METACLUSTER: ENABLING DEEP COMPRESSION OF KOLMOGOROV-ARNOLD NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Kolmogorov-Arnold Networks (KANs) replace scalar weights with per-edge vectors of basis coefficients, thereby boosting expressivity and accuracy but at the same time resulting in a multiplicative increase in parameters and memory. We propose MetaCluster, a framework that makes KANs highly compressible without sacrificing accuracy. Specifically, a lightweight meta-learner, trained jointly with the KAN, is used to map low-dimensional embedding to coefficient vectors, shaping them to lie on a low-dimensional manifold that is amenable to clustering. We then run K-means in coefficient space and replace per-edge vectors with shared centroids. Afterwards, 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 no loss in accuracy. Code will be released upon publication.

1 INTRODUCTION

Kolmogorov–Arnold Networks (KANs) have recently emerged as a compelling alternative to multi-layer perceptrons (MLPs), delivering strong results in equation modeling and scientific machine learning, often with improved task performance at comparable or lower parameter counts in those settings (Liu et al., 2024; Li et al., 2025; Coffman & Chen, 2025; Koenig et al., 2024). Recently, KANs have also begun to show promise in computer vision (Yang & Wang, 2024; Raffel & Chen, 2025). However, unlike equation modeling, at larger scales, KANs frequently incur a substantial parameter overhead relative to MLPs (Yu et al., 2024). This overhead stems from KANs per-edge degrees of freedom: 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 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 grow but also concentrate (nearest and farthest become similar). With such an effect brought by the curse of dimensionality, typical clustering methods struggle to form tight clusters (Beyer et al., 1999; Donoho et al., 2000).

We address this challenge with MetaCluster, a three-stage compression framework that merges meta-learning with weight sharing. First, a small meta-learner maps low-dimensional embeddings into per-edge coefficient vectors, constraining KAN activations to a low-dimensional manifold while 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 accuracy loss. 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., 2024) 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., 2024; Li, 2024; SS et al., 2024). To further verify the robustness of our approach, we validate it across MNIST (Deng, 2012), CIFAR-10, and CIFAR-100 (Krizhevsky, 2009). From our experiments, we find that MetaCluster achieves a reduction of up to $80\times$ in parameter storage relative to the uncompressed KAN without degrading accuracy, 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 experiments demonstrating up to $80\times$ memory reduction with no loss in accuracy, along with extensive ablations.

2 Preliminaries and Motivation

2.1 Kolmogorov-Arnold Network

KANs have gained attention as an alternative to conventional MLPs (Liu et al., 2024). 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. (2024) demonstrates a practical generalization that permits arbitrary width and depth tailored to the task. We can express such a formulation with L layers as

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

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 is equipped with a learnable univariate activation $\phi_{l,j,i}(\cdot)$, for $i \in [1, n_l]$ and $j \in [1, n_{l+1}]$.

The most common choice for implementing $\phi_{l,j,i}(\cdot)$ has been through a weighted summation of basis functions $B_i(\cdot)$ represented as,

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

In Equation 3, the weighted summation of basis functions is parameterized with learnable coefficients, $\mathbf{w} = [w_1, ..., w_{|\mathbf{w}|}]$.

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 & Chen, 2025; Koenig et al., 2024) and has even started to extend its reach into computer vision (Raffel & Chen, 2025; Yang & 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 Motivations and Challenges of Weight Sharing for KAN

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 to save substantial storage for 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 the KAN activations, we require a fundamentally different method for transforming the high-dimensional vector into a more manageable space.

3 Methods

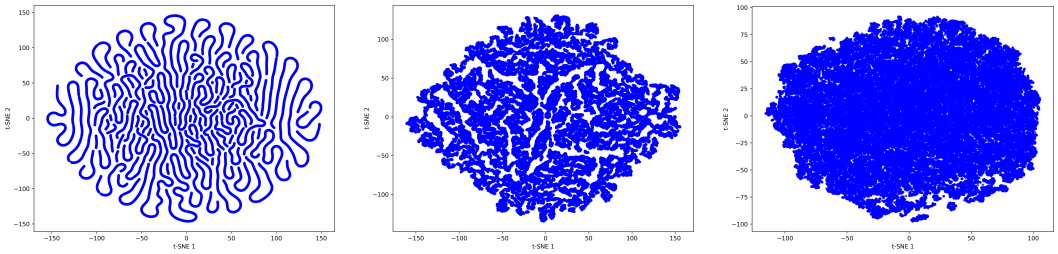
This section presents MetaCluster, a three-stage framework that makes per-edge KAN activation coefficients amenable to clustering and compression. First, we train a single meta-learner that turns compact embeddings into full activation-coefficient vectors, forcing them onto a task-aligned low-dimensional manifold. Next, we run K-means on those vectors and record, for each edge, which centroid it belongs to 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.

