

ViKANformer: Embedding Kolmogorov Arnold Networks in Vision Transformers for Pattern-Based Learning

Shreyas S

*School of Computer Science and Engineering
VIT-AP University, India*

Akshath M

*School of Computer Science and Engineering
VIT-AP University, India*

Abstract—Vision Transformers (ViTs) have significantly advanced image classification by applying self-attention on patch embeddings. However, the standard MLP blocks in each Transformer layer may not capture complex nonlinear dependencies optimally. In this paper, we propose ViKANformer, a Vision Transformer where we replace the MLP sub-layers with Kolmogorov–Arnold Network (KAN) expansions, including *Vanilla KAN*, *Efficient-KAN*, *Fast-KAN*, *SineKAN*, and *FourierKAN*, while also examining a *Flash Attention* variant. By leveraging the Kolmogorov–Arnold theorem, which guarantees that multivariate continuous functions can be expressed via sums of univariate continuous functions, we aim to boost representational power. Experimental results on MNIST demonstrate that SineKAN, Fast-KAN, and a well-tuned Vanilla KAN can achieve over 97% accuracy, albeit with increased training overhead. This trade-off highlights that KAN expansions may be beneficial if computational cost is acceptable. We detail the expansions, present training/test accuracy and F1/ROC metrics, and provide pseudocode and hyperparameters for reproducibility. Finally, we compare ViKANformer to a simple MLP and a small CNN baseline on MNIST, illustrating the efficiency of Transformer-based methods even on a small-scale dataset.

Index Terms—Vision Transformer, Kolmogorov Arnold Networks, MNIST, Attention Mechanisms, Deep Learning, Flash Attention

I. INTRODUCTION

The Transformer architecture [1] has dramatically improved performance in NLP tasks, and its adaptation to images, the Vision Transformer (ViT) [2], has also achieved strong results. ViTs divide images into patches, embed them, and rely on self-attention over the patch embeddings. However, the feed-forward sub-layers (MLPs) may not optimally capture intricate patterns.

Kolmogorov Arnold Networks (KANs) exploit the Kolmogorov Arnold theorem [3], [4], which states any continuous function of n variables can be decomposed into sums of univariate continuous mappings plus additions. In practice, expansions such as *Sine* [5], *Fourier*, *radial basis*, or *polynomial* can be used dimension by dimension. We embed such expansions within ViT feed-forward layers, replacing the standard MLP. Additionally, we experiment with *Flash Attention*, an approach for more efficient attention, to test synergy with KAN expansions.

Contributions:

- We propose **ViKANformer**, a plug-and-play code framework that uses KAN expansions in place of standard MLPs in Vision Transformers.
- We benchmark multiple KAN variants (Vanilla, Sine, Fourier, Fast, Efficient) plus a Flash Attention version on the MNIST dataset.
- Empirical results show that while expansions such as SineKAN, Fast-KAN, and tuned Vanilla KAN can surpass 97–98% accuracy, they incur higher training costs (7–47 min/epoch).
- We discuss a simple MLP and a small CNN baseline on MNIST for comparison, noting that while these methods can reach comparable or higher accuracy with less overhead, our aim is to demonstrate the viability of KAN expansions within Transformer-based pipelines.

II. RELATED WORK AND LITERATURE

A. Vision Transformers

ViTs [2] chunk an image into patches (e.g., 16×16 or smaller/larger), flatten, and embed them. Positional embeddings are added, then a series of Transformer blocks with multi-head self-attention plus feed-forward sub-layers is applied. While very successful, research continues on optimizing or improving these feed-forward sub-layers, *e.g.*, **MLP-Mixer**, **ConvMixer**, or, in our case, **KAN expansions**.

B. Kolmogorov Arnold Theorem

Kolmogorov [3] proved that any continuous $f(\mathbf{x})$ on $[0, 1]^n$ can be expressed as finite sums of univariate continuous functions plus addition. The theorem is non-constructive, so practical “KANs” use expansions to approximate these univariate pieces. Recent expansions:

- *Sine* expansions [5],
- *Fourier* expansions,
- *Radial basis* expansions,
- *Polynomial* or *B-spline* expansions.

They can be dimension-wise or can share parameters across dimensions, with varying overhead.

