MetaCluster: Enabling Deep Compression of Kolmogorov-Arnold Networks

Matthew Raffel* **Adwaith Renjith*** **Lizhong Chen**
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331
{raffelm, renjitha, chenliz}@oregonstate.edu

Abstract

Kolmogorov-Arnold Networks (KANs) replace scalar weights with per-edge vectors of basis coefficients, thereby increasing expressivity and accuracy while also leading to a multiplicative increase in parameters and memory usage. We propose MetaCluster, a framework that makes KANs highly compressible without sacrificing accuracy. Specifically, a lightweight meta-learner, trained jointly with the KAN, maps low-dimensional embeddings to coefficient vectors, thereby constraining them to lie on a low-dimensional manifold amenable to clustering. We then run K-means in coefficient space and replace per-edge vectors with shared centroids. Afterward, the meta-learner can be discarded, and a brief fine-tuning of the centroid codebook recovers any residual accuracy loss. The resulting model stores only a small codebook and per-edge indices, exploiting the vector nature of KAN parameters to amortize storage across multiple coefficients. On MNIST, CIFAR-10, and CIFAR-100, across standard KANs and ConvKANs using multiple basis functions, MetaCluster achieves a reduction of up to $80\times$ in parameter storage, with minimal impact on accuracy. Similarly, in high-dimensional equation modeling tasks, MetaCluster achieves a $124.1\times$ parameter reduction while preserving baseline MSE.

1 Introduction

Kolmogorov–Arnold Networks (KANs) have recently emerged as a compelling alternative to multilayer perceptrons (MLPs), delivering strong results in equation modeling and scientific machine learning, often achieving improved task performance with comparable or lower parameter counts in these settings [Liu et al., 2024b, Li et al., 2025, Coffman and Chen, 2025, Koenig et al., 2024]. Recently, KANs have also shown promise in computer vision [Yang and Wang, 2024, Raffel and Chen, 2025]. However, unlike equation modeling, KANs at larger scales often incur substantial parameter overhead relative to MLPs [Yu et al., 2024]. This overhead stems from the per-edge degrees of freedom in KANs: each connection carries a vector of basis coefficients (e.g., B-spline weights) rather than a single scalar weight, resulting in a multiplicative increase in the number of parameters.

One approach that targets this multiplicative increase in parameters is weight sharing, which clusters parameters into a codebook that stores compact indices. Unfortunately, naively applying weight sharing to KANs is ineffective. Instead of clustering scalars (as in MLPs), we must cluster high-dimensional coefficient vectors in KANs. In such high-dimensional spaces, absolute distances increase and become more concentrated (so that the nearest and farthest points become similar), causing clustering methods to struggle to form tight clusters [Beyer et al., 1999, Donoho et al., 2000].

*Equal contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

We address this challenge with MetaCluster[1], a three-stage compression framework that merges meta-learning with weight sharing. First, a small meta-learner maps low-dimensional embeddings to per-edge coefficient vectors, thereby constraining KAN activations to a low-dimensional manifold during training on the task loss. This manifold shaping makes the coefficient vectors highly clusterable. Second, we run K-means on the generated coefficients, replacing per-edge weights with codebook centroids indexed with compact codes. Finally, we discard the meta-learner and embeddings and lightly fine-tune the centroids to recover any lost accuracy. Since each centroid stores an entire coefficient vector, the codebook amortizes over many scalars, yielding a much higher compression factor for KANs than for MLPs at the same number of clusters.

We validate MetaCluster on two model families consisting of a fully-connected KAN [Liu et al., 2024b] and a convolutional KAN (ConvKAN) [Bodner et al., 2024, Drokin, 2024]. For each of these models, we test the efficacy of using B-Splines, radial basis functions (RBFs), and Gram polynomials as the bases [Liu et al., 2024b, Li, 2024, SS et al., 2024]. To further verify the robustness of our approach, we validate it across MNIST [Deng, 2012], CIFAR-10, CIFAR-100 [Krizhevsky, 2009], and equation modeling. From our experiments, we find that MetaCluster achieves up to $80\times$ and $124.1\times$ reductions in parameter storage for image classification and equation modeling, respectively, relative to the uncompressed KAN while closely matching original performance. We also find it is robust across various architectures and datasets, and ablations confirm that enforcing a low-dimensional manifold is key to high-quality clustering.

The main contributions of this paper are:

1. We identify the potential of weight-sharing for reducing the KANs memory footprint and, to our knowledge, provide the first effective weight-sharing method tailored to KANs.
2. We propose a meta-learning approach that shapes per-edge KAN coefficients to lie on a low-dimensional manifold, enabling effective clustering in high dimensions.
3. We provide extensive image classification and equation modeling experiments demonstrating up to $80\times$ and $124.1\times$ memory reduction, respectively, while largely preserving accuracy, along with extensive ablations.

2 Preliminaries and Motivation

2.1 Kolmogorov-Arnold Networks

KANs have gained attention as an alternative to conventional MLPs [Liu et al., 2024b]. Their design is motivated by the Kolmogorov–Arnold representation theorem, which guarantees that any continuous multivariate function on a bounded domain, $f : [0, 1]^n \to \mathbb{R}$, can be expressed as a finite sum of compositions of univariate continuous functions, $\phi_{q,p} : [0, 1] \to \mathbb{R}$ and $\phi_q : \mathbb{R} \to \mathbb{R}$ such that

$$f(\mathbf{x}) = f(x_1, ..., x_n) = \sum_{q=1}^{2n+1} \phi_q(\sum_{p=1}^{n} \phi_{q,p}(x_p)). \tag{1}$$

While Equation 1 captures the classical theoretical form, Liu et al. [2024b] demonstrates a practical generalization that permits arbitrary width and depth tailored to the task. We can express such a formulation with L layers as

$$\begin{aligned} f(\mathbf{x}) &= (\Phi_L \circ \Phi_{L-1} \circ ... \circ \Phi_1)\mathbf{x}, \\ \Phi_l &= \begin{bmatrix} \phi_{l,1,1}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \vdots & \ddots & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{bmatrix} \end{aligned} \tag{2}$$

In Equation 2, we let n_l denote the number of inputs to layer l, with inputs $x_{l,i}$. Each edge from input i to output j in layer l has a learnable univariate activation $\phi_{l,j,i}(\cdot)$, for $i \in [1, n_l]$ and $j \in [1, n_{l+1}]$.

[1] Our code is publicly available at `https://github.com/OSU-STARLAB/MetaCluster`.

The most common choice for implementing $\phi_{l,j,i}(\cdot)$ has been through a weighted summation of basis functions $B_i(\cdot)$ parameterized with $\mathbf{w} = [w_1, ..., w_{|\mathbf{w}|}]$ as shown with,

$$\phi_{l,j,i}(\cdot) = \sum_{i=1}^{|\mathbf{w}|} w_i B_i(\cdot). \tag{3}$$

From its increased representation power compared to the MLP, the KAN has offered impressive results on equation modeling tasks for scientific applications [Li, 2024, Li et al., 2025, Coffman and Chen, 2025, Koenig et al., 2024] and has even started to extend its reach into computer vision [Raffel and Chen, 2025, Yang and Wang, 2024]. Despite these gains, widespread adoption in modern large-scale architectures has been hindered by memory inefficiency [Yu et al., 2024]. The root cause is structural: each KAN edge carries a vector of basis coefficients (e.g., B-spline weights), whereas an MLP edge carries a single scalar. Ignoring biases, an arbitrary MLP with L layers will possess $\sum_{l=0}^{L-1}(n_l \times n_{l+1})$ parameters, whereas a KAN will possess $\sum_{l=0}^{L-1}(n_l \times n_{l+1}) \times (|\mathbf{w}|)$ parameters. Thus, for identical topologies, a KAN is approximately $|\mathbf{w}|$ times larger than its MLP counterpart.

2.2 KAN Weight Sharing Motivations and Challenges

These observations motivate a compression strategy that targets the dimensionality of the vector of basis coefficients. Weight sharing is one method that reduces dimensionality directly by clustering parameters into a small codebook and storing compact indices. In its classical form for MLPs, one applies K-means to the set of scalar weights $\mathbf{W} = w_1, w_2, \ldots, w_n$, obtaining centroids $\mathbf{C} = c_1, c_2, \ldots, c_k$ and assignments that minimize within-cluster sum of squares [Han et al., 2015]:

$$\arg\min_{\mathcal{C}} \sum_{i=1}^{k} \sum_{w \in C_i} (w - c_i)^2 \tag{4}$$

Weight sharing has repeatedly been shown to preserve accuracy while substantially reducing the number of parameters [Han et al., 2015, Cho et al., 2021]. In MLPs, however, their effectiveness is constrained by the need to store, for every weight, an index that maps back to a codebook centroid; an overhead that limits the ultimate compression. Under a simple model with n weights, a codebook of k centroids stored at b bits per scalar, the achievable compression factor is

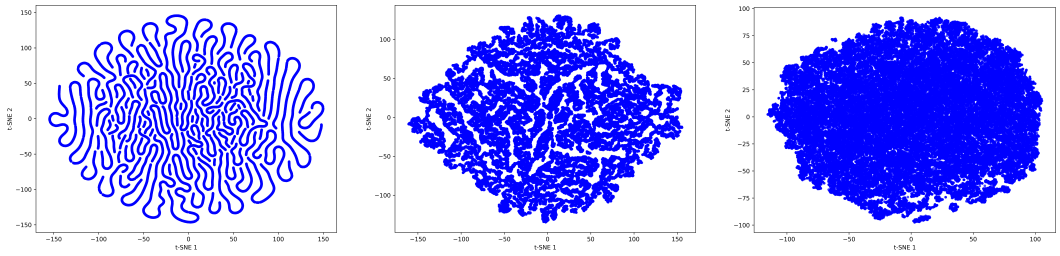
$$r = \frac{nb}{n \log_2(k) + kb} \tag{5}$$

The dominant term in the denominator, $n \log_2 k$, represents the per-weight index cost required to record the mapping from each original weight to its corresponding centroid. When extending weight sharing to KANs, the index mapping is still required, but its relative cost can be significantly reduced because we cluster $|\mathbf{w}|$-dimensional coefficient vectors per edge. Thus, each centroid stores $|\mathbf{w}|$ scalars, amortizing the codebook over many parameters while the index cost remains $n \log_2 k$. We detail this amortization and its impact on compression in Section 3.4.