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.



(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.

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 $|\mathbf{w}| = G + k + 1$ with $G = 5$ and $k = 3$, yielding $|\mathbf{w}| = 9$. 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, indicating that the meta-learner concentrates variability onto task-relevant directions. In contrast, the baseline KAN provided in Figure 1c yields a diffuse cloud without coherent low-dimensional structure, which hampers downstream clustering. These visualizations substantiate our premise: shaping coefficient vectors via a low-dimensional embedding makes them markedly more clusterable, a property we later translate into the higher compression at fixed accuracy shown in Section 4.3.

3.2 KAN Activation Compression with K-means 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 $M_\theta(\mathbf{z}_j) \mapsto \mathbf{c}_{I_j}$.

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 a standard MLP.

4 Experiments

To evaluate the capabilities of MetaCluster, we conduct extensive experiments and an ablation study on both fully-connected and convolutional architectures. In Section 4.1, we report the experimental setup, and in Section 4.2, we report the results of the fully-connected and convolutional architecture on MNIST, CIFAR10, and CIFAR100 (Deng, 2012; Krizhevsky, 2009). Then, in Section 4.3, 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).

4.1 Experimental Setup

For all our 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). We compare our approach across 24 model schemes. The base model for each scheme consists of KAN (B-spline basis) (Liu et al., 2024), 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).

4.1.1 Fully-Connected Setup

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., 2024).
- **FastKAN:** 8 RBFs over $[-2, 2]$ (Li, 2024).
- **GramKAN:** degree–3 polynomial (Drokin, 2024).

We train with AdamW (Loshchilov & 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.The full list of hyperparameters is given in Appendix A.1. We report performance in terms of classification accuracy and model memory footprint.

4.1.2 Convolutional Setup

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

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	95.36	3,064.95	47.52	3,177.45	17.90
METAKAN	141.32	96.45	410.82	46.99	422.08	20.08
CLUSTERKAN	13.84	63.47	38.34	29.82	39.75	7.32
+ FINE-TUNE	13.84	93.34	38.34	43.73	39.75	12.54
METACLUSTERKAN	13.84	96.02	38.34	46.39	39.75	19.03
+ FINE-TUNE	13.84	96.45	38.34	46.59	39.75	19.43
FASTKAN	900.56	96.75	2,676.83	49.28	2,778.08	20.76
METAFASTKAN	108.09	96.29	316.35	47.41	327.60	19.89
CLUSTERFASTKAN	20.54	32.85	57.30	23.53	58.71	7.63
+ FINE-TUNE	20.54	87.37	57.30	38.26	58.71	12.85
METACLUSTERFASTKAN	20.54	96.28	57.30	47.46	58.71	19.65
+ FINE-TUNE	20.54	96.58	57.30	47.35	58.71	19.54
GRAMKAN	497.46	96.11	1,477.48	51.15	1,534.43	21.92
METAGRAMKAN	101.49	95.83	297.49	49.31	309.45	19.74
CLUSTERGRAMKAN	13.96	48.16	38.46	30.32	40.58	5.61
+ FINE-TUNE	13.96	85.22	38.46	45.36	40.58	14.51
METACLUSTERGRAMKAN	14.15	95.22	38.65	49.06	40.76	18.87
+ FINE-TUNE	14.15	95.82	38.65	49.31	40.76	19.46

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., 2024).
- **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. See Appendix A.1 for the complete hyperparameter sweep. Evaluation metrics are classification accuracy and model memory footprint.

4.2 RESULTS

The main results are reported in Tables 1 and 2. The general observation is that Meta variants have limited memory reduction but achieve classification accuracy comparable to their non-Meta counterparts, whereas clustering alone yields much higher memory compression, but at the cost of large accuracy drops. Our MetaCluster models retain the high accuracy of the Meta variants while achieving the memory compression of clustering.

4.2.1 FULLY-CONNECTED NETWORK

Table 1 demonstrates the high classification accuracy and compression factor of fully-connected MetaCluster models across all datasets and basis functions. For example, in the case of MetaClusterKAN, we can see that on Cifar-10, it achieves a compression factor of up to $10.7\times$ (from 410.82KB to 38.32KB) compared to the MetaKAN, and up to $79.9\times$ (from 3064.95 KB to 38.32KB) compared to the KAN. Furthermore, it achieves this compression factor while matching the classification accuracy of MetaKAN. Although the Meta variants are capable of achieving 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 $3.9\times$ (from 21.92% to 5.61%), which, when recovered with fine-tuning, remains a $1.5\times$ decrease in accuracy (from 21.92% to 14.51 %). Such results showcase the importance of leveraging a meta-learner to learn a set of activations that sit on a lower-dimensional subspace to aid in classification accuracy after clustering in fully-connected models.