C. Flash Attention

Flash Attention is a more efficient attention mechanism that computes $\mathbf{Q}\mathbf{K}^\top$ blocks in a memory-optimized way.

Some prior works incorporate better feed-forward designs with Flash Attention to further accelerate Transformers. We attempt a **FlashKAN** approach, combining Flash-based self-attention with KAN expansions in the feed-forward sub-layer.

III. ViKANFORMER ARCHITECTURE

A. Replacing MLP with KAN

Our approach is to replace the standard MLP block in the Transformer layer with dimension-wise KAN expansions. Suppose we have d -dimensional embeddings. A KAN feed-forward block has the form:

$$\mathbf{y} = \mathbf{W} [\phi_1(x_1) \oplus \cdots \oplus \phi_d(x_d)],$$

where each x_j passes through a parametric univariate function ϕ_j . For instance, in SineKAN:

$$\phi_j(x_j) = \sum_{m=1}^M \alpha_{j,m} \sin(\omega_{j,m} x_j + b_{j,m}). \quad (1)$$

The same overall Transformer structure remains intact—multi-head attention, layer normalization, etc.—but the feed-forward sub-layer is replaced by the chosen KAN variant. This modular “plug-and-play” design allows quick experimentation with different expansions.

B. Architecture Diagram

Figure 1 illustrates an overview of the ViKANformer, in a two-column figure for clarity. We use a small Vision Transformer on MNIST as a proof of concept. The main modifications affect only the MLP blocks, while the rest of the Transformer (attention, skip connections, normalization) remains standard.

IV. IMPLEMENTATION DETAILS AND PSEUDOCODE

A. KAN Hyperparameters and Initialization

Each KAN variant requires choices of expansion size and parameter initialization:

- **SineKAN / FourierKAN:** We set $M = 8$ frequencies per dimension. Frequencies and phases $(\omega_{j,m}, b_{j,m})$ are initialized from a uniform distribution in $[-1, 1]$. The amplitude coefficients $\alpha_{j,m}$ are also learned.
- **Fast-KAN:** Uses a radial-basis (RBF) expansion with 5 centers per dimension. Centers and widths are initialized randomly in a range $[0, 1]$, then learned via backprop.
- **Efficient-KAN:** Uses piecewise polynomial (B-spline) expansions of order 3, with a small set of knot points per dimension (we used 6). These expansions can become quite large internally.

All parameters ($\alpha_{j,m}, \omega_{j,m}, b_{j,m}$, or RBF centers, etc.) are trained end-to-end via backpropagation. The overhead grows with M , the number of expansion terms per dimension. For certain expansions, we found that the GPU memory usage and computations scale rapidly with M and the input embedding dimension, explaining the high training time on an A100 for, e.g., *Efficient-KAN*.

B. Training on A100 GPU with Hyperparameters

We train all models (standard ViT and KAN-based variants) on an NVIDIA A100 GPU. We typically use:

- **Dataset:** MNIST (60k train / 10k test, 28×28).
- **ViT Config:** Patch size of 7×7 , thus yielding **16 patches** total. (Each patch is $7 \times 7 = 49$ pixels, then flattened.)
- **Number of Transformer blocks:** 2
- **Number of attention heads:** 2
- **Embedding dimension:** $d = 8$
- **Batch size:** 128
- **Epochs:** 10 or 20 (depending on variant)
- **Optimizer:** Adam, learning rate $\approx 0.001\text{--}0.005$
- **Loss:** Cross-entropy for classification

Algorithm 1 shows high-level pseudocode.