While weight sharing is promising for substantially reducing storage in KANs, directly applying it to KANs for model compression is nontrivial. For instance, since each edge on a KAN is defined by $|\mathbf{w}|$ weights rather than 1 as in the MLP, the dimensionality of the points we must cluster is far greater. In increasing the dimensionality, the vector space becomes sparser and points become more equally spaced apart, making clustering points more difficult [Beyer et al., 1999, Donoho et al., 2000]. Therefore, to effectively cluster KAN activations, we require a fundamentally different method to transform the high-dimensional vector into a more manageable space.

3 Methods

This section presents MetaCluster, a three-stage framework that makes per-edge KAN activation coefficients amenable to clustering and compression. First, we train a single meta-learner that maps compact embeddings to full activation-coefficient vectors, thereby forcing them onto a task-aligned low-dimensional manifold. Next, we apply K-means to those vectors and record, for each edge, the centroid to which it belongs in that layer. Finally, we discard the meta-learner and embeddings, replace per-edge weights with their assigned centroids via a lookup table, and briefly fine-tune the network to recover any lost accuracy.



(a) Meta-learner when $d_{emb} = 1$ (b) Meta-learner when $d_{emb} = 2$ (c) KAN

Figure 1: T-SNE visualization of KAN activation weights with and without the meta-learner M_θ. (a) With a 1D embedding, the MetaKAN learns a nearly one-dimensional manifold, (b) with a 2D embedding, it learns a structured two-dimensional manifold, while (c) the baseline KAN without a meta-learner fails to organize the weights into a coherent low-dimensional structure.

3.1 Manifold Learning through a Meta-learner

Clustering the weights of a KAN is challenging due to the high dimensionality of each weight vector. To address this, we introduce a single meta-learner, M_θ, which maps a lower-dimensional embedding $\mathbf{z}_i \in \mathbb{R}^{d_{\text{emb}}}$ to the full KAN weights $\mathbf{w}_i \in \mathbb{R}^{|\mathbf{w}|}$, constraining them to lie on a low-dimensional manifold. Formally, the mapping is defined as:

$$M_\theta(\mathbf{z}_i) = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{z}_i + b_1) + b_2 = \mathbf{w}_i, \tag{6}$$

where $W_1 \in \mathbb{R}^{d_{\text{hidden}} \times d_{\text{emb}}}$, $W_2 \in \mathbb{R}^{|\mathbf{w}| \times d_{\text{hidden}}}$, and $\sigma(\cdot)$ is a ReLU activation. The meta-learner is trained jointly with the KAN using standard backpropagation for α epochs, so that the generated weights $\mathbf{w_i}$ both lie on a meaningful manifold and optimize the task-specific loss.

To demonstrate the manifold-shaping effect of the meta-learner, we train a two-layer KAN with a B-spline basis on CIFAR-10 [Krizhevsky, 2009] for one epoch under three settings: (i) a meta-learner with $d_{\text{emb}} = 1$, (ii) a meta-learner with $d_{\text{emb}} = 2$, and (iii) a baseline KAN without a meta-learner. We collect all per-edge activation coefficient vectors $\mathbf{w}_i$ and visualize them with t-SNE. Each point corresponds to $\mathbf{w}_i \in \mathbb{R}^{|\mathbf{w}|}$, where $|\mathbf{w}| = 9$. As shown in Figure 1a, when $d_{\text{emb}} = 1$ the model organizes coefficients along an effectively one-dimensional manifold. Increasing to $d_{\text{emb}} = 2$ produces the well-structured two-dimensional sheet shown in Figure 1b. In contrast, the baseline KAN provided in Figure 1c yields a diffuse cloud without coherent low-dimensional structure, which hampers downstream clustering. This qualitative observation is quantitatively supported by our Principal Component Analysis and cluster validation metrics in Appendix A.5, which confirm that the meta-learner effectively restricts the variance of the coefficient space to the target dimensionality [Maćkiewicz and Ratajczak, 1993]. These results substantiate our premise: shaping coefficient vectors via a low-dimensional embedding makes them markedly more clusterable, a property we later translate into higher compression at fixed accuracy, as shown in Section 4.2.

3.2 KAN Activation Compression with MetaClustering

We will now outline our method for clustering the weights $M_\theta(\mathbf{z}_i)$ using K-means. Unlike Equation 4, the weights $M_\theta(\mathbf{z}_i)$ lie in $\mathbb{R}^{|\mathbf{w}|}$ rather than $\mathbb{R}$ as with the scalar weights w_i. Accordingly, each centroid $\mathbf{c}_i$ also resides in $\mathbb{R}^{|\mathbf{w}|}$. Therefore, our K-means objective must be adapted as follows:

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{M_\theta(\mathbf{z}) \in C_i} \left\| M_\theta(\mathbf{z}) - \mathbf{c}_i \right\|_2^2, \tag{7}$$

where $\mathbf{c}_i \in \mathbb{R}^{|\mathbf{w}|}$ denotes the centroid of cluster C_i.

The centroids are updated iteratively according to the standard K-means update rule:

$$\mathbf{c}_i = \frac{1}{|C_i|} \sum_{M_\theta(\mathbf{z}_j) \in C_i} M_\theta(\mathbf{z}_j), \tag{8}$$

ensuring that each centroid represents the mean of the weight vectors assigned to its cluster.

To record assignments, we define an index mapping vector $I \in \{1, 2, \ldots, k\}^N$, where the j-th entry indicates the cluster index of $M_\theta(\mathbf{z}_j)$. Formally,

$$I_j = \arg \min_{i \in \{1,\ldots,k\}} \left\| M_\theta(\mathbf{z}_j) - \mathbf{c}_i \right\|_2^2, \tag{9}$$

so that the mapping is expressed as $M_\theta(\mathbf{z}_j) \mapsto \mathbf{c}_{I_j}$. We perform this clustering independently per layer, which allows the codebook to adapt to the distinct dynamic weight ranges across network depth.

3.3 Accuracy Recovery with Brief Fine-tuning

Once we have determined the mapping I and the centroids C, we can remove the meta-learner M_θ and the embeddings $\mathbf{z}$ from the network as they are no longer needed (saving more memory in addition to compression). Since the clustered weights are an approximation of the original weights, we can recover any lost performance by fine-tuning the network. The fine-tuning process consists of an identical procedure to the original training, in which we optimized the centroid codebook for the task-specific loss for β epochs, where $\beta << \alpha$ (the number of training epochs).

3.4 Storage Analysis

The compression factor relative to KAN is given by:

$$r = \frac{n|\mathbf{w}|b}{n \log_2(k) + |\mathbf{w}|kb} = \frac{nb}{n \log_2(k)/|\mathbf{w}| + kb} \tag{10}$$

As in Equation 5, n denotes the number of connections, k the number of centroids, and b the number of bits used to represent each edge. Notably, the term $|\mathbf{w}|$ in the denominator reduces the relative contribution of $n \log_2(k)$ to the overall storage. Intuitively, this happens because each centroid stores more weight information ($|\mathbf{w}|$ entries per centroid), so the overhead of indexing and storing the centroids becomes proportionally smaller. As a result, KAN benefits more from the compression scheme than MLP.

4 Experiments

We provide extensive experiments to showcase the efficacy of MetaCluster. In Section 4.1, we present the main results of the fully connected and convolutional architectures on MNIST, CIFAR10, and CIFAR100 [Deng, 2012, Krizhevsky, 2009]. Next, in Section 4.2, we provide the ablation study for our approach, covering the influence of the KAN meta-learner embedding size, basis coefficient vector sizes, and the cluster count on CIFAR10 [Krizhevsky, 2009]. In Section 4.3 we report the computational cost incurred from MetaCluster. Finally, in Section 4.4, we provide high-dimensional equation modeling results with MetaCluster. Across all experiments, our reported memory measurements reflect the full deployable footprint: per-edge indices are stored as compact integer tensors (4 bits for $k = 16$ and 8 bits for $k = 256$), centroid codebooks are stored in FP32, and basis evaluation grids are generated on the fly.

4.1 Image Classification Results

We compare our approach across 24 model schemes. The base model for each scheme consists of KAN (B-spline basis) [Liu et al., 2024b], FastKAN (RBF basis) [Li, 2024], and GramKAN (Gram polynomial basis)[Drokin, 2024]. In our naming conventions, we include *Cluster* to designate a clustered model, *Meta* to designate a model using a meta-learner, a design that originates from Zhao et al. [2025], and *Conv* to designate a model with a convolutional architecture [Bodner et al., 2024, Drokin, 2024]. This choice balances edge density against index overhead, where a compact 4-bit codebook ($k = 16$) suffices for smaller fully-connected networks, while denser convolutional models require an 8-bit codebook ($k = 256$) to maintain capacity. Each result is reported as an average of 5 runs with standard error bars. A full description of our hyperparameters is given in Appendices A.1.1 and A.1.2.

Table 1: Classification accuracy and memory (KB) comparison of a fully-connected network.

