

HybridKAN: Leveraging Multi-Sized Sub-MLPs for Enhanced Performance

Fabio D'Andreamatteo¹, Andrea D'Angelo^{1,*} and Giovanni Stilo¹

¹University of L'Aquila

Abstract

In recent years, Kolmogorov-Arnold Networks (KANs), based on the homonymous theorem, have been explored as an alternative to the classic Multi-Layer Perceptron (MLPs) architecture for deep neural networks. Despite showing some promising results for specific tasks, KANs are still limited by pathological cases where poor gradient behavior causes the network to fail. Moreover, the tuning and training requirements of KANs can be higher than those of MLPs. In this paper, we introduce a novel architecture called HybridKAN, which retains the overall structure of KANs but replaces the B-spline functions with sub-MLPs that approximate them. Our architecture is grounded in the Universal Approximation Theorem (UAT) and avoids abnormal gradient behavior by relying on sub-MLPs instead of B-spline functions. We test our new architecture under the same experimental setting as the original KAN paper and record significant improvements in both accuracy and training time.

Keywords

KAN, MLP, Machine Learning, Universal Approximation Theorem

1. Introduction

Multi-Layer Perceptrons (MLPs) have long served a foundational architecture in deep neural networks, largely due to the Universal Approximation Theorem (UAT). This theorem states that a sufficiently large feedforward neural network with at least one hidden layer can approximate any continuous function to an arbitrary degree of accuracy.

Recent advancements in machine learning have motivated researchers to explore a broad range of architectural alternatives to traditional MLPs. These efforts aim to improve computational efficiency [1, 2], enhance model interpretability [3], and overcome inherent limitations in MLPs. Among all, none have garnered as much attention as the *Kolmogorov-Arnold Networks (KANs)* [4].

Analogous to how MLPs are grounded in the UAT, the KAN architecture is inspired by the Kolmogorov-Arnold Representation Theorem (KART), which establishes that any multivariate continuous function can be expressed as a sum of univariate functions. Unlike conventional neural networks, KAN employs spline-based activation functions directly on the network's edges to approximate these univariate functions. This approach eliminates the necessity for fixed activation functions, linear weights, and traditional nodes performing only summation operations, thereby providing a more flexible and explainable framework for function approximation.

However, KANs are subject to many limitations [5]. Specifically, they struggle in pathological cases where the derived univariate functions are non-smooth, irregular, or even fractal in nature—conditions under which the model's ability to approximate target functions can degrade significantly [6]. Furthermore, KANs typically exhibit longer training times compared to MLPs, a trend we observe and validate through our empirical experiments.

In this paper, we propose a novel neural network architecture, called the **Hybrid Kolmogorov-Arnold Network (HybridKAN)**, where the spline-based activation functions on the edges are replaced with small, trainable MLP subnetworks, namely sub-MLPs, while keeping the traditional architectural design of the KAN network.

ITADATA2025: The 4th Italian Conference on Big Data and Data Science, September 9–11, 2025, Turin, Italy

*Corresponding author

✉ fabio.dandreamatteo@student.univaq.it (F. D'Andreamatteo); andrea.dangelo6@graduate.univaq.it (A. D'Angelo); giovanni.stilo@univaq.it (G. Stilo)

ORCID 0009-0003-6036-9467 (F. D'Andreamatteo); 0000-0002-0577-2494 (A. D'Angelo); 0000-0002-2092-0213 (G. Stilo)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

With this change, we shift the underlying approximation theoretical basis back to the Universal Approximation Theory (UAT), and improve the model’s expressive power and flexibility, allowing it to generalize better across more complex, nonlinear relationships in the data, avoiding pathological cases. HybridKAN merges the best of the two worlds: the more effective architecture of KANs with the reliability of MLPs. Figure 1 shows a comparison between KAN and HybridKAN architectures.

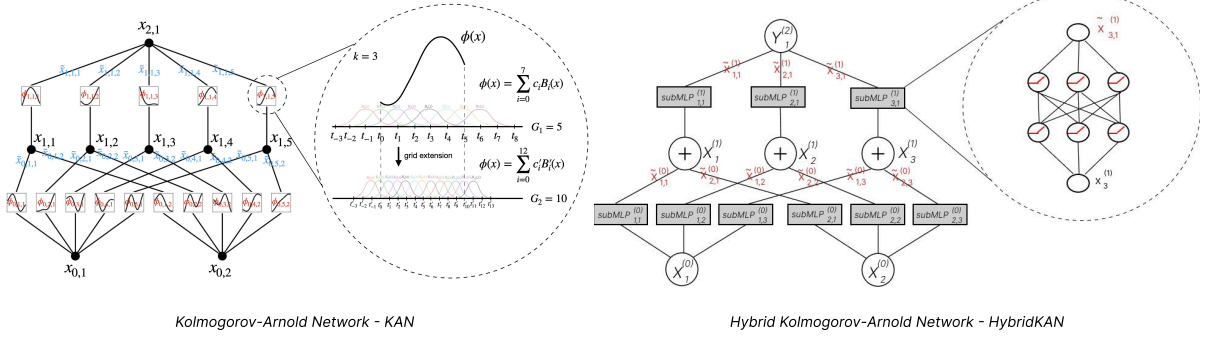


Figure 1: Comparison of KAN (left) and HybridKAN (right) architectures. HybridKAN retains the high-level architecture of a KAN network but substitutes B-splines, located on the connections between nodes of different layers, with Sub-MLPs. The Figure on the left is taken from [4].

Specifically, Figure 1 shows that a HybridKAN of the same shape of the corresponding KAN maintains the same architecture, but replaces B-spline functions with Sub-MLPs, small MLPs whose aim is to approximate the B-spline function they replace.