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	99.55	13,654.27	73.15	13,744.62	21.46
METAKANCONV	3,048.40	99.25	3,052.90	72.45	3,143.25	43.59
CLUSTERKANCONV	429.95	94.81	430.51	33.98	520.87	11.32
+ FINE-TUNE	429.95	99.35	430.51	57.59	520.87	20.85
METACLUSTERKANCONV	434.77	99.21	435.34	67.50	525.70	37.06
+ FINE-TUNE	434.77	99.11	435.34	71.67	525.70	43.31
FASTKANCONV	13,632.09	99.27	13,652.37	76.06	13,742.72	45.40
METAFASTKANCONV	3,041.57	99.06	3,046.08	71.42	3,136.43	41.97
CLUSTERFASTKANCONV	428.03	72.87	428.61	14.24	518.97	6.47
+ FINE-TUNE	428.03	97.35	428.61	55.14	518.97	23.59
METACLUSTERFASTKANCONV	427.97	99.07	428.55	71.06	518.91	41.39
+ FINE-TUNE	427.97	99.06	428.55	71.38	518.91	41.60
GRAMKANCONV	7,586.47	99.25	7,597.72	80.92	7,688.07	47.10
METAGRAMKANCONV	3,047.93	99.50	3,052.43	81.88	3,142.79	53.26
CLUSTERGRAMKANCONV	418.93	84.83	419.49	12.44	509.85	2.39
+ FINE-TUNE	418.93	99.37	419.49	76.35	509.85	43.77
METACLUSTERGRAMKANCONV	418.81	99.48	419.38	79.36	509.73	51.56
+ FINE-TUNE	418.81	99.53	419.38	81.37	509.73	52.63

4.2.2 CONVOLUTIONAL NETWORK

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 $5.3\times$ decrease (from 76.06% to 14.24 %) in classification accuracy. Then, when recovered with fine-tuning, it still retains a $1.4\times$ decrease (from 76.06% to 55.14 %) in classification accuracy. Once again, as with the fully-connected architecture, these results demonstrate the importance of our MetaCluster framework for maintaining accuracy with a high compression factor.

4.3 ABLATION STUDY

4.3.1 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 coefficient count, our MetaCluster framework remains capable of learning a lower-dimensional manifold, which facilitates easier clustering of weights.

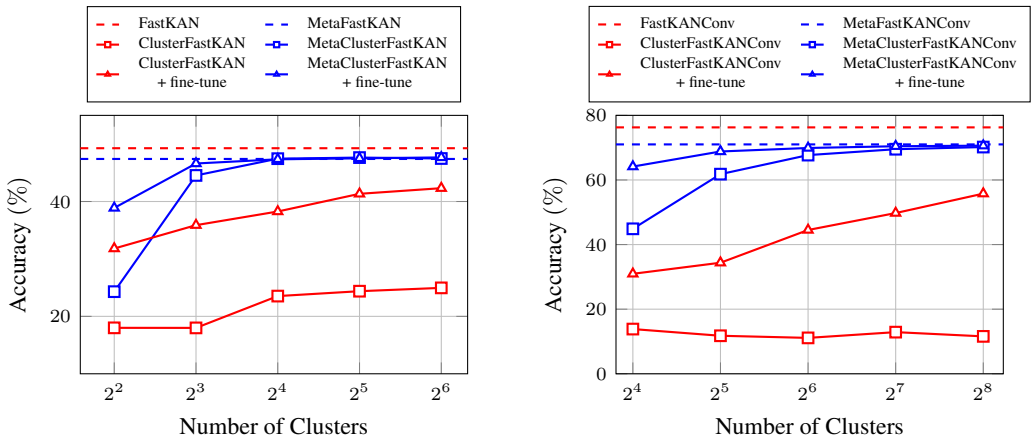
4.3.2 CLUSTER COUNT

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,

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

	MetaFastKAN		MetaClusterFastKAN			MetaFastKANConv		MetaClusterFastKANConv		
Coeff count	Memory	Acc%	Memory	Acc%	%Chg	Memory	Acc%	Memory	Acc%	%Chg
5	316.35	47.41	57.30	47.46	0.11	3,046.08	71.42	428.55	71.06	-0.50
8	316.76	47.62	57.68	46.96	-1.39	3,046.52	69.61	440.61	68.66	-1.36
10	317.03	46.91	57.93	47.04	0.28	3,046.81	66.78	448.64	66.58	-0.30
15	317.71	45.88	58.55	45.15	-1.59	3,047.54	64.24	468.72	63.99	-0.39
20	318.40	43.57	59.18	43.75	0.41	3,048.26	54.41	488.80	54.45	0.07

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 that our approach has the capability of achieving even greater levels of compression, albeit at the expense of a minor accuracy impact.