Model	MNIST		CIFAR-10		CIFAR-100	
	Memory	Acc.	Memory	Acc.	Memory	Acc.
KAN	1,031.44	95.37 ± 0.11	3,064.95	47.52 ± 0.16	3,177.45	17.80 ± 0.10
MetaKAN	141.32	96.36 ± 0.09	410.82	46.28 ± 0.39	422.08	20.17 ± 0.24
ClusterKAN	13.84	61.37 ± 2.77	38.34	27.92 ± 2.88	39.75	6.20 ± 0.31
+ fine-tune	13.84	92.66 ± 0.08	38.34	43.71 ± 0.27	39.75	11.04 ± 0.64
MetaClusterKAN	13.84	96.06 ± 0.10	38.34	45.47 ± 0.44	39.75	18.97 ± 0.15
+ fine-tune	13.84	96.41 ± 0.10	38.34	45.95 ± 0.33	39.75	19.16 ± 0.17
FastKAN	900.56	96.78 ± 0.08	2,676.83	49.10 ± 0.29	2,778.08	20.56 ± 0.09
MetaFastKAN	108.09	95.51 ± 0.12	316.35	47.40 ± 0.46	327.60	19.14 ± 0.40
ClusterFastKAN	20.54	32.34 ± 4.16	57.30	24.35 ± 0.83	58.71	6.72 ± 0.41
+ fine-tune	20.54	86.24 ± 1.08	57.30	38.89 ± 0.35	58.71	11.35 ± 0.34
MetaClusterFastKAN	20.54	94.87 ± 0.29	57.30	46.96 ± 0.44	58.71	18.66 ± 0.48
+ fine-tune	20.54	95.59 ± 0.18	57.30	47.60 ± 0.37	58.71	19.05 ± 0.47
GramKAN	497.46	96.76 ± 0.08	1,477.48	49.57 ± 0.43	1,534.43	21.89 ± 0.09
MetaGramKAN	101.49	95.48 ± 0.12	297.49	48.24 ± 0.19	309.45	19.11 ± 0.24
ClusterGramKAN	13.96	68.72 ± 1.65	38.46	31.13 ± 0.64	40.58	6.34 ± 0.43
+ fine-tune	13.96	95.40 ± 0.15	38.46	45.81 ± 0.66	40.58	13.98 ± 0.44
MetaClusterGramKAN	14.15	93.80 ± 0.72	38.65	47.66 ± 0.25	40.76	18.18 ± 0.20
+ fine-tune	14.15	95.32 ± 0.12	38.65	48.47 ± 0.30	40.76	18.96 ± 0.25

4.1.1 Fully-Connected Image Classification

Table 1 demonstrates the high classification accuracy and compression factor of fully-connected MetaCluster models across all datasets and basis functions. For example, on Cifar-10, MetaClusterKAN achieves a compression factor of up to $10.7\times$ (from 410.82KB to 38.34KB) compared with MetaKAN and up to $79.9\times$ (from 3064.95 KB to 38.34KB) compared with KAN. Furthermore, it achieves this compression factor while matching the classification accuracy of MetaKAN. Although the Meta variants can achieve high accuracy without fine-tuning, this is not the case for the non-Meta variants. For example, in the case of GramKAN, it experiences a decrease in classification accuracy of up to $3.45\times$ (from 21.89% to 6.34%), which, when recovered with fine-tuning, remains a $1.57\times$ decrease in accuracy (from 21.89% to 13.98 %). Such results demonstrate the importance of leveraging a meta-learner to learn a set of activations that lie in a lower-dimensional subspace to improve classification accuracy after clustering in fully-connected models. We also compare MetaCluster models against GPTVQ [Van Baalen et al., 2024], a popular vector quantization technique that performs hessian-aware weight sharing. From our experiments, we find that on Cifar-10, GPTVQ achieves only 31.83% classification accuracy, compared to MetaClusterKAN, which achieves 43.47%. Such results persist across all the datasets and all the models as we show in Appendix A.2.

4.1.2 Convolutional Image Classification

Table 2 demonstrates that, aside from MetaFastKANConv, the Meta variants match or exceed the classification accuracy of all the non-Meta variants at a reduced memory cost. For example, in the case of MetaKANConv on Cifar-10, the reduced memory cost is up to $4.5\times$ (from 13.7 MB to 3.1 MB) compared to KANConv. Upon clustering, we find that the compression factor increases by a factor of $7.1\times$ (from 3.1 MB to 435.34 KB). In total, compared to the KANConv, the MetaClusterKANConv achieves a reduction of up to $31.7\times$ (from 13.7 MB to 435.34 KB). For such a reduction in storage cost, after fine-tuning, we find there is a negligible impact on the downstream classification accuracy when comparing KANConv to MetaKANConv. Although the non-Meta variants recover accuracy from the fine-tuning step, the recovery still leaves them far behind their original state. For instance, upon clustering, FastKANConv experiences up to a $2.93\times$ decrease (from 76.30% to 26.01 %) in classification accuracy. Then, when recovered with fine-tuning, it still retains a $1.32\times$ decrease (from 76.30% to 57.45 %) in classification accuracy. These results again demonstrate the importance of MetaCluster for maintaining accuracy at high compression factors.

Table 2: Classification accuracy and memory (KB) comparison of a convolutional network.

Model	MNIST		CIFAR-10		CIFAR-100	
	Memory	Acc.	Memory	Acc.	Memory	Acc.
KANConv	13,634.00	99.46 ± 0.02	13,654.27	70.66 ± 2.89	13,744.62	23.99 ± 0.98
MetaKANConv	3,048.40	99.32 ± 0.05	3,052.90	68.36 ± 2.84	3,143.25	35.60 ± 1.78
ClusterKANConv	429.95	92.16 ± 1.24	430.51	33.26 ± 1.30	520.87	10.81 ± 0.54
+ fine-tune	429.95	98.89 ± 0.09	430.51	61.80 ± 0.34	520.87	22.65 ± 0.81
MetaClusterKANConv	434.77	99.29 ± 0.06	435.34	62.48 ± 2.43	525.70	29.62 ± 1.65
+ fine-tune	434.77	99.35 ± 0.05	435.34	66.85 ± 3.11	525.70	35.11 ± 1.79
FastKANConv	13,632.09	99.11 ± 0.09	13,652.37	76.30 ± 0.11	13,742.72	46.36 ± 0.38
MetaFastKANConv	3,041.57	99.00 ± 0.11	3,046.08	69.06 ± 1.15	3,136.43	39.60 ± 1.13
ClusterFastKANConv	428.03	62.61 ± 2.70	428.61	26.01 ± 3.55	518.97	5.16 ± 0.31
+ fine-tune	428.03	96.10 ± 0.09	428.61	57.45 ± 1.53	518.97	20.15 ± 0.70
MetaClusterFastKANConv	427.97	99.01 ± 0.11	428.55	68.72 ± 1.06	518.91	38.75 ± 1.04
+ fine-tune	427.97	99.08 ± 0.06	428.55	69.10 ± 1.10	518.91	39.26 ± 1.10
GramKANConv	7,586.47	99.40 ± 0.06	7,597.72	78.56 ± 1.14	7,688.07	48.25 ± 0.59
MetaGramKANConv	3,047.93	99.44 ± 0.05	3,052.43	80.91 ± 0.41	3,142.79	52.07 ± 0.63
ClusterGramKANConv	418.93	61.17 ± 9.40	419.49	12.08 ± 0.66	509.85	1.77 ± 0.29
+ fine-tune	418.93	99.27 ± 0.08	419.49	75.74 ± 0.63	509.85	44.01 ± 0.42
MetaClusterGramKANConv	418.81	99.41 ± 0.06	419.38	79.25 ± 0.59	509.73	49.70 ± 0.51
+ fine-tune	418.81	99.47 ± 0.02	419.38	80.85 ± 0.41	509.73	51.20 ± 0.67

Table 3: Impact of basis coefficient count on memory (KB) footprint and classification accuracy.

	MetaFastKAN		MetaClusterFastKAN			MetaFastKANConv		MetaClusterFastKANConv		
Coeff	Memory	Acc%	Memory	Acc%	%Chg	Memory	Acc%	Memory	Acc%	%Chg
5	316.35	47.40 ± 0.46	57.30	47.60 ± 0.37	0.42	3,046.08	69.97 ± 0.62	428.55	69.73 ± 0.63	-0.34
8	316.76	46.29 ± 0.19	57.68	46.30 ± 0.24	0.02	3,046.52	68.89 ± 0.63	440.61	68.92 ± 0.60	0.05
10	317.03	46.63 ± 0.23	57.93	47.08 ± 0.27	0.95	3,046.81	68.23 ± 0.30	448.64	68.22 ± 0.27	-0.01
15	317.71	45.26 ± 0.52	58.55	45.54 ± 0.44	0.62	3,047.54	64.87 ± 0.63	468.72	64.67 ± 0.55	-0.30
20	318.40	46.11 ± 0.20	59.18	46.03 ± 0.27	-0.17	3,048.26	61.16 ± 0.72	488.80	60.90 ± 0.75	-0.43

4.2 Ablation Study

4.2.1 Basis Coefficient Count

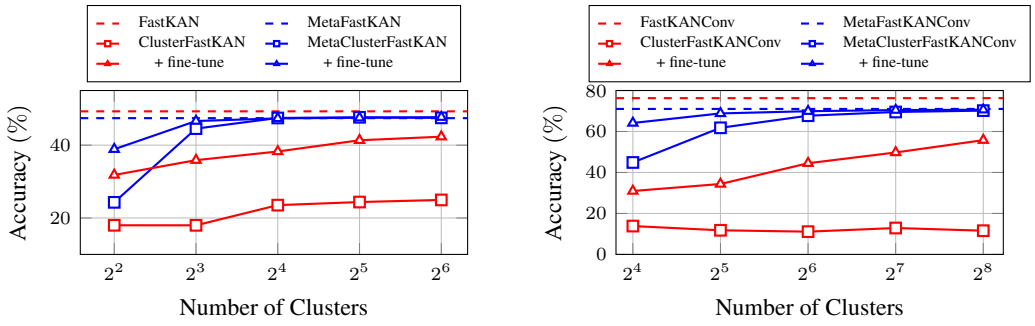
We investigate the impact of the basis coefficient count (i.e., the number of radial basis functions) on downstream clustering performance. Since our technique employs a meta-learner to learn a lower-dimensional subspace, our approach should be resilient to changes in the basis coefficients we cluster. From Table 3, we can see that the initial accuracy of MetaFastKAN and MetaFastKANConv accuracy peaks at a coefficient count of 5 before dropping. However, even with these variations in accuracy for the coefficient count, the relative percentage change remains the same. This indicates that, regardless of the number of coefficients, our MetaCluster facilitates easier clustering of weights.

4.2.2 Cluster Count

We verify that the MetaCluster framework scales well with the cluster count by comparing its cluster scaling properties with those of a non-meta KAN. As shown in Figure 2b, MetaCluster maintains accuracy well as clusters are reduced, especially after fine-tuning. For the convolutional model, decreasing the cluster count from 256 to 32 lowers MetaClusterFastKANConv accuracy by only 2%, whereas ClusterFastKANConv drops by 21.39% over the same range. Such results demonstrate our approach can achieve even greater compression, albeit with a minor loss in accuracy.

