting on datasets with few examples per class. To explore this hypothesis we compared the Negative Log Likelihood loss (NLL) of the models on the training and test sets at every epoch for the SVHN, CIFAR10, and CIFAR100 datasets. We visualize these results in Figure 8.

Dividing the NLL_{Test} by NLL_{Train} gives a measure of overfitting. Values of NLL_{Test}/NLL_{Train} greater than 1 indicate overfitting. While none of the models had values greater than 1 at any point it is important to remember that we perform heavy agumentation of the training set making the training set significantly harder than the test set. More important is the slope of the curve of NLL_{Test}/NLL_{Train} over time. A downward trend indicates the model is generalizing better over time, while an upward trend indicates the model is becoming increasingly overfit.

From Figure 8 it may be seen that all functions experience downward trends in NLL_{Test}/NLL_{Train} over time except for 1st and 2nd order functions on the CIFAR100 dataset. This suggests that the poor performance of 1st and 2nd order functions on CIFAR100 is due to increased overfitting relative to the other functions. This is consistent with our hypothesis that more expressive, complex functions are more prone to overfitting on sparsely sampled problem spaces.

5 Conclusion

In this study we systematically investigated the impact of GLU-type nonlinear layer functions in the MLP portions of Vision Transformers. We tested these layer functions with three different activation functions: $Sigmoid$, $Tanh$, and Sin. From these we identified SinGLU, a 1st order GLU-Type layer with a Sin activation as the most performant on the CIFAR10, and SVHN datasets. We then confirmed that SinGLU outperforms SwiGLU on Imagenet64 justifying its use on larger datasets. However, our testing on CIFAR100 suggests that more expressive functions may be more prone to overfitting on sparsely sampled problem spaces. We suggest that this could be mitigated by stronger regularization such as increased weight decay or dropout, or by heavier data augmentation.

We showed that different order GLU-type layers have fundamentally different dynamics, with 1st order functions having a pronounced linear quality and 2nd order functions having a pronounced parabolic quality. We further showed that this causes different layer functions of the same order to have similar training curves. However, the most performant order depends on the activation being used, with Sin and $Tanh$ showing best results with 1st order GLU functions, and $Sigmoid$ showing best results 2nd order functions.

We believe we have demonstrated that periodic activation functions such as Sin may be used in Vision Transformers without explicit changes to the architecture or training methods to account for their fundamentally non-monotonic nature.

While our results demonstrate a clear advantage for SinGLU under controlled baseline conditions, we acknowledge several limitations. First, due to computational constraints, our experiments were restricted to the ViT-Tiny architecture (approximately 5.5 million parameters) with proportionally small datasets such as CIFAR10 and ImageNet64. It remains an open question whether the performance benefits of periodic gating mechanisms generalize to massively scaled models such as ViT-Huge or Large Language Models. Second, to ensure a strictly controlled comparison, we applied an identical training recipe and hyperparameter set across all 21 models. It is entirely possible that SwiGLU and other higher-order functions might achieve better relative performance if hyperparameters such as learning rate, weight decay, and initialization schemes were independently optimized for each specific layer function. Future work should explore the scalability of SinGLU on larger architectures and datasets, alongside function-specific hyperparameter tuning.

References

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

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.

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

Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969. doi: 10.1109/TSSC.1969.300225.

Hanan Gani, Muzammal Naseer, and Mohammad Yaqub. How to train vision transformer on small-scale datasets? *ArXiv*, abs/2210.07240, 2022. URL `https://api.semanticscholar.org/CorpusID:252873104`.

Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `https://proceedings.mlr.press/v9/glorot10a.html`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL `http://arxiv.org/abs/1502.01852`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

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

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pp. 448–456, 2015.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009.

Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/f1748d6b0fd9d439f71450117eba2725-Paper.pdf`.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011. URL `http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf`.

Marieme Ngom and Oana Marin. Fourier neural networks as function approximators and differential equation solvers. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 14(6):647–661, 2021.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Giambattista Parascandolo, Heikki Huttunen, and Tuomas Virtanen. Taming the waves: sine as activation function in deep neural networks. 2016.

Namuk Park and Songkuk Kim. How do vision transformers work? *arXiv preprint arXiv:2202.06709*, 2022.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? volume 31, 2018.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. volume 33, pp. 7462–7473, 2020.

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

Shikhar Tuli, Ishita Dasgupta, Erin Grant, and Thomas L Griffiths. Are convolutional neural networks or transformers more like human vision? *arXiv preprint arXiv:2105.07197*, 2021.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. volume 30, 2017.

Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Tinyvit: Fast pretraining distillation for small vision transformers. pp. 68–85, 2022.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pp. 10524–10533. PMLR, 2020.

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