In order to evaluate our proposed architecture in comparison to KAN, we conduct several experiments on a dataset of physics equations, the same dataset used in the original KAN paper [4]. Specifically: *(i)*. We perform an in-depth grid search over the parameters to find the best configurations for both KAN and HybridKAN, and we report the best-performing architecture for each equation in the dataset. *(ii)*. We report the mean performance across all configurations to assess the stability of each model under suboptimal settings. *(iii)*. We analyze the loss surfaces to understand the optimization landscape of HybridKAN compared to KAN.

Our main contributions are the following.

- We propose **HybridKAN**, a novel architecture that replaces spline-based functions in KANs with small trainable MLP subnetworks (sub-MLPs).
- We empirically demonstrate that HybridKAN outperforms KANs in accuracy, training time, and robustness to pathological cases.
- We analyze the loss surfaces, showing more reliable optimization for HybridKAN compared to KAN.

The remainder of the paper is structured as follows. In Section 3, we lay out preliminaries on KANs and B-spline functions. In Section 4, we illustrate our proposed HybridKAN architecture. In Section 5 we list all the settings for the experiments discussed in the following Section 6. Lastly, in Section 7, we discuss conclusions.

2. Related Work

Kolmogorov–Arnold Networks (KANs) were first proposed in [4] as an interpretable alternative to traditional Multi-Layer Perceptrons (MLPs). Inspired by the Kolmogorov–Arnold representation theorem [7][8], KANs replace standard activation functions with learnable univariate functions and utilize fixed sparse linear connections for improved interpretability. Since their introduction, KANs have been applied across several domains, including time-series analysis [9], image classification [10, 11], and tabular data [4]. In addition to architectural innovations, KANs have begun to attract attention for

their robustness and explainability. For example, recent studies have evaluated KANs under adversarial conditions and shown that they may offer improved resistance to certain types of attacks compared to standard MLPs [11]. Furthermore, the structure of KANs lends itself well to post-hoc explainability analyses [4].

Despite these advances, KANs remain a relatively young architecture, and their performance in certain domains has not yet reached parity with more mature models such as convolutional or transformer-based networks, as reported in different benchmarks [12, 13]. Nonetheless, the architecture has undergone rapid iteration, with several extensions and enhancements being actively explored [14].

For instance, FastKAN [15] introduces optimizations that significantly reduce training time and improve scalability, thereby enhancing the feasibility of KANs for larger datasets. However, because FastKAN approximates the original KAN formulation rather than preserving its exact architecture, it does not provide a direct baseline for our comparative evaluation and is therefore excluded from the scope of this study. Similarly, [16] approximates the B-splines with wavelet functions. Convolutional KANs (CKANs) [17] integrate convolutional priors into the KAN framework to better handle spatial information. Autoencoder variants of KANs have also been proposed [18], aiming to leverage KANs’ interpretability in unsupervised learning and representation learning settings.

However, direct comparisons between KANs and MLPs have consistently highlighted the instability and inconsistency of KANs, largely due to their problematic gradient behavior. In particular, B-spline-based KANs often suffer from poor convergence and sensitivity to initialization, leading to divergent loss trajectories or numerical instability during training. As shown in [5], while KANs can reduce parameter counts, they struggle with higher training times and degraded performance—especially after symbolic conversion—on complex classification tasks, and often require significantly more computational resources. Similarly, [6] shows that certain KAN variants exhibited non-smooth loss landscapes and divergent gradients (e.g., NaNs), especially with high-order polynomials or deeper architectures, reinforcing the critical role of gradient behavior in their training stability.

In this paper, we propose a novel MLP-based architecture that replaces the spline functions in the original KAN design with small MLPs (sub-MLPs). This approach retains the structural benefits of the KAN framework while mitigating issues related to unstable gradients during training and significantly improving training time.

3. Preliminaries

KAN Background. The *Kolmogorov-Arnold Representation Theorem*, also known as *Superposition Theorem*, was introduced in 1957 by Andrey Kolmogorov and later refined by his student Vladimir Arnold [8, 7]. The theorem states: “If f is a multivariate continuous function on a bounded domain, then f can be written as a finite composition of continuous univariate functions and the binary operation of addition”.

Considering a continuous function $f : [0, 1]^n \rightarrow \mathbb{R}$, according to the theorem, there exist continuous univariate functions $\phi_{q,p} : [0, 1] \rightarrow \mathbb{R}$ and $\Phi_q : \mathbb{R} \rightarrow \mathbb{R}$ such that:

$$f(x) = f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \phi_{q,p}(x_p) \right) \quad (1)$$

In other words, the theorem states that any continuous multivariate function can be expressed as a finite sum of compositions of continuous univariate functions. This powerful result lays the theoretical foundation for architectures such as Kolmogorov–Arnold Networks (KANs), which model multivariate mappings using learnable univariate transformations.

The fundamental difference between Multi-Layer Perceptrons (MLPs) and Kolmogorov-Arnold Networks (KANs) lies in how the networks handle non-linear transformations. In MLPs, nodes have fixed activation functions to introduce nonlinearity, while edges carry the learnable linear weights. As opposite, KANs place learnable activation functions on edges, parametrized as one-dimensional splines,

and the binary operation of sum on nodes, without additional processing other than simply summing incoming signals. The purpose of introducing nonlinearity is then shifted from nodes to edges, where in KANs there are no linear weight matrices since each weight is replaced by a learnable function $\phi_{q,p}$ which processes a single input variable.