4.2.3 Meta-learner Embedding Size

The embedding size of the meta-learner directly influences the ease of clustering the meta-learner generated weights. The general trend is that as the embedding dimension increases, it becomes increasingly difficult to cluster the generated weights appropriately. While we have qualitatively demonstrated this in Figure 1, where increasing the embedding dimension makes the plot less structured, we provide a quantitative analysis of this effect in Appendix A.4.2.



(a) Fully-connected KAN variants.

(b) Convolutional KAN variants.

Figure 2: Classification accuracy vs. number of clusters for FastKAN model variants.

4.3 Computational Cost Results

As our approach is a three-stage framework consisting of a training stage, a clustering stage, and a fine-tuning stage, each stage will contribute to the computational cost during training. Furthermore, because our approach stores the KAN architecture as a codebook and indices that index the codebook, the forward inference cost also incurs computational overhead. We quantify the clustering stage cost in Table 5 and training/fine-tuning and inference costs in Table 4. Each metric is reported in seconds, averaged over 10 runs, with standard error, on an RTX 5090 GPU. The training and inference times are reported as the time to complete one epoch and the time to evaluate on the test set, respectively.

Table 5: Clustering times (in seconds)

Model	MNIST	CIFAR-10	CIFAR-100
KAN	0.029 ± 0.011	0.059 ± 0.007	0.080 ± 0.022
MetaKAN	0.024 ± 0.012	0.048 ± 0.011	0.062 ± 0.011
KANConv	3.587 ± 0.145	3.650 ± 0.098	3.542 ± 0.114
MetaKANConv	2.806 ± 0.072	2.812 ± 0.067	2.791 ± 0.092

From observing Table 5 we can see that compared to training times the clustering times are negligible (e.g., 0.029 s once vs 2.155 s per epoch). We can also see that the time required to train MetaKAN per epoch is slightly longer than KAN. Similarly, during fine-tuning (which runs for much fewer epochs than the training stage), the time per epoch is slightly longer for MetaClusterKAN (an identical architecture to ClusterKAN) than for KAN. Such a trend deviates in inference, where the MetaClusterKAN is as fast as KAN. Similar trends are seen with the convolutional models. Such results show there is minimal overhead from MetaCluster.

4.4 Equation Modeling Results

Our high-dimensional equation modeling results examined MetaCluster's ability to model 1000-dimensional equations bounded on $[-1, 1]$. All KAN models used were identical to our fully-connected image classification experiments. We explored the equations $f_1(x) = \exp\left(\frac{1}{n}\sum \sin^2\left(\frac{\pi x}{2}\right)\right)$, $f_2(x) = \sum x^2 + x^3$, $f_3(x) = \exp\left(-\frac{1}{n}\sum x^2\right)$ following a similar procedure to Zhao et al. [2025]. The clustered models used a cluster size of 4. Each reported MSE result is an average over 10 runs. We provide the complete set of hyperparameters in Appendix A.1.3.

Table 4: Training/fine-tuning and inference times (in seconds)

Model	MNIST		CIFAR-10		CIFAR-100	
	Train	Inference	Train	Inference	Train	Inference
KAN	2.155 ± 0.047	0.320 ± 0.005	2.661 ± 0.049	0.510 ± 0.008	2.730 ± 0.060	0.509 ± 0.007
MetaKAN	2.501 ± 0.055	0.337 ± 0.003	2.963 ± 0.056	0.538 ± 0.012	2.956 ± 0.096	0.526 ± 0.008
MetaClusterKAN	2.284 ± 0.048	0.323 ± 0.002	3.259 ± 0.045	0.521 ± 0.008	3.252 ± 0.091	0.542 ± 0.019
KANConv	9.158 ± 0.184	0.922 ± 0.009	10.105 ± 0.085	1.130 ± 0.003	10.370 ± 0.162	1.204 ± 0.023
MetaKANConv	10.039 ± 0.228	0.930 ± 0.009	11.024 ± 0.080	1.184 ± 0.025	10.946 ± 0.097	1.223 ± 0.011
MetaClusterKANConv	9.931 ± 0.160	0.951 ± 0.017	10.605 ± 0.144	1.149 ± 0.032	10.929 ± 0.156	1.202 ± 0.004

From Table 6, we can see that MetaClusterKAN achieves identical or, in some cases, reduced mean squared error (MSE) than the original KAN model after clustering. MetaClusterKAN can achieve these equation modeling capabilities with a 124.1x reduction in memory. This is not the case for ClusterKAN, which shows a significant increase in MSE after clustering (from 5.061×10^{-5} to 2.239×10^{-4}). These results demonstrate that the principles from MetaCluster extend beyond image classification into the equation modeling domain. Furthermore, as detailed in Appendix A.3, this structural resilience also translates to physics-informed neural networks, where MetaCluster successfully preserves the precision required to solve partial differential equations despite memory compression.

Table 6: High-dimensional equation modeling MSE and memory (KB) footprint comparison.

MODEL	MEMORY	$f_1(x)$ MSE	$f_2(x)$ MSE	$f_3(x)$ MSE
KAN	1303.6	5.06×10^{-5}	2.70×10^{-1}	5.30×10^{-7}
METAKAN	181.6	3.05×10^{-5}	2.41×10^{0}	2.76×10^{-6}
CLUSTERKAN	10.5	9.85×10^{-1}	1.66×10^{2}	1.27×10^{-1}
+ FINE-TUNE	10.5	2.24×10^{-4}	8.92×10^{-1}	7.82×10^{-6}
METACLUSTERKAN	10.5	9.03×10^{-4}	1.58×10^{0}	3.75×10^{-4}
+ FINE-TUNE	10.5	2.70×10^{-5}	5.04×10^{-1}	7.55×10^{-7}

5 Related Works

5.1 Kolmogorov-Arnold Networks

KANs have evolved rapidly, with numerous variants exploring different basis functions such as B-splines, radial basis functions (RBFs), Chebyshev and Legendre polynomials, Gram polynomials, wavelets, and rational functions [Liu et al., 2024b, Li, 2024, SS et al., 2024, Bozorgasl and Chen, 2024, Aghaei, 2024]. These architectures have demonstrated strong performance in scientific computing and equation modeling [Li et al., 2025, Wang et al., 2025, Coffman and Chen, 2025, Koenig et al., 2024] and have recently extended to computer vision through designs such as the Kolmogorov-Arnold Transformer [Yang and Wang, 2024, Raffel and Chen, 2025]. Despite these successes, KANs face persistent challenges, including training instability, computational overhead, and a substantial increase in parameter count compared to MLPs [Yu et al., 2024, Chen et al., 2024]. Our work directly addresses the memory-scaling bottleneck of KANs by creating topologically identical KANs that are smaller than comparable MLPs while maintaining accuracy.

5.2 HyperNetworks

Hypernetworks reduce trainable parameter counts by replacing task or instance-specific weights with a shared generator that predicts them. The idea originates in early meta-learning work such as Ba et al. [2016], which uses a small network to generate classifier weights from context, and Bertinetto et al. [2016], which learns to emit a one-shot tracker's parameters conditioned on a single exemplar. Ha et al. [2016] then formalized Hypernetworks for CNNs and RNNs, showing a compact hypernetwork can match the accuracy of a standard model while drastically cutting the number of directly optimized parameters. Later extensions use task embeddings to condition a single hypernetwork for multi-task or multi-objective weight generation [Savarese and Maire, 2019, Navon et al., 2020] .

MetaKANs [Zhao et al., 2025] extend this paradigm to KANs by observing that the dominant cost in a KAN is the storage of the coefficients for each univariate activation. Instead of optimizing all coefficients directly, a small meta-learner maps per-activation embeddings to basis coefficients, capturing a shared rule for weight generation across activations [Zhao et al., 2025]. In contrast, our method uses the meta-learner only during training to impose a clusterable geometry on per-edge coefficient embeddings. Then, at inference, we dispense with any hypernetwork, incurring zero runtime overhead while achieving even greater parameter efficiency.

5.3 Weight Sharing

Weight sharing compresses networks by restricting each layer's parameters to a small set of shared centroids and storing a codebook, along with per-weight indices. Scalar quantization approaches, popularized by Han et al. [2015] introduced a pipeline combining of magnitude pruning and K-means weight sharing. Subsequent theory linked quantization error to loss curvature and leveraged a

Hessian-weighted K-means to cluster [Choi et al., 2016]. Evolving from scalar quantization, vector quantization offers a natural extension by clustering weight vectors rather than individual scalars, which early work applied to improve compression–accuracy trade-offs [Gong et al., 2014]. More recently, the rise of large language models has driven specialized vector quantization approaches, which leverage Hessian information and the inherent sparsity of LLMs to optimize vector-level assignments [Liu et al., 2024a, Egiazarian et al., 2024, Van Baalen et al., 2024, Tseng et al., 2024]. We demonstrate in Appendix A.2 that such Hessian information is insufficient for KANs, a result that is in line with Hessian-aware quantization techniques reported in Fuad and Chen [2025]. Building on vector quantization research, our MetaCluster framework is the first to successfully apply weight-sharing principles to the KAN. Crucially, our approach deviates from standard vector quantization paradigms: whereas traditional methods typically cluster across the external degrees of freedom between neurons or layers, MetaCluster targets the internal degrees of freedom within the per-edge activation functions themselves.

It is worth noting that, while weight-sharing is one avenue for model compression, bit-width quantization [Fuad and Chen, 2025] is an alternative avenue, which is largely orthogonal and complementary to the proposed approach. Applying quantization on top of MetaCluster (e.g., quantizing the codebook of centroids) can further reduce the memory footprint.

6 Conclusion

We introduced MetaCluster, a compression framework that makes KANs scalable by combining meta-learned manifold shaping with weight sharing. A lightweight meta-learner maps low-dimensional embeddings to per-edge basis coefficients, thereby constraining KAN activations to lie on a compact manifold amenable to clustering. We then apply K-means in coefficient space, replace per-edge parameters with codebook centroids and compact indices, discard the meta-learner, and briefly fine-tune centroids to recover any loss. This design directly targets the dimensionality driver of KAN memory, yielding a storage advantage that grows with the number of coefficients per edge.