Algorithm 1 Training Algorithm for ViKANformer

Input: model M (ViKANformer), training set $\mathcal{D}_{\text{train}}$, testing set $\mathcal{D}_{\text{test}}$, learning rate α , number of epochs E , batch size B

Output: trained model M^*

- 1: Initialize M parameters (KAN expansions, etc.)
 - 2: **for** epoch = 1 to E **do**
 - 3: Shuffle $\mathcal{D}_{\text{train}}$ into mini-batches of size B
 - 4: **for** each mini-batch (x, y) in $\mathcal{D}_{\text{train}}$ **do**
 - 5: $y_{\text{hat}} \leftarrow M(x)$
 - 6: $\ell \leftarrow \text{CrossEntropyLoss}(y_{\text{hat}}, y)$
 - 7: Zero out gradients in M
 - 8: $\ell.\text{backward}()$ // backprop
 - 9: Update parameters of M using Adam with lr = α
 - 10: **end for**
 - 11: Evaluate M on $\mathcal{D}_{\text{test}}$ (compute accuracy/F1/etc.)
 - 12: **end for**
 - 13: **return** M^*
-

V. EXPERIMENTS ON MNIST

A. Implementation and Setup

a) **Dataset:** We use MNIST [6], which consists of 28×28 grayscale digit images (60,000 training, 10,000 test).

b) **ViT Configuration:** We divide each 28×28 image into 7×7 **patches**, yielding **16 patches** total. Each patch is flattened into 49 pixels, then embedded to dimension $d = 8$. We add learned positional embeddings. We use 2 Transformer blocks, each with 2 attention heads.

c) **KAN Variants:** We test:

- **Vanilla KAN:** minimal dimension-wise expansion,
- **SineKAN:** expansions using $\sin(\omega x + b)$,
- **FourierKAN:** expansions using $\sin(kx)$ and $\cos(kx)$,
- **Fast-KAN:** radial basis expansions (Gaussian RBF),
- **Efficient-KAN:** typically B-spline or piecewise polynomials.

d) **Flash Attention Variant:** We incorporate Flash Attention in two forms:

- **Flash-ViT:** regular MLP feed-forward but flash-based self-attention.

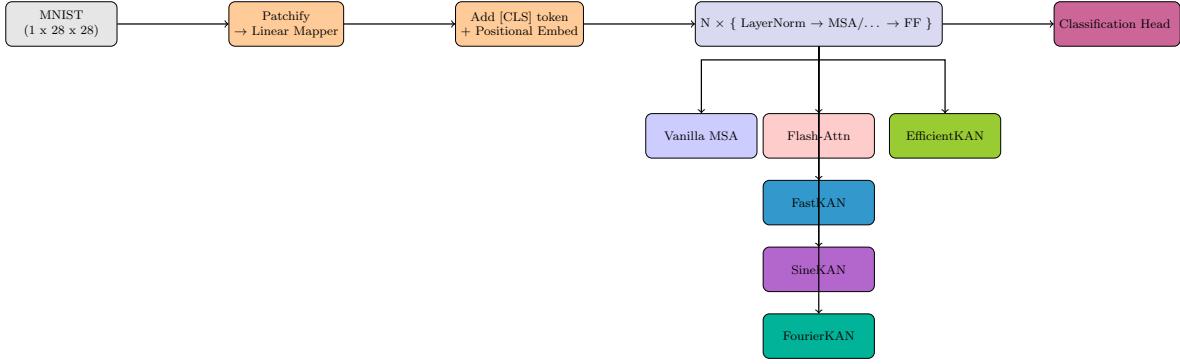


Fig. 1: **ViKANformer Overview.** We show two Transformer blocks with their self-attention sub-layer. The feed-forward sub-layer (normally an MLP) is replaced by a dimension-wise KAN expansion. Various KAN variants (Sine, Fourier, etc.) can be plugged in.

- *FlashKAN-ViT*: KAN expansions in the feed-forward plus flash-based attention.
- e) *Training Details*: All models are trained for 10 epochs on an NVIDIA A100 GPU, with Adam optimizer and learning rate in $[0.003, 0.005]$, batch size 128. Approximate *time per epoch*:

TABLE I: Approximate time per epoch across variants.

Variant	Time (minutes/epoch)
Vanilla KAN	7
SineKAN	9
FourierKAN	8
Fast-KAN	20
Efficient-KAN	47
Flash-ViT (std. MLP)	1–2
FlashKAN-ViT (KAN+Flash)	3–5

B. Accuracy and Loss Curves

Figures 2 and 3 show representative training/test accuracy curves across epochs. SineKAN, Fast-KAN, and a carefully tuned Vanilla KAN consistently converge to higher accuracy (97–98%).