A *KAN Layer* with n_{in} -dimensional inputs and n_{out} -dimensional outputs can be expressed as a matrix of learnable 1D functions: $\Phi = [\phi_{q,p}]$, $p = 1, 2, \dots, n_{in}$; $q = 1, 2, \dots, n_{out}$. Generally, the shape of a KAN network is represented by an integer array $[n_0, n_1, \dots, n_L]$. Between two consecutive layers, say l and $(l+1)$, there are $(n_l \cdot n_{l+1})$ activation functions, where the activation function that connects (l, i) and $(l+1, j)$ is denoted as: $\phi_{l,j,i}$, $l = 0, \dots, L-1$, $i = 1, \dots, n_l$, $j = 1, \dots, n_{l+1}$. Each activation function $\phi_{l,j,i}$ processes a pre-activation value x_l and produces a post-activation value $\tilde{x}_{l+1,j} = \phi_{l,j,i}(x_{l,i})$, which serves as part of pre-activation value for the next layer $\phi_{l+1,j,i}$.

B-splines. Since all learned functions are univariate, it is possible to parameterize them as B-spline curves, with learnable coefficients of local B-Spline basis functions. *B-Spline Curves* are defined by a set of *Control Points*, which are going to represent the learned parameters. Unlike other composite curves, control points in B-Splines do not affect the entire curve but only provide local control, meaning that modifying a single control point influences only the portion of the curve that it is associated with.

In the KAN architecture, the B-Spline functions are constructed using basis functions $b(x)$, which are similar to residual connections, such that the activation function $\phi(x)$ is the sum of the basis functions $b(x)$ and the spline function:

$$\phi(x) = w_b b(x) + w_s spline(x) \quad (2)$$

where:

$$b(x) = silu(x) = \frac{x}{(1 + e^{-x})} \quad spline(x) = \sum_i c_i B_i(x)$$

The spline function is expressed as a sum of weighted basis functions B_i , where c_i are the scalar coefficients corresponding to each B-Spline basis function B_i that determine its contribution to the overall spline. These coefficients are learned and adjusted during training to fit the B-Spline to the target function that needs to be approximated. In addition to the coefficients c_i , the parameters w_b and w_s are learned to better control the overall contribution of the basis and spline components to the activation function.

4. HybridKAN - Approximating Splines with sub-MLPs

To address the limitations of KANs mentioned in Section 2, we propose a new architecture, called the *Hybrid Kolmogorov-Arnold Network - HybridKAN*, which is a modified version of the Kolmogorov-Arnold Network that replaces the splines-based learnable activation functions on edges with small trainable MLP subnetworks. With this new approach, our aim is to maintain the advantages of the structural composition of KANs, which mainly consists of isolated transformations of the input variables and pairwise connections to the aggregating nodes, while trying to improve its expressive power, adaptive learning, and scalability through the use of sub-MLPs rather than spline-based functions.

HybridKAN diverges from the Kolmogorov-Arnold Representation Theorem (KART), which forces the decomposition of the underlying multivariate function into a sum of univariate functions, to return to the classic Universal Approximation Theorem. The aim of shifting the underlying approximation theory is to entirely avoid the possibility that some multivariate functions may be decomposed into non-smooth or irregular functions, effectively enhancing the model's performance and flexibility. While KANs explicitly follow the theorem by ensuring that each spline-based edge approximates a single univariate function, HybridKAN replaces this mechanism with small subnetworks, increasing the expressive power of the network and allowing it to detect more complex patterns and relationships within the data.

As shown in Figure 2, the HybridKAN architecture consists of multiple stacked HybridKAN Layers. Each layer contains a set of independent sub-MLPs that transform single input features separately

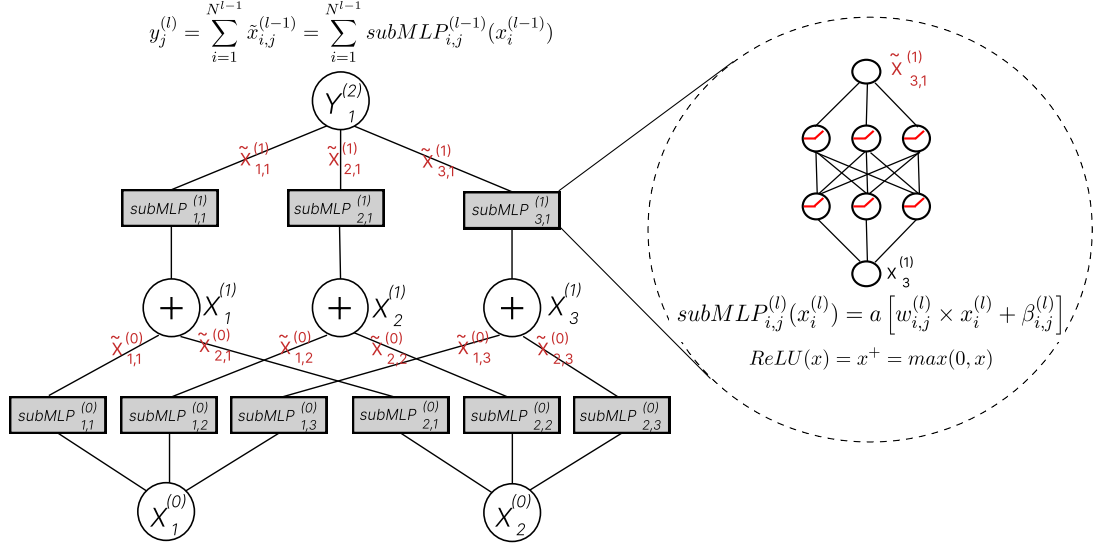


Figure 2: An example HybridKAN architecture with shape [2,3,1]. The architecture is analogous to a KAN of the same shape; however, every B-spline is replaced by a sub-MLP, in this case of shape [1,3,3,1]. The aim of these sub-MLPs is to approximate the original B-splines.

before aggregating them on the nodes. The number of subMLPs per layer is determined by the number of input nodes and output nodes, with each subnetwork assigned to process one specific input node and contribute to one specific output node. Consider a layer l that has $N(l)$ input nodes and $N(l+1)$ output nodes. In this layer, the input vector $x^l \in R^{N(l)}$ is processed to generate an aggregated output vector $y^{l+1} \in R^{N(l+1)}$.