Evaluation shows that MetaCluster achieves up to $80\times$ and $124.1\times$ reductions in parameter storage for image classification and equation modeling, respectively, with competitive accuracy compared to uncompressed baselines. While this work validates the framework on standard benchmarks, establishing formal theoretical guarantees and extending these storage gains to large-scale architectures remain promising directions for future research. Ultimately, MetaCluster facilitates the deployment of KANs in resource-constrained environments, offering a path toward more accessible AI without introducing unique ethical risks.

References

Alireza Afzal Aghaei. rkan: Rational kolmogorov-arnold networks. *arXiv preprint arXiv:2406.14495*, 2024.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. *Advances in neural information processing systems*, 29, 2016.

Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

Alexander Dylan Bodner, Antonio Santiago Tepsich, Jack Natan Spolski, and Santiago Pourteau. Convolutional kolmogorov-arnold networks. *arXiv preprint arXiv:2406.13155*, 2024.

Zavareh Bozorgasl and Hao Chen. Wav-kan: Wavelet kolmogorov-arnold networks. *arXiv preprint arXiv:2405.12832*, 2024. URL `https://arxiv.org/abs/2405.12832`.

Ziwen Chen, Gundavarapu, and WU DI. Vision-kan: Exploring the possibility of kan replacing mlp in vision transformer. `https://github.com/chenziwenhaoshuai/Vision-KAN.git`, 2024.

Minsik Cho, Keivan A Vahid, Saurabh Adya, and Mohammad Rastegari. Dkm: Differentiable k-means clustering layer for neural network compression. *arXiv preprint arXiv:2108.12659*, 2021.

Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Towards the limit of network quantization. *arXiv preprint arXiv:1612.01543*, 2016.

Cale Coffman and Lizhong Chen. Matrixkan: Parallelized kolmogorov-arnold network. *arXiv preprint arXiv:2502.07176*, 2025.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.

David L Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture*, 1(2000):32, 2000.

Ivan Drokin. Kolmogorov-arnold convolutions: Design principles and empirical studies. *arXiv preprint arXiv:2407.01092*, 2024.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

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.

Kazi Ahmed Asif Fuad and Lizhong Chen. Quantkan: A unified quantization framework for kolmogorov arnold networks. *arXiv preprint arXiv:2511.18689*, 2025.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Benjamin C Koenig, Suyong Kim, and Sili Deng. Kan-odes: Kolmogorov–arnold network ordinary differential equations for learning dynamical systems and hidden physics. *Computer Methods in Applied Mechanics and Engineering*, 432:117397, 2024.

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

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

Chenxin Li, Xinyu Liu, Wuyang Li, Cheng Wang, Hengyu Liu, Yifan Liu, Zhen Chen, and Yixuan Yuan. U-kan makes strong backbone for medical image segmentation and generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 4652–4660, 2025.

Ziyao Li. Kolmogorov-arnold networks are radial basis function networks. *arXiv preprint arXiv:2405.06721*, 2024.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

Yifei Liu, Jicheng Wen, Yang Wang, Shengyu Ye, Li Lyna Zhang, Ting Cao, Cheng Li, and Mao Yang. Vptq: Extreme low-bit vector post-training quantization for large language models. *arXiv preprint arXiv:2409.17066*, 2024a.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342, 1993. ISSN 0098-3004. doi: https://doi.org/10.1016/0098-3004(93)90090-R. URL https://www.sciencedirect.com/science/article/pii/009830049390090R.

Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. Learning the pareto front with hypernetworks. *arXiv preprint arXiv:2010.04104*, 2020.

Matthew Raffel and Lizhong Chen. Flashkat: Understanding and addressing performance bottlenecks in the kolmogorov-arnold transformer. *arXiv preprint arXiv:2505.13813*, 2025.

Pedro Savarese and Michael Maire. Learning implicitly recurrent cnns through parameter sharing. *arXiv preprint arXiv:1902.09701*, 2019.

Sidharth SS, Keerthana AR, Anas KP, et al. Chebyshev polynomial-based kolmogorov-arnold networks: An efficient architecture for nonlinear function approximation. *arXiv preprint arXiv:2405.07200*, 2024.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*, 2024.

Mart Van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*, 2024.

Yizheng Wang, Jia Sun, Jinshuai Bai, Cosmin Anitescu, Mohammad Sadegh Eshaghi, Xiaoying Zhuang, Timon Rabczuk, and Yinghua Liu. Kolmogorov–arnold-informed neural network: A physics-informed deep learning framework for solving forward and inverse problems based on kolmogorov–arnold networks. *Computer Methods in Applied Mechanics and Engineering*, 433: 117518, 2025.

Xingyi Yang and Xinchao Wang. Kolmogorov-arnold transformer. *arXiv preprint arXiv:2409.10594*, 2024.

Runpeng Yu, Weihao Yu, and Xinchao Wang. Kan or mlp: A fairer comparison. *arXiv preprint arXiv:2407.16674*, 2024.

Zhangchi Zhao, Jun Shu, Deyu Meng, and Zongben Xu. Improving memory efficiency for training kans via meta learning. *arXiv preprint arXiv:2506.07549*, 2025.

A Appendix

A.1 Hyperparameters

A.1.1 Image Classification Fully-Connected Setup

For all image classification experiments, we did a 90/10 training/validation split of the training data. Our fully-connected architecture follows Zhao et al. [2025] in that we stack two fully-connected KAN layers, with a 32-dimensional hidden state between them. Each KAN variant uses a SiLU activation [Elfwing et al., 2018]. The meta-learner variants contained a meta-learner with a hidden dimension of 32. Concretely:

- **KAN:** degree–3 B-splines, grid range $[-1, 1]$, grid size 5 [Liu et al., 2024b].
- **FastKAN:** 8 RBFs over $[-2, 2]$ [Li, 2024].
- **GramKAN:** degree–3 polynomial [Drokin, 2024].

We train with AdamW [Loshchilov and Hutter, 2017] and for at most 50 epochs, with early stopping (patience = 10). After the final epoch, we cluster the learned weights into 16 groups, then fine-tune for an additional 5 epochs.For our fully-connected architecture experiments Table 7 reports the full set of hyperparameters for our KAN variants and Table 8 reports the full set of hyperparameters for our MetaKAN variants.

Table 7: Hyperparameters for fully-connected KAN variants.

Hyperparameter	KAN	FastKAN	GramKAN
Hidden dimension	32	32	32
Activation	SiLU	SiLU	SiLU
Degree	3 (Spline)	-	3
Grid range	$[-1, 1]$	$[-2, 2]$	-
Grid size	5	8(Num grids)	-
Optimizer	AdamW	AdamW	AdamW
Learning rate	1×10^{-4}	1×10^{-3}	1×10^{-3}
Learning rate for finetuning (lr_c)	1×10^{-4}	1×10^{-4}	1×10^{-4}
Batch size	128	128	128
Epochs	50	50	50
Early stopping patience	10	10	10
Clustered training epochs	5	5	5
Early stopping patience (fine-tuning)	3	3	3
Number of clusters	16	16	16

Table 8: Hyperparameters for fully-connected MetaKAN variants.

Hyperparameter	MetaKAN	MetaFastKAN	MetaGramKAN
Hidden dimension	32	32	32
Embedding dimension	1	1	1
Activation	SiLU	SiLU	SiLU
Degree	3 (Spline)	-	3
Grid range	$[-1, 1]$	$[-2, 2]$	-
Grid size	5	8(Num-grids)	-
Optimizer set	double	double	double
Optimizer	AdamW	AdamW	AdamW
Learning rate for meta-learner (lr_h)	5×10^{-4}	1×10^{-3}	1×10^{-4}
Learning rate for embeddings (lr_c)	5×10^{-3}	1×10^{-2}	1×10^{-3}
Learning rate for finetuning (lr_c)	1×10^{-4}	1×10^{-4}	1×10^{-4}
Batch size	128	128	128
Epochs	50	50	50
Clustered training epochs	5	5	5
Early stopping patience	10	10	10
Early stopping patience (fine-tuning)	3	3	3
Number of clusters	16	16	16

A.1.2 Image Classification Convolutional Setup

For all image classification experiments, we did a 90/10 training/validation split of the training data. We apply data augmentation to the train set for the convolutional architecture, following an identical procedure to Drokin [2024]. Our convolutional architecture is taken from Drokin [2024]:

four convolutional layers with channel progression $[32, 64, 128, 512]$, each using 3×3 kernels, stride 1, and padding 1. As with the fully-connected experiments, each KAN variant uses a SiLU activation [Elfwing et al., 2018] and the meta-learner versions use a meta-learner with a hidden dimension of 32. We replace the usual activation with our kernels:

- **KANConv:** degree–3 B-splines, grid range $[-3, 3]$, grid size 5 [Liu et al., 2024b].
- **FastKANConv:** 8 RBFs on $[-3, 3]$ [Li, 2024].
- **GramKANConv:** degree–3 polynomial [Drokin, 2024].

Training again uses AdamW for up to 150 epochs with early stopping (patience = 10). Final weights are clustered into 16 groups and fine-tuned for 5 epochs. For our convolutional architecture experiments Table 9 reports the full set of hyperparameters for our KAN variants and Table 10 reports the full set of hyperparameters for our MetaKAN variants.

Table 9: Hyperparameters for convolutional non-meta KAN variants.

Hyperparameter	KAN	FastKAN	KAGN
Hidden dimension	[32, 64, 128, 256]	[32, 64, 128, 256]	[32, 64, 128, 256]
Activation	SiLU	SiLU	SiLU
Degree	3 (Spline)	-	3
Grid range	$[-3, 3]$	$[-3, 3]$	-
Grid size	5	8(Num-Grids)	-
Dropout	0.25	0.25	0.25
Dropout (linear layers)	0.5	0.5	0.5
Optimizer	AdamW	AdamW	AdamW
Learning rate for finetuning (lr_c)	1×10^{-5}	1×10^{-5}	1×10^{-5}
learning rate (lr)	1×10^{-3}	1×10^{-3}	1×10^{-3}
Batch size	128	128	128
Epochs	150	150	150
Clustered training epochs	5	5	5
Early stopping patience	10	10	10
Early stopping patience (fine-tuning)	3	3	3
Number of clusters	256	256	256
Convolution Groups	1	1	1

Table 10: Hyperparameters for convolutional MetaKAN variants.