(a) Fully-connected KAN variants.

(b) Convolutional KAN variants.

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

4.3.3 Meta-learner Embedding Size

The embedding size of the metalearner 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 we see that increasing embedding dimension causes the plot to become less structured, we provide a quantitative analysis of the effect in Appendix A.2.2.

5 Related Works

5.1 Kolmogorov-Arnold Networks

The KAN has emerged as a popular alternative to the MLP (Liu et al., 2024). The most popular variant has used a weighted summation of a B-spline basis to represent the activations on each edge (Liu et al., 2024). However, since its rise to popularity, there have been numerous alternative variations of the KAN, such as using a host of basis functions, such as RBFs, Chebyshev polynomials, Legendre Polynomials, Gram polynomials, wavelets, and rational functions (Li, 2024; SS et al., 2024; Bozorgasl & Chen, 2024; Aghaei, 2024).

Although KAN has primarily achieved success in scientific applications and equation modeling tasks (Li et al., 2025; Wang et al., 2024; 2025; Coffman & Chen, 2025; Koenig et al., 2024), it has

recently shown promise in the realm of computer vision with the introduction of the Kolmogorov-Arnold Transformer (Yang & Wang, 2024; Raffel & Chen, 2025). However, despite its successes, the KAN has struggled with shortcomings such as increased training instability, computational cost, and parameter count (Yu et al., 2024; Chen et al., 2024). Our work directly targets the memory scaling obstacle of KANs, 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 on demand. The idea originates from early meta-learning works such as Ba et al. (2016), where a small network generates 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 that 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 & Maire, 2019), (Navon et al., 2020) .

MetaKANs (Zhao et al., 2025) bring this paradigm to Kolmogorov–Arnold Networks by observing that the dominant cost in a KAN is storing the coefficients of every 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. Han et al. (2015) popularized the practice with their practical pipeline consisting of magnitude pruning and K-means weight sharing. Subsequent theory linked quantization error to loss curvature and leveraged a Hessian-weighted K-means (Choi et al., 2016). Another line integrates clustering into training. For instance, differentiable K-means (DKM) optimizes centroids and assignments jointly with task loss, producing strong models in both vision and NLP (Cho et al., 2021). Building on prior work, our MetaCluster framework is the first to apply weight sharing to KANs successfully.

It is worth noting that, while weight-sharing is one avenue for model compression, bit-width quantization is an alternative avenue, which is largely orthogonal and complementary to the proposed approach. Applying quantization on top of MetaCluster (e.g., quantizing the code-book of centroids) has the potential to achieve a further reduction in the memory footprint.

6 Conclusion

We introduced MetaCluster, a compression framework that makes Kolmogorov–Arnold Networks practical at scale by introducing the novel combination of meta-learned manifold shaping with weight sharing. A lightweight meta-learner maps low-dimensional embeddings to per-edge basis coefficients, constraining KAN activations to lie on a compact manifold that is 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.

Across standard KANs and ConvKANs on MNIST, CIFAR-10, and CIFAR-100, MetaCluster achieves up to $80\times$ reduction in parameter storage without degrading accuracy. Visualizations and ablations demonstrate that manifold shaping is crucial for high-quality clustering in high dimensions, and that KANs benefit particularly from weight sharing compared to MLPs, as each centroid amortizes many coefficients.

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*, pp. 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.

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.

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.

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, pp. 4652–4660, 2025.

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

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*, 2024.

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

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.

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. *arXiv preprint arXiv:2406.11045*, 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

We report the complete set of hyperparameters for our experiments in Tables 4, 5, 6, and 7. For our fully-connected architecture experiments Table 5 reports the hyperparameters for our KAN variants and Table 5 reports the hyperparameters for our MetaKAN variants. Then for our convolutional architecture experiments Table 6 reports the hyperparameters for our KAN variants and Table 7 reports the hyperparameters for our MetaKAN variants.

Table 4: 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 5: 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

Table 6: 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 7: 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 Metalearner (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.2 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.2.1 Basis Coefficient Count Extended

Table 8 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 can see that since all the original classification accuracy is retained after clustering, there are minimal additional accuracy gains offered by fine-tuning the clustered model.

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

Table 8: Detailed results of grid sizes versus accuracy and memory for both fully-connected and convolutional MetaFastKAN networks on Cifar-10.

A.2.2 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 9. From Table 9, 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 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.

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

Table 9: Detailed results of embedding dimension versus accuracy for both fully-connected and convolutional MetaFastKAN networks on Cifar-10.

A.3 LLM Disclosure

We used large language models to assist in drafting and revising the manuscript.