To achieve this, the layer is composed of $N(l) \times N(l+1)$ sub-MLPs. These subnetworks are denoted as $subMLPs_{i,j}^{(l)}$, where $i \in \{1, 2, \dots, N^{l-1}\}$ represents the index of the input node and $j \in [1, 2, \dots, N^l]$ corresponds to the index of the output node. Each sub-MLP applies a transformation to its assigned input variable, and the outputs of all sub-MLPs corresponding to a given node are summed to produce the aggregated output of that node.

As an example, in Figure 2, layer 0 has two input nodes ($X_{0,1}$ and $X_{0,2}$) and three output nodes ($X_{1,1}, X_{1,2}, X_{1,3}$). As a result, layer 0 is composed of 6 sub-MLPs.

The most important difference with MLPs occurs mainly in the first layer, where each input feature undergoes a separate transformation before aggregation, enabling modular and independent feature representations. The subsequent layers then refine these representations through additional transformations and summations. Formally, let $x = (x_1, x_2, \dots, x_N)$ be the input vector. Each feature x_i is passed to a set of sub-MLPs, one for each node in the next layer. The transformed output of the sub-MLP associated with the i^{th} input feature and to the j^{th} node in the layer is the following:

$$\tilde{x}_{i,j}^{(l)} = subMLP_{i,j}^{(l)}(x_i^{(l)}) = a[w_{i,j}^{(l)} \times x_i^{(l)} + \beta_{i,j}^{(l)}] \quad (3)$$

After independent transformations of the input variables, the layer nodes aggregate these transformed values by summing them pairwise. The output of the j^{th} node in the layer is computed as:

$$y_j^{(l)} = \sum_{i=1}^{N^{l-1}} \tilde{x}_{i,j}^{(l-1)} = \sum_{i=1}^{N^{l-1}} subMLP_{i,j}^{(l-1)}(x_i^{(l-1)}) \quad (4)$$

Each node in the subsequent layers receives the transformed signals from the previous layer outputs, which were already aggregated from multiple input features. This implies that while the first layer processes raw input features independently, all subsequent layers work on already transformed inputs to refine the representation learned in the previous layers, making the network able to learn increasingly

complex function approximations. For the final output node $y_j^{(L)}$:

$$y_j^{(L)} = \sum_{i=1}^{N^{L-1}} \text{subMLP}_{i,j}^{(L-1)}(x_i^{(L-1)}) \quad (5)$$

Consider a 2-layer HybridKAN network with shape $[2,3,1]$, like the one in Figure 2. Then, the first layer computes:

$$\begin{aligned} X_1^{(1)} &= \text{subMLP}_{1,1}^{(0)}(x_1^{(0)}) + \text{subMLP}_{2,1}^{(0)}(x_2^{(0)}) \\ &= a \left[\phi_{1,1}^{(0)}(x_1^{(0)}) + \beta_{1,1}^{(0)} \right] + a \left[\phi_{2,1}^{(0)}(x_2^{(0)}) + \beta_{2,1}^{(0)} \right] \end{aligned} \quad (6)$$

$$\begin{aligned} X_2^{(1)} &= \text{subMLP}_{1,2}^{(0)}(x_1^{(0)}) + \text{subMLP}_{2,2}^{(0)}(x_2^{(0)}) \\ &= a \left[\phi_{1,2}^{(0)}(x_1^{(0)}) + \beta_{1,2}^{(0)} \right] + a \left[\phi_{2,2}^{(0)}(x_2^{(0)}) + \beta_{2,2}^{(0)} \right] \end{aligned} \quad (7)$$

$$\begin{aligned} X_3^{(1)} &= \text{subMLP}_{1,3}^{(0)}(x_1^{(0)}) + \text{subMLP}_{2,3}^{(0)}(x_2^{(0)}) \\ &= a \left[\phi_{1,3}^{(0)}(x_1^{(0)}) + \beta_{1,3}^{(0)} \right] + a \left[\phi_{2,3}^{(0)}(x_2^{(0)}) + \beta_{2,3}^{(0)} \right] \end{aligned} \quad (8)$$

The final output is then computed on the previous representations as:

$$\begin{aligned} Y_1^{(2)} &= \text{subMLP}_{1,1}^{(1)}(X_1^{(1)}) + \text{subMLP}_{2,1}^{(1)}(X_2^{(1)}) + \text{subMLP}_{3,1}^{(1)}(X_3^{(1)}) \\ &= a \left[\phi_{1,1}^{(1)}(X_1^{(1)}) + \beta_{1,1}^{(1)} \right] + a \left[\phi_{2,1}^{(1)}(X_2^{(1)}) + \beta_{2,1}^{(1)} \right] + a \left[\phi_{3,1}^{(1)}(X_3^{(1)}) + \beta_{3,1}^{(1)} \right] \end{aligned} \quad (9)$$

Implementing this type of network can provide many significant advantages. One of the most important would be the increased expressivity and flexibility compared to standard KANs. By replacing B-Splines on edges with trainable MLP subnetworks it is possible to learn more complex relationships within the data, avoiding all pathological cases of non-smooth or irregular univariate functions inherited by KART. HybridKANs are no longer constrained by the structural limitations of summing univariate functions, allowing the model to dynamically approximate more complex relationships and adapt to a broader range of function approximations. In particular, the use of sub-MLPs in the first layer can provide an adaptive feature transformation, meaning that each input feature can be processed independently in a more flexible way before being aggregated. Only in the subsequent layers aggregated features can be refined to better approximate the function.

Additionally, HybridKANs introduce a certain degree of regularization through the summation mechanism at nodes. Since each node aggregates transformed signals from multiple sub-MLPs, it can reduce redundant information and the risk of over-reliance on individual features, leading to an improved generalization, and making the model more robust to noise and more suitable for real-world applications where the input often has dependencies between features that are difficult to catch explicitly.