Hyperparameter	MetaKAN	MetaFastKAN	MetaKAGN
Hidden dimension	[32, 64, 128, 256]	[32, 64, 128, 256]	[32, 64, 128, 256]
Embedding dimension	2	2	1
Activation	SiLU	SiLU	SiLU
Degree	3 (Spline)	-	3
Grid range	$[-3, 3]$	$[-3, 3]$	-
Grid size	5	8(Num-grid)	-
Optimizer set	double	double	double
Optimizer	AdamW	AdamW	AdamW
Learning rate for meta-learner (lr_h)	1×10^{-4}	1×10^{-4}	1×10^{-4}
Learning rate embedding (lr_e)	5×10^{-3}	5×10^{-3}	5×10^{-3}
Learning rate for finetuning (lr_c)	1×10^{-5}	1×10^{-5}	1×10^{-5}
Global learning rate (lr)	1×10^{-3}	5×10^{-3}	5×10^{-3}
Batch size	128	128	128
Epochs	150	150	150
Clustered training epochs	5	5	5
Early stopping patience	10	10	10
Early stopping patience (fine-tuning)	3	3	3
Number of clusters	256	256	256
Embedding scheduler	Yes	Yes	Yes
Hypernet scheduler	Yes	Yes	Yes
Dropout	0.25	0.25	0.25
Dropout (linear layers)	0.5	0.5	0.5
Convolution Groups	1	1	1

A.1.3 Equation Modeling Setup

We report our complete set of hyperparameter for our equation modeling experiments in Table 11.

Table 11: Hyperparameters for fully-connected KAN equation modeling experiments.

HYPERPARAMETER	KAN
HIDDEN DIMENSION	32
ACTIVATION	SILU
DEGREE	3 (SPLINE)
GRID RANGE	$[-1, 1]$
GRID SIZE	5
OPTIMIZER	ADAMW
LEARNING RATE	1×10^{-4}
LEARNING RATE FOR FINETUNING (LR_C)	1×10^{-4}
BATCH SIZE	256
EPOCHS	100
CLUSTERED TRAINING EPOCHS	10
NUMBER OF CLUSTERS	4
DOMAIN OF DATAPOINTS	[-1,1]
TRAIN SET SAMPLES	50,000
TEST SET SAMPLES	20,000

A.2 MetaCluster vs. GPTVQ

To further validate our approach, we benchmark MetaCluster against GPTVQ [Van Baalen et al., 2024], a state-of-the-art Hessian-aware vector quantization technique, across fully-connected KAN, FastKAN, and GramKAN architectures. As detailed in Table 12, MetaCluster consistently demonstrates a superior accuracy-compression trade-off at identical memory footprints. Most notably, MetaCluster preserves structural integrity immediately after clustering, whereas GPTVQ suffers from catastrophic performance degradation. For example, MetaClusterFastKAN on MNIST retains 94.87% accuracy before fine-tuning, whereas GPTVQ collapses to 30.80%. Although a brief fine-tuning stage helps GPTVQ recover a significant portion of its lost accuracy, it ultimately fails to match MetaClusterFastKAN's final performance, particularly on more complex datasets like CIFAR-100, where the fine-tuned MetaClusterFastKAN achieves 19.05% accuracy compared to GPTVQ's 11.68%. These empirical results indicate that Hessian-aware clustering is suboptimal for the high-dimensional activation coefficient space of KANs, and that explicitly constraining weights to a low-dimensional manifold via MetaCluster provides a fundamentally better initialization for weight sharing, avoiding severe accuracy loss.

Table 12: Classification accuracy and memory (KB) comparison of a fully-connected network between GPTVQ and MetaCluster.

MODEL	MNIST		CIFAR-10		CIFAR-100	
	MEMORY	ACC.	MEMORY	ACC.	MEMORY	ACC.
METACLUSTERKAN	13.84	96.06 ± 0.10	38.34	45.47 ± 0.44	39.75	18.97 ± 0.15
+ FINE-TUNE	13.84	96.41 ± 0.10	38.34	45.95 ± 0.33	39.75	19.16 ± 0.17
GPTVQ	13.84	80.96 ± 2.60	38.34	31.38 ± 3.73	39.75	3.49 ± 0.28
+ FINE-TUNE	13.84	92.61 ± 0.28	38.34	43.47 ± 0.13	39.75	11.21 ± 0.50
METACLUSTERFASTKAN	20.54	94.87 ± 0.29	57.30	46.96 ± 0.44	58.71	18.66 ± 0.48
+ FINE-TUNE	20.54	95.59 ± 0.18	57.30	47.60 ± 0.37	58.71	19.05 ± 0.47
GPTVQ	20.54	30.80 ± 2.24	57.30	24.57 ± 0.93	58.71	7.13 ± 0.31
+ FINE-TUNE	20.54	87.16 ± 0.61	57.30	$39.10 \pm 0.0.32$	58.71	11.68 ± 0.32
METACLUSTERGRAMKAN	14.15	93.80 ± 0.72	38.65	47.66 ± 0.25	40.76	18.18 ± 0.20
+ FINE-TUNE	14.15	95.32 ± 0.12	38.65	48.47 ± 0.30	40.76	18.96 ± 0.25
GPTVQ	13.96	82.81 ± 2.23	38.46	32.44 ± 1.58	40.58	7.49 ± 0.56
+ FINE-TUNE	13.96	95.44 ± 0.14	38.46	45.48 ± 0.57	40.58	14.02 ± 0.63

A.3 Partial Differential Equations

To evaluate the compressibility of KAN and MetaKAN in the physics-informed setting, we train both architectures as PINNs on four canonical PDEs with known analytical solutions: the heat equation ($\partial_t u = 0.1\,\partial_{xx} u$), the wave equation ($\partial_{tt} u = \partial_{xx} u$), the advection equation ($\partial_t u + \partial_x u = 0$), and the Laplace equation ($\partial_{xx} u + \partial_{yy} u = 0$), all defined on $[0, 1]^2$ with Dirichlet boundary conditions. We enforce boundary and initial conditions exactly using the hard constraint formulation [Lagaris et al., 1998], expressing the solution as $u = B + D \cdot N$ where B satisfies the prescribed conditions,

D vanishes on constrained boundaries, and N is the network output. Specifically, these exact constraints are formulated as $u(x,t) = \sin(\pi x) + x(1-x)t \cdot N(x,t)$ for the heat equation, $u(x,t) = \sin(\pi x)(1-t^2/2) + x(1-x)t^2 \cdot N(x,t)$ for the wave equation, $u(x,t) = \sin(\pi(x-t)) + xt \cdot N(x,t)$ for the advection equation, and $u(x,y) = y\sin(\pi x) + x(1-x)y(1-y) \cdot N(x,y)$ for the Laplace equation. By satisfying the initial and boundary conditions by construction, this approach reduces the training objective to the PDE residual alone.

Both models use a KAN architecture with layers $[2, 64, 64, 1]$, B-spline order k=3, grid size G=5, and grid range $[-1, 1]$. Training consists of 10,000 AdamW epochs with cosine annealing followed by 500 L-BFGS refinement steps on 3,000 fixed collocation points [Loshchilov and Hutter, 2017, Liu and Nocedal, 1989]. For MetaKAN, we use a dual-optimizer setup with separate learning rates for the MetaNet (10^{-3}) and embeddings (10^{-3}). After training, we compress each model by clustering the per-edge spline coefficient vectors (dimension $G+k+1$=9) into 16 centroids via K-means. We evaluate the L^2 relative error on a 100×100 grid at three stages: before compression, after clustering (no fine-tuning), and after fine-tuning the cluster centroids for 5,000 epochs.

Table 13: L2 metric and memory (KB) comparison for PDE modeling (clusters=16).

Model	Heat Equation		Wave Equation		Advection		Laplace	
	Memory	L2 Error	Memory	L2 Error	Memory	L2 Error	Memory	L2 Error
KAN	150.80	7.29×10^{-6}	150.80	2.85×10^{-5}	150.80	7.32×10^{-6}	150.80	3.00×10^{-6}
ClusterKAN	3.80	1.61×10^{-1}	3.80	2.79×10^{-1}	3.80	4.91×10^{-2}	3.80	5.87×10^{-2}
+ fine-tune	3.80	1.81×10^{-4}	3.80	1.04×10^{-3}	3.80	6.11×10^{-5}	3.80	1.18×10^{-4}
MetaKAN	21.00	3.19×10^{-5}	21.00	9.03×10^{-5}	21.00	3.92×10^{-6}	21.00	8.37×10^{-6}
MetaClusterKAN	3.80	9.51×10^{-3}	3.80	5.52×10^{-3}	3.80	5.28×10^{-3}	3.80	5.97×10^{-3}
+ fine-tune	3.80	2.26×10^{-5}	3.80	1.05×10^{-4}	3.80	3.48×10^{-6}	3.80	1.82×10^{-5}

The results on the PDEs demonstrate that MetaCluster enables extreme memory compression without sacrificing the high-fidelity precision required for physics-informed neural networks. While ClusterKAN successfully reduces the model's memory footprint by nearly 40× (from 150.80 KB down to 3.80 KB), it causes severe and irreversible degradation to predictive performance. For example, on the Heat equation, the fine-tuned ClusterKAN only recovers an L^2 error of 1.81×10^{-4}, which is significantly worse than the baseline KAN's 7.29×10^{-6}. In contrast, after fine-tuning, MetaClusterKAN achieves the exact same 3.80 KB compressed footprint but successfully recovers baseline accuracy across all evaluated equations, even slightly outperforming the baseline KAN on the Advection equation (3.48×10^{-6} compared to 7.32×10^{-6}). This confirms that the proposed manifold-shaping approach is not just beneficial but fundamentally essential for extending effective weight sharing to scientific modeling tasks.