Beyond raw accuracy, we also track F1-score and one-vs-rest ROC AUC. Figure 4 shows expansions often reach 0.95+ F1 by epoch 5 and near-perfect ROC AUC by epoch 8–10.

C. Additional Baselines on MNIST

Although our focus is on embedding KAN expansions into ViT, one might wonder how a simple MLP or a standard CNN perform on MNIST:

- **2-layer MLP** with 128 hidden units can reach ~97% test accuracy in under a minute per epoch on CPU/GPU.
- **LeNet-like CNN** can surpass 99% test accuracy on MNIST, typically running very quickly on a modern GPU.

Thus, while KAN-based ViTs can achieve 97–98% accuracy, they are *not* necessarily more efficient or higher-accuracy than classic baselines on such a small dataset. Our results simply

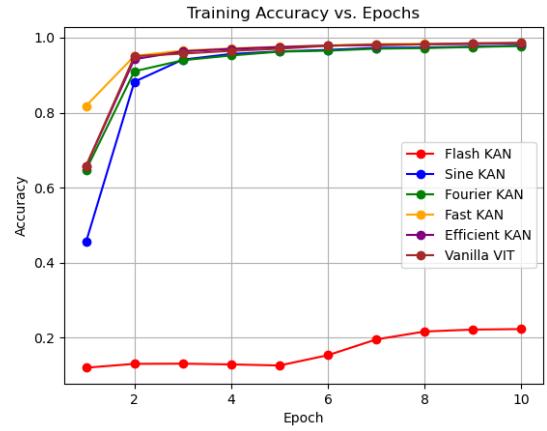


Fig. 2: Training Accuracy vs. Epochs on MNIST. SineKAN, Fast-KAN, and Vanilla KAN exceed 95–97% by epoch 5–6.

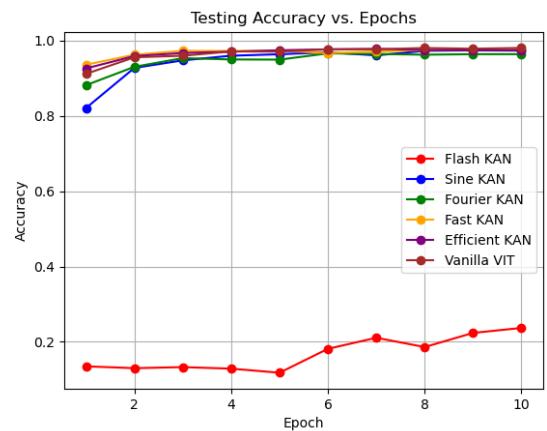
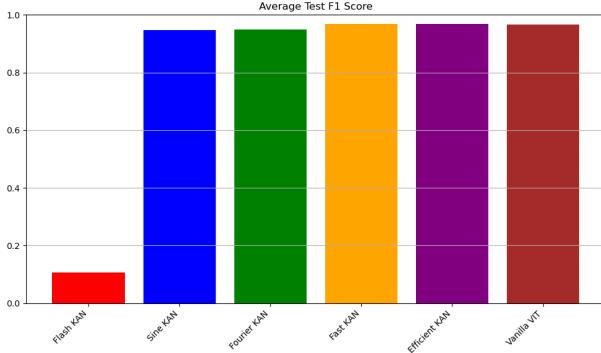
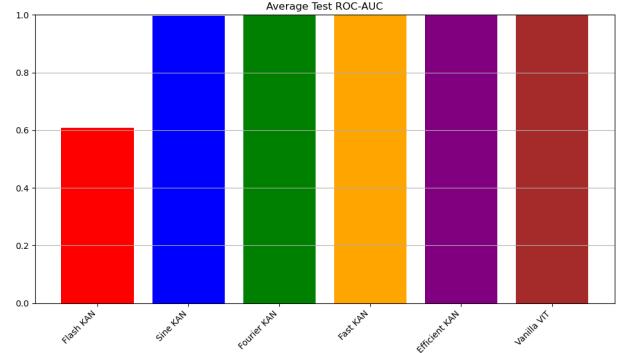


Fig. 3: Test Accuracy vs. Epochs on MNIST. SineKAN and Fast-KAN reach 97–98% by epoch 10, with Vanilla KAN close behind.