Computational Complexity. In terms of computational complexity, both models have an asymptotic complexity of $O(N^2)$, where N denotes the width of the layers. The models share a similar complexity: HybridKAN’s complexity is expressed as $O(N^2 LH)$, where L denotes the number of layers and H represents the number of internal parameters of sub-MLP, while KAN’s complexity is expressed as $O(N^2 LG)$, where G denotes the number of control points in splines. Typically, G is smaller than H . However, the increased computational cost required for updating the more complex spline-based functions can deny its parameter advantage. In contrast, even if HybridKAN involves a larger set of parameters to update, these parameters are simple linear weights. As a result, HybridKAN’s overall computational cost often remains comparable, or even lower, to the computational cost of KAN.

5. Experimental Settings

To assess the performance of the HybridKAN model, we tested both HybridKAN and KAN and compared their performance on a regression task to determine whether the modifications introduced in the

HybridKAN model can actually lead to enhanced performance compared to the standard KAN model in terms of accuracy and/or training time. Both architectures were trained and tested using the same data samples and a fixed number of epochs to ensure reproducibility of the results.

Dataset. The task is designed to replicate the experiment performed by the authors of KAN [4] who evaluated their model on a set of 27 artificially generated datasets from different physics equations, selected from a larger collection [19]. They were chosen to represent a variety of functional forms and complexities and used to synthetically generate datasets divided into 10,000 training samples and 10,000 testing samples. In our study, for a fair comparison, we select the **same subset of equations** that was selected by the original authors of KAN. In the following, we will reference the equations by their ID in the original Feynman dataset, which implies some IDs could be missing. See Appendix A for more details.

Goal. Since the datasets are inherently free of noise, where every data point is taken from a distribution which follows a well-defined underlying multivariate formula, our primary goal was to check the models’ ability to approximate these underlying functions as precisely as possible in a fair amount of training time, where any deviation between the model prediction and the true function is only due to the model’s ability to precisely approximate the function and minimize loss rather than handling the variability and presence of noise within the data.

Hardware. All runs were computed on the Cluster Caliban HPC of the University of l’Aquila, in isolation on a Rocky Linux 8 server with 32-core Intel(R) Xeon(R) CPU @ 2.30GHz, up to 64GB RAM, NVIDIA A100 GPU.

Parameters. The experiment involved an extensive grid search on hyperparameters, exploring different configurations of macro-structures, such as variations in the number of layers and nodes in each layer, as well as different combinations of lower-level parameters such as neurons, grid points, and spline orders (k values). The research is focused on identifying the optimal configurations for the models to compare them in a fair way and to understand the extent to which these settings affect performance. For the macro-structures, to determine to what extent various depths and widths affect, five different configurations were tested: $[(N_{in}, 3, 1), (N_{in}, 5, 1), (N_{in}, 3, 3, 1), (N_{in}, 5, 3, 1), (N_{in}, 5, 5, 1)]$. Additionally, various lower-level hyperparameters were explored. For the HybridKAN model, each sub-MLP is composed of two layers with a set of predefined combinations of neurons: $[(4, 6), (4, 8), (4, 10), (6, 6), (6, 8), (6, 10), (8, 6), (8, 8), (8, 10), (10, 6), (10, 8), (10, 10)]$. Given the unprecedented performance of the HybridKAN model, a broader range of hyperparameter configurations was chosen in order to understand its variability and potential. All combinations were made using four only values: 4, 6, 8, 10, allowing to include mirror pairs to check whether an expansion or contraction in the number of neurons between the layers of the subMLP can have a different impact on the model’s behavior. For the KAN model instead, which was designed specifically for symbolic functions approximation, it was performed a grid search on the number of control points for the B-Spline and on the K value (spline order). The grid points were set to 3, 5, 10, and 15, while the K value was set to 3 and 5. For both models, the value of learning rate was set to 1 due to the usage of the LBFGS optimizer, which fine-tunes the learning rate over epochs to facilitate convergence in a full-batch approach.

6. Experimental Analysis

In this section, we present a comprehensive analysis of the experimental results obtained by comparing KAN and HybridKAN architectures. The main objective of this analysis is to evaluate the performance, stability, and optimization characteristics of the proposed HybridKAN model relative to KAN across a diverse set of equations from the Feynman dataset.

We structure our analysis into three main parts. First, we identify and compare the **best-performing configurations** for each model on each equation to assess their peak predictive capabilities (Section 6.1). Second, we evaluate **model stability and training efficiency** by analyzing aggregated statistics, such as average accuracy and training times across all tested configurations, to understand each model’s robustness and sensitivity to hyperparameter choices (Section 6.2). Finally, we investigate the **loss**

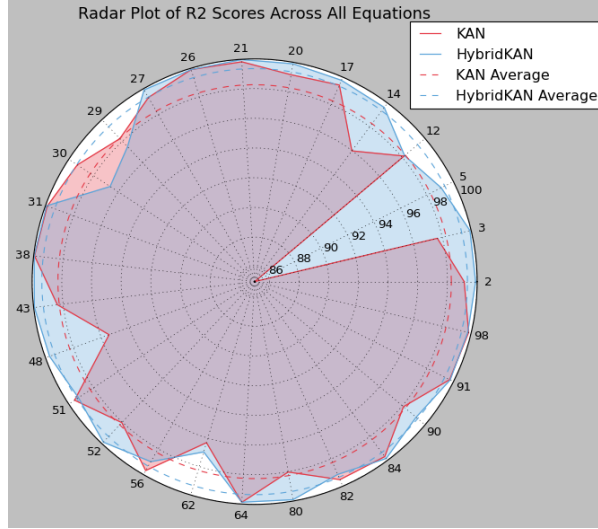


Figure 3: Best R^2 Score Achieved (Range 85-100%) of KAN (red) and HybridKAN (blue) across 27 equations from the Feynman dataset. On the radar there are the IDs of the selected equations.

surfaces of both models, examining how different macro- and micro-level parameter configurations influence their optimization landscapes and performance, thereby shedding light on their generalization behavior (Section 6.3).