A.4 Ablations Extended

We present this extended ablation study to provide a more detailed analysis of the impact of various hyperparameters on accuracy and the effectiveness of fine-tuning across different settings. These extended results provide further insight into how variations in embedding size and coefficient count, along with fine-tuning, influence performance differences in clustering across the different KAN model variants and network architectures.

A.4.1 Basis Coefficient Count Extended

Table 14 provides detailed insight into the influence of the fine-tuning stage of the MetaCluster framework on different basis coefficient counts (i.e. the grid size or number of radial basis functions). We observe that because the majority of the original classification accuracy is preserved immediately after clustering, the fine-tuning stage yields only marginal additional gains.

A.4.2 Meta-learner Embedding Size Extended

Our ablation exploring the influence of the embedding dimension aims to quantify the influence of the embedding size on the downstream clustering performance. We report these results in Table 15. From Table 15, we can see that in both the fully-connected and convolutional architectures, as the embedding dimension increases, the classification accuracy after clustering decreases. This validates

Table 14: Detailed results of grid sizes versus accuracy and memory for both fully-connected and convolutional MetaFastKAN networks on Cifar-10.

Model	Grid Size	Grid Range	Embed Dim	# Clusters	Model Accuracy	Clustered Accuracy	Fine-tuned Accuracy	Mem Before Clustering (KB)	Mem After Clustering (KB)
MetaClusterFastKAN	5	-2,2	1	16	47.41	47.46	47.35	316.35	57.30
MetaClusterFastKAN	8	-2,2	1	16	47.62	46.96	47.80	316.76	57.68
MetaClusterFastKAN	10	-2,2	1	16	46.91	47.04	47.49	317.03	57.93
MetaClusterFastKAN	15	-2,2	1	16	45.88	45.15	45.74	317.71	58.55
MetaClusterFastKAN	20	-2,2	1	16	43.57	43.75	44.31	318.40	59.18
MetaClusterFastKANConv	5	-3,3	2	256	71.42	71.06	71.38	3,046.08	428.55
MetaClusterFastKANConv	8	-3,3	2	256	69.61	68.66	69.24	3,046.52	440.61
MetaClusterFastKANConv	10	-3,3	2	256	66.78	66.58	66.74	3,046.81	448.64
MetaClusterFastKANConv	15	-3,3	2	256	64.24	63.99	64.33	3,047.54	468.72
MetaClusterFastKANConv	20	-3,3	2	256	54.41	54.45	54.48	3,048.26	488.80

our assumption, which we developed based on Figure 1, that finding a lower-dimensional subspace improves downstream clustering performance. During the fine-tuning stage, we can recover most of the accuracy decrease experienced by choosing a higher-dimensional embedding. Furthermore, the higher-dimensional choice of embedding does not affect the MetaCluster model memory footprint.

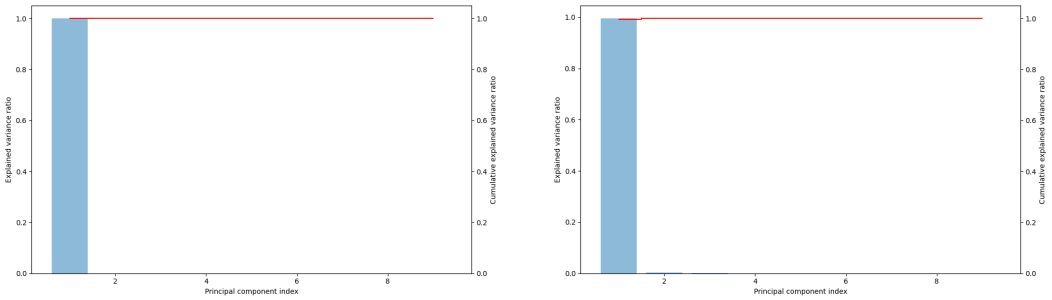
Table 15: Detailed results of embedding dimension versus accuracy for both fully-connected and convolutional MetaFastKAN networks on Cifar-10.

Model	Num-grid	Grid Range	Embed Dim	# Clusters	Model Accuracy	Clustered Accuracy	Fine-tuned Accuracy	Mem Before Clustering (KB)	Mem After Clustering (KB)
MetaClusterFastKAN	5	-2,2	4	16	48.66	38.28	44.63	1,202.50	57.30
MetaClusterFastKAN	5	-2,2	3	16	47.28	39.11	45.20	907.10	57.30
MetaClusterFastKAN	5	-2,2	2	16	48.19	45.48	47.29	611.72	57.30
MetaClusterFastKANConv	5	-3,3	4	256	71.16	66.46	69.58	6,077.09	428.55
MetaClusterFastKANConv	5	-3,3	3	256	73.78	70.21	72.39	4,561.59	428.55
MetaClusterFastKANConv	5	-3,3	2	256	71.42	71.05	71.33	3,046.08	428.55

A.5 Dimensionality and Clusterability Analysis

A.5.1 PCA for Lower Dimensional Manifold

To empirically verify that the meta-learner successfully restricts the KAN activation coefficients to a low-dimensional manifold, we perform Principal Component Analysis (PCA) on the learned edge functions. By examining the explained variance ratio of the principal components, we can quantify the effective dimensionality of the coefficient space. As detailed in Section 3.1, the meta-learner maps a low-dimensional embedding ($z_i \in \mathbb{R}^{d_{emb}}$) to the full set of KAN weights, constraining them to a task-aligned manifold. For our fully-connected MetaKAN variant on CIFAR-10, we utilize an embedding dimension of $d_{emb} = 1$.



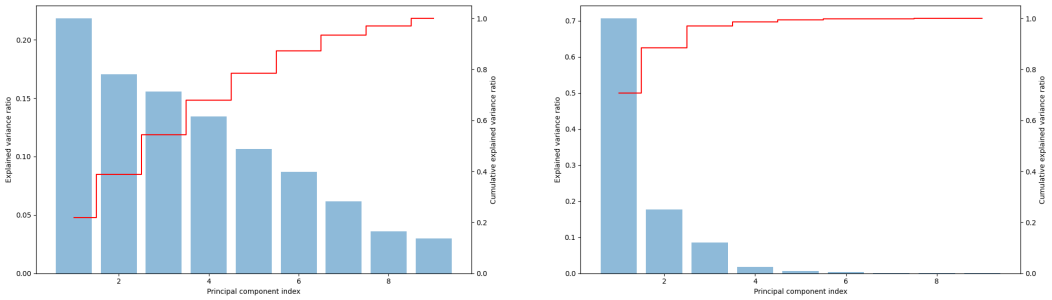
(a) Layer 1 MetaKAN PCA plot.

(b) Layer 2 MetaKAN PCA plot.

Figure 3: PCA explained variance ratio for the (a) first and (b) second layers of the fully-connected MetaKAN model on CIFAR-10. The single dominant bar demonstrates that the variance is almost entirely explained by the first principal component, confirming the effective restriction of coefficients to a 1D manifold via the $d_{emb} = 1$ constraint.

Figure 3 displays the PCA explained variance ratio for the first and second layers of this fully-connected MetaKAN model. As expected, nearly 100% of the variance in the generated weights is captured by the very first principal component. This confirms that the meta-learner effectively enforces a one-dimensional manifold, making the resulting high-dimensional coefficient vectors highly compressible and amenable to K-means clustering.

In contrast, Figure 4 illustrates the PCA explained variance ratio for the baseline KAN model trained without a meta-learner. Without the structural bottleneck imposed by the low-dimensional embedding, the variance of the learned coefficients is spread across multiple principal components. The cumulative explained variance (shown by the red line) takes several components to approach 1.0. This lack of a coherent low-dimensional structure fundamentally explains why standard clustering methods struggle to achieve high compression rates without significant accuracy degradation in baseline KANs.



(a) Layer 1 KAN PCA plot.

(b) Layer 2 KAN PCA plot.

Figure 4: PCA explained variance ratio (blue bars) and cumulative explained variance (red line) for the (a) first and (b) second layers of the baseline fully-connected KAN model on CIFAR-10. The gradual accumulation of variance across multiple components highlights the absence of a distinct low-dimensional manifold when a meta-learner is not utilized.

A.5.2 Quantitative Topological Evaluation

To further confirm that the meta-learner successfully shapes the learned edge coefficients into a more clusterable space, we evaluate standard cluster validation metrics on the first-layer coefficient vectors on CIFAR-10 ($k = 16$):

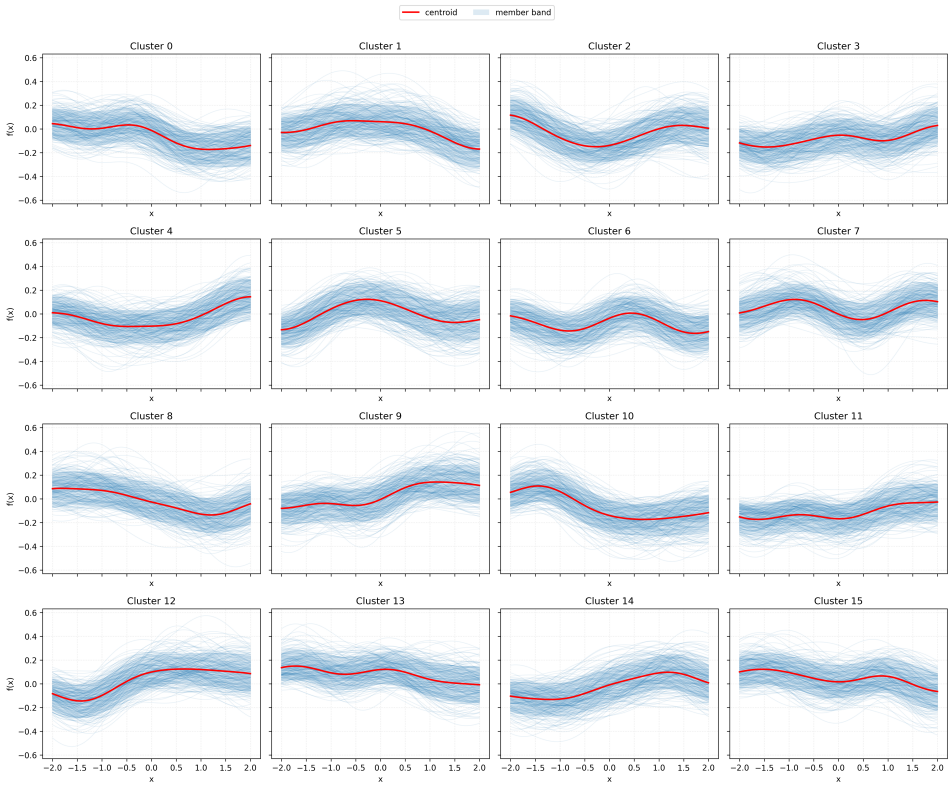
Table 16: Clustering quality metrics for KAN coefficient vectors on CIFAR-10.