(a) F1 Score vs. Epochs



(b) ROC AUC vs. Epochs

Fig. 4: All expansions eventually surpass 0.95 F1, with SineKAN and Fast-KAN frequently reaching 0.98+ and ROC AUC near 1.0.

illustrate that KAN expansions in a Transformer pipeline can learn effectively, if one accepts additional computational overhead.

D. Final Performance Metrics

Table II summarizes the final test performance (accuracy, F1, ROC) after 10 epochs. While Fast-KAN and Efficient-KAN match or exceed $\sim 97\%$, they incur heavy time costs. SineKAN and a well-tuned Vanilla KAN also reach the 97–98% range, with somewhat lower overhead.

TABLE II: MNIST final test results after 10 epochs.

Variant	Acc.	F1	ROC AUC	Time/epoch
Vanilla KAN	98.0%	0.9808	0.9997	7 min
SineKAN	97.8%	0.9789	0.9996	9 min
FourierKAN	96.6%	0.9662	0.9991	8 min
Fast-KAN	97.8%	0.9789	0.9997	20 min
Efficient-KAN	97.4%	0.9744	0.9996	47 min

Flash Attention Results. Using Flash Attention alone (standard MLP) trains in 1–2 min/epoch but can yield slightly lower final accuracy unless carefully tuned. FlashKAN-ViT (KAN expansions + Flash) yields 3–5 min/epoch training times and can approach the top accuracy if the KAN hyperparameters are well-tuned.

VI. DISCUSSION AND FUTURE DIRECTIONS

A. Key Observations

- 1) SineKAN, Fast-KAN, and a *carefully tuned* Vanilla KAN can exceed 97–98% test accuracy on MNIST, with F1 and ROC near 0.98–1.0.
- 2) FourierKAN typically saturates around 96–97%.
- 3) Efficient-KAN approaches 97.4% but suffers from large training overhead (47 min/epoch).
- 4) Simple MLP or CNN baselines on MNIST can reach similar or better accuracy with far less overhead, highlighting that the main value here is *demonstrating viability of KAN expansions* within a Transformer pipeline.

B. Potential Extensions

Scaling Up. Testing these expansions on CIFAR-10 or ImageNet would reveal whether KAN expansions remain beneficial for larger-scale tasks.

Adaptive Expansions. Dynamically learning the number of frequencies, RBF centers, or polynomial degrees could reduce overhead without sacrificing representational power.

Hybrid MLP/KAN. Partial expansions for certain dimensions, combined with a standard MLP, might strike a balance between representational power and computational cost.

GPU-Optimized B-Splines/RBF. Specialized GPU kernels could reduce the training overhead, especially for radial basis or polynomial expansions.

VII. CONCLUSION

We have presented **ViKANformer**, a Vision Transformer that replaces standard MLP layers with dimension-wise Kolmogorov–Arnold Network expansions. On MNIST, SineKAN, Fast-KAN, and tuned Vanilla KAN can reach 97–98% accuracy, with higher overhead. FourierKAN and Efficient-KAN also show strong performance, though either saturating at lower accuracy or incurring steep training costs. A *Flash Attention* variant reduces training time but requires careful hyperparameter tuning to maintain high accuracy. Overall, KAN expansions can significantly boost representation capability within a Transformer framework, provided additional computational resources are acceptable. Future work will focus on scaling these expansions to larger datasets and exploring more efficient partial/hybrid expansions.

REFERENCES

- [1] A. Vaswani *et al.*, “Attention is All You Need,” *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.
- [2] A. Dosovitskiy *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” *arXiv preprint arXiv:2010.11929*, 2020.
- [3] A. N. Kolmogorov, “On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of One Variable and Addition,” *Doklady Akademii Nauk SSSR*, vol. 114, no. 5, pp. 953–956, 1957.
- [4] Z. Liu *et al.*, “Kolmogorov–Arnold Networks,” *arXiv preprint arXiv:2404.19756*, 2024 (forthcoming).

- [5] E. Reinha *et al.*, “SineKAN: Kolmogorov-Arnold Networks Using Sineusoidal Activation Functions,” *arXiv preprint arXiv:2407.04149*, 2024 (forthcoming).
- [6] Y. LeCun *et al.*, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.