Through this structured analysis, we aim to provide a comprehensive evaluation of HybridKAN’s advantages over KAN in terms of performance, reliability, and architectural flexibility.

6.1. Best performing architecture per equation

We now report the performance of the best performing networks on the selected equations.

As detailed in Section 5, we performed a grid search over several hyperparameters for both KAN and HybridKAN. The optimal configuration for the models was selected based on the highest accuracy achieved, while keeping training time as a tie-breaker when differences in terms of accuracy were minimal.

The individual best performance and the corresponding parameters are reported in Appendix B.

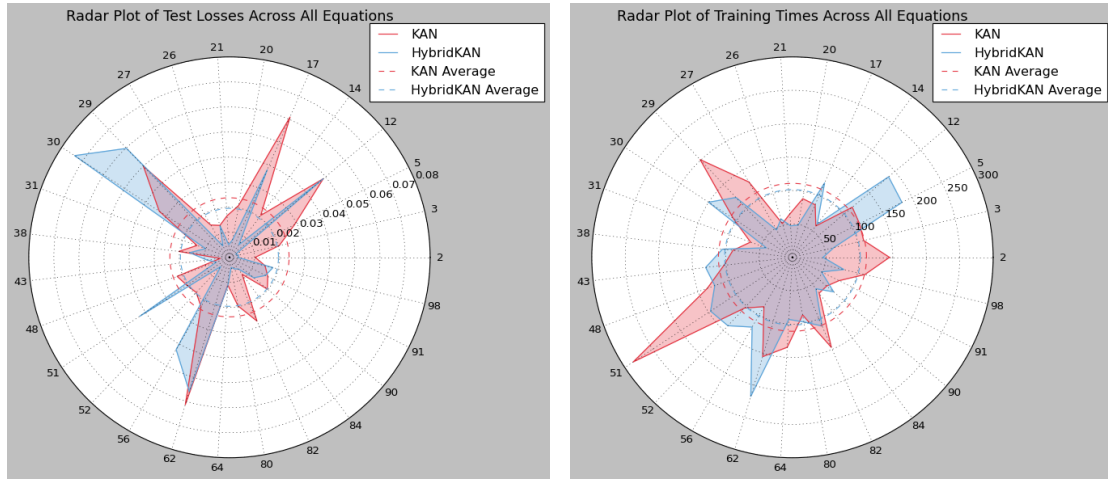
Figure 3 shows the R^2 score (where higher is better) achieved by the best models of KAN and HybridKAN for each equation. The HybridKAN architecture outperforms KANs on nearly all equations, as validated by the Average across all equations shown in dashed circles. Moreover, the poor gradient behavior inherent in KANs is clearly demonstrated by their inability to achieve acceptable performance on Equation 5. In contrast, HybridKAN effectively mitigates this pathological behavior, producing consistent and reliable results on the same equation, thus highlighting its improved stability and robustness.

Figure 4 shows the test loss (Figure 4a, where lower is better) and the training times (Figure 4b, where lower is better) of the best performing KAN and HybridKAN architectures over a grid search.

As Figure 4a shows, the HybridKAN architecture generally achieves lower test losses (aside from a couple of hiccups in Equations 29,30,56) with respect to the KAN architecture, which was specifically tested on these equations by the original authors. The mean test loss, shown as dashed circles, is also lower for the HybridKan architecture, validating the superior performance of HybridKAN.

Figure 4b shows that the training time for HybridKANs is generally lower than the ones for KANs, as indicated by the dashed circle showing the average training times. HybridKAN shows more consistent and regular training times, whereas KANs sometimes abnormally peak (as in Equations 2, 29, 51, 82).

These results show that the HybridKAN architecture is better both in terms of performance (R^2 score, test loss) and training times, taking generally less time to train to achieve best performance.



(a) Test losses across equations (smaller covered area -> better performance) (b) Training times across equations (smaller covered area -> better performance)

Figure 4: Radar plots showing Test losses (left) and Training times (right) of KAN (red) and HybridKAN (blue) across 27 equations from the Feynman dataset. On the radar there are the IDs of the selected equations.

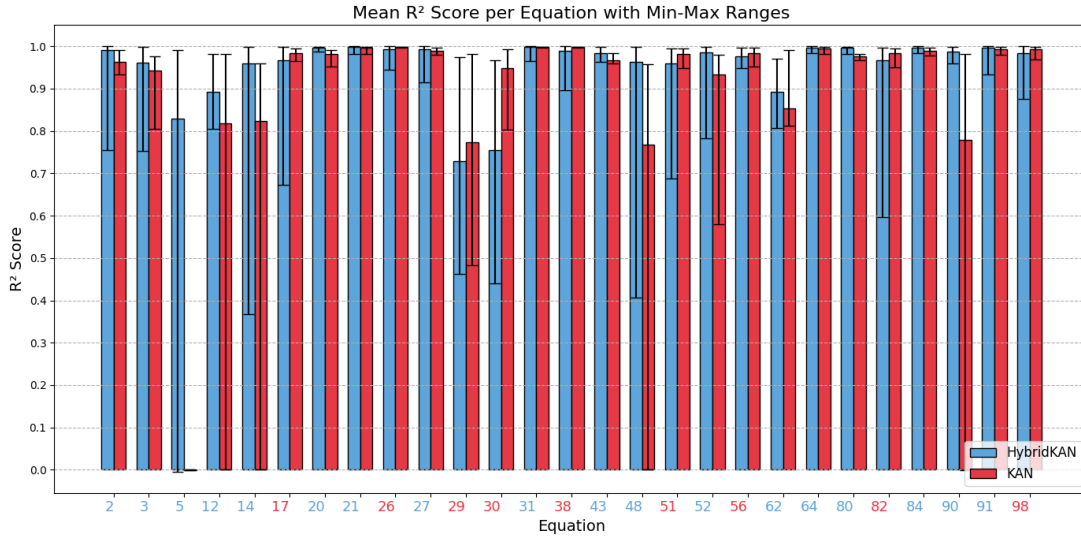


Figure 5: Average R^2 scores for all equations (higher is the better), computed across all tested hyperparameter configurations, with error bars representing the standard deviation. Colored equation IDs indicate the best-performing architecture for each equation (red for KAN, blue for HybridKAN).