MODEL	SILHOUETTE (↑)	CALINSKI-HARABASZ (↑)	DAVIES-BOULDIN (↓)
KAN	0.08	4,678.13	1.82
METACLUSTERKAN	**0.52**	**359,394.87**	**0.52**

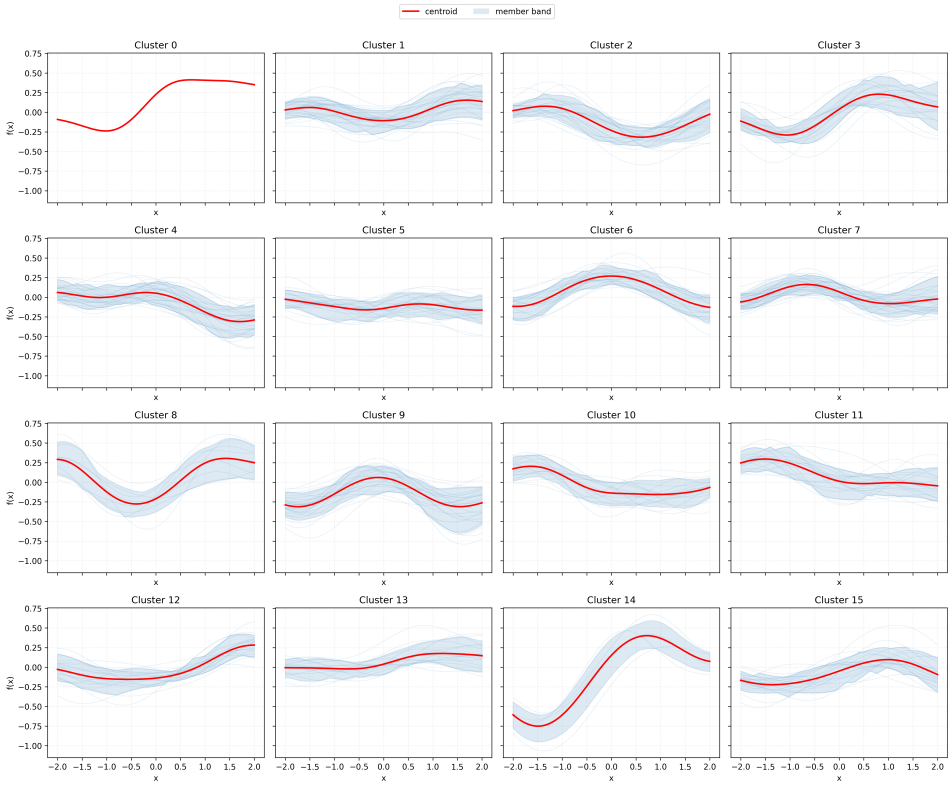
As shown in Table 16, using the meta-learner increases the Silhouette Coefficient from 0.08 to 0.52 and substantially increases the Calinski-Harabasz Index, while reducing the Davies-Bouldin Index from 1.82 to 0.52. These results quantitatively support our PCA findings, demonstrating that constraining weights to a lower-dimensional manifold yields tighter, better-separated clusters that make downstream K-means clustering far more effective.

A.6 Visualizing Clusters

We verify the importance of MetaCluster qualitatively by plotting edge functions and the respective centroids associated with the first layer of FastKAN and MetaFastKAN in Figures 5 and 6, respectively. From observing the functions associated with FastKAN and MetaFastKAN, it is visually clear that the functions associated with MetaFastKAN are much closer to the respective centroid than in FastKAN. Such a result demonstrates the importance of using meta-learners to find a lower-dimensional functional space before clustering.

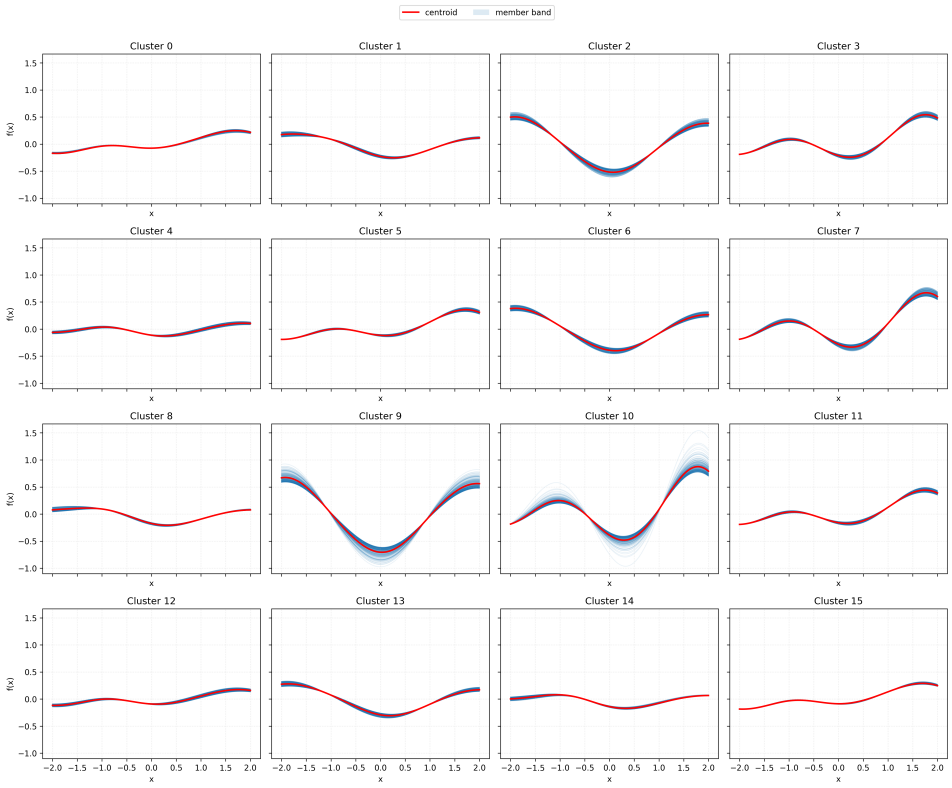


(a) The set of clusters showcasing the edge functions for each cluster on the first layer.

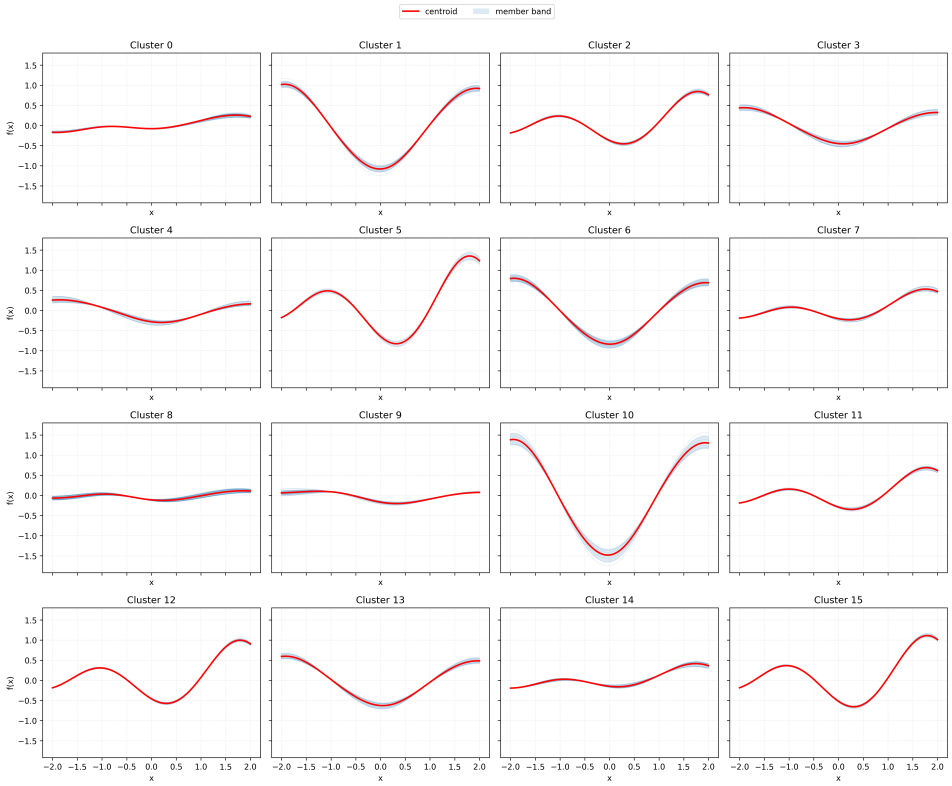


(b) The set of clusters showcasing the edge functions for each cluster on the second layer.

Figure 5: The edge functions for each cluster (in blue) and the centroid functions (in red) for the (a) first and (b) second layers of the fully-connected FastKAN models on Cifar-10.



(a) The set of clusters showcasing the edge functions for each cluster on the first layer.



(b) The set of clusters showcasing the edge functions for each cluster on the second layer.

Figure 6: The edge functions for each cluster (blue) and the centroid functions (red) for the (a) first and (b) second layers of the fully-connected MetaFastKAN models on Cifar-10.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Sections 3 and 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: [N/A]

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4 and Appendix A.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code is publicly available at `https://github.com/OSU-STARLAB/MetaCluster`.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix A.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix 4.3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research follows the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section 6.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Section 4 and Appendix A.1.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: [N/A]

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.