6.2. Evaluating Model Stability and Training Efficiency

In this phase of the analysis, we focus on evaluating aggregated statistics—including the average, standard deviation, minimum, and maximum—of key performance indicators across all configurations tested in the grid search. By analyzing these metrics, we aim to understand how different architectural and hyperparameter choices influence the performance of both models. This allows us to evaluate each model's sensitivity to such variations and to identify trade-offs between metrics across configurations.

Figure 5 shows the R^2 scores across all models tested on the grid search. The color of the equation IDs on the X-axis show the best performing architecture. HybridKAN surpasses the KANs on the vast majority of equations. Again, KANs collapse on Equation 5, corroborating pathological behavior. Moreover, KAN exhibits significantly larger error bars for most of the equations, indicating that it is the model which is subject the most to variability in performance and suggesting it is more sensitive to different initializations or specific equation structures. HybridKAN, on the other hand, has smaller

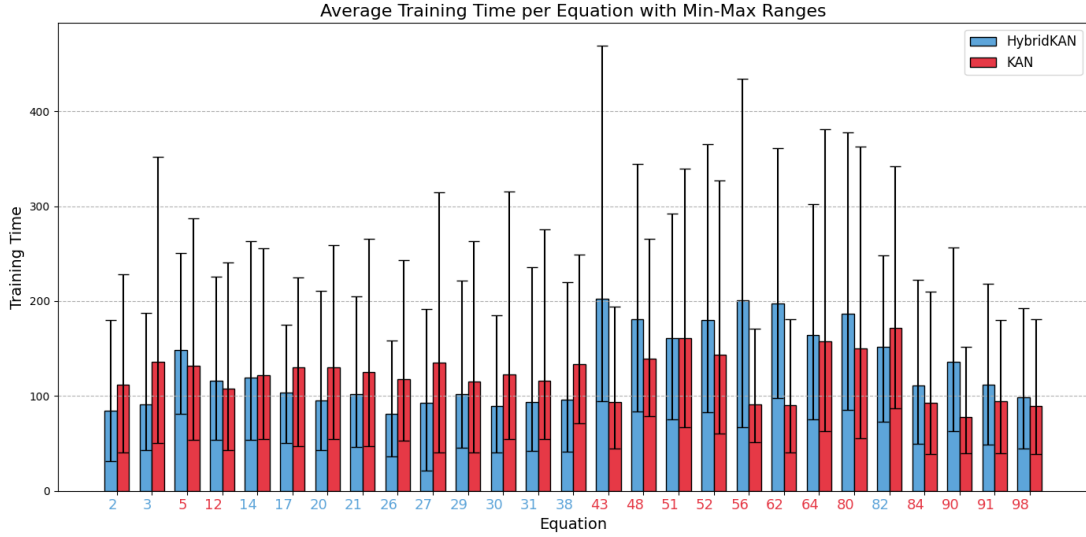


Figure 6: Average training times for all equations (lower is the better), computed across all tested hyperparameter configurations, with error bars representing the standard deviation. Colored equation IDs indicate the best-performing architecture for each equation (red for KAN, blue for HybridKAN).

error bars, meaning that its performance is stable across different configurations, making the model more flexible regarding particular conditions.

Figure 6 shows training times in a similar fashion. Even if the highest peaks are reached by the HybridKAN model, a closer look at error bars reveals that KAN exhibited greater variability in training time across all equations, with larger error bars for most of the cases. Despite occasional high peaks, HybridKAN’s overall training times are actually more consistent, highlighting that the model is more predictable and reliable also when it comes to running times compared to the more erratic performance of KAN.

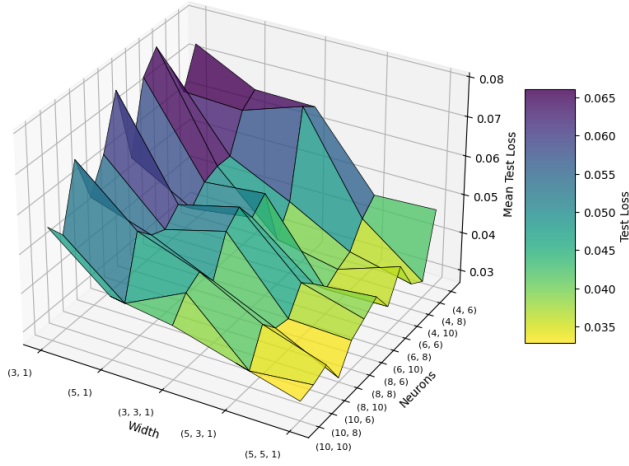
Overall, these results demonstrate that HybridKAN not only achieves superior performance compared to KAN across the majority of equations but also exhibits greater robustness and stability. While KAN models display substantial variability in both accuracy and training times—indicating sensitivity to initialization and equation structure—HybridKAN maintains consistent performance and training efficiency across configurations.

6.3. Understanding loss surfaces for KAN and HybridKAN

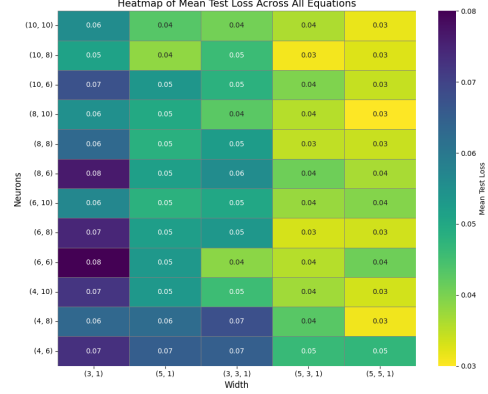
This final part of the analysis investigates the impact of different initialization choices on model performance, focusing specifically on test loss and accuracy. In particular, the study distinguishes between the effects of macro-level parameters, which refer to architectural aspects such as the number of layers and nodes per layer, and micro-level parameters, which include lower-level configuration settings—such as the number of neurons in each sub-MLP for HybridKAN, or the number of control points (grid size) for splines in KAN. By examining the influence of both parameter levels, we seek to understand how their combinations affect each model’s ability to approximate the underlying functions, as measured by test loss.

Figure 7 show the test loss surface and heatmap for HybridKAN across different configurations. Specifically, width refers to the shape of the larger model, while Neurons is the number of neurons in each sub-MLP. HybridKAN displays a more structured and layered test loss surface, with marked variations in loss depending on the architecture and hyperparameters. In particular, lower test losses (yellow regions) are observed for deeper architecture with more neurons per subMLP. This suggests that increasing the number of neurons in HybridKAN notably improves the performance of the model. However, there are diminishing returns in increasing the size of the subMLPs since improvements in loss caused by added neurons become minimal, even if they can still be somehow relevant. Higher

Surface Plot of Mean Test Loss Across All Equations



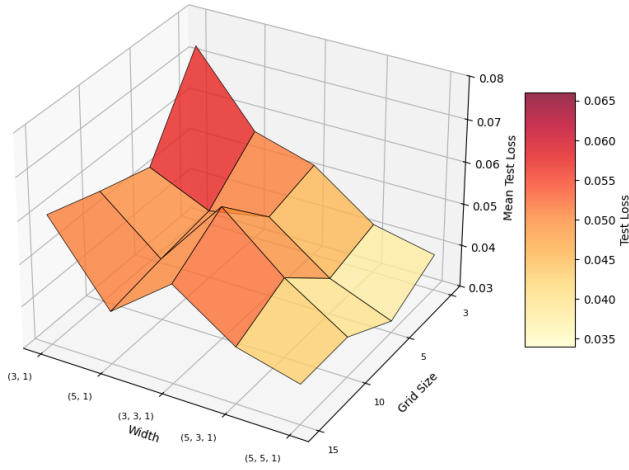
(a) Test Loss Surface for HybridKAN



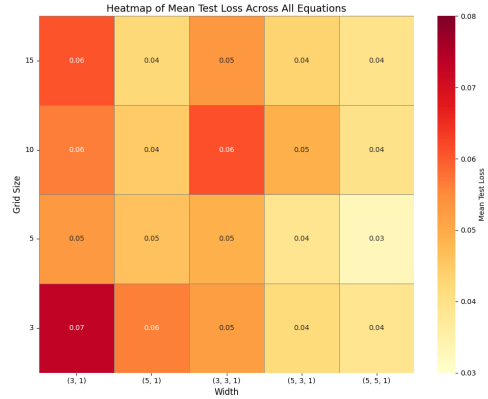
(b) Test Loss Heatmap for HybridKAN

Figure 7: Surface and Heatmap of Average Test Loss Among All Equations for HybridKAN on different hyperparameters, both for the Network shape (Width) and Sub-MLP shape (Neurons).

surface Plot of Mean Test Loss Across All Equations



(a) Test Loss Surface for KAN



(b) Test Loss Heatmap for KAN

Figure 8: Surface and Heatmap of Average Test Loss Among All Equations for KAN on different hyperparameters, both for the Network shape (Width) and the number of control points in the KAN network (Grid size).

peaks, indicated by purple regions, are located in correspondence of shallower networks with fewer neurons, meaning that, in general, under-parametrized configurations struggle to generalize across multiple equations. This can ultimately suggest that the HybridKAN network may benefit and scale well with added complexity to function smoothly, allowing for more control over variability, which is also suggested by the lack of peaks during the descent.

In KAN, as shown in Figure 8, the surface of the test loss is more irregular, with sharp spikes from high to low losses at certain configurations, suggesting that the model may be more sensitive to grid size and architectural widths. Unlike HybridKAN, where increasing both width and depth of the model always reduces test loss, KAN exhibit a different pattern for various grid sizes, indicating that the choices on this hyperparameter depends on the task at hand where too small or too large grid sizes may lead to higher test losses and, therefore, reduced performance.

In fact, there are regions where adding or removing grid points worsen performance, likely due to

overfitting or insufficient capacity in representing the function. From these results it is possible to say that, unlike HybridKAN, where increases in neurons consistently improve performance, KAN requires specific combinations of grid sizes for each architectural choice, as poor choices can worsen performance. For better visualization, to the surface plots of both models are added the corresponding heatmaps of the average test loss values computed from all equations across all combination of macro and micro-level configurations.

7. Conclusion and Future Work

In this work, we proposed a novel MLP-based architecture called HybridKAN, which retains the overall structure of Kolmogorov-Arnold Networks (KANs) while replacing their B-spline functions with trainable sub-MLPs. This architectural shift grounds the model back in the Universal Approximation Theorem, addressing the pathological cases observed in KANs and resulting in improved performance with more stable and generally shorter training times. We conducted an exhaustive grid search over network and subMLP configurations, providing detailed results and ablation studies to evaluate the impact of different architectural choices. We test both KAN and HybridKAN on the same 27 equations that the original KAN paper sampled from the Feynman dataset, showing HybridKAN’s superiority. Finally, our analysis of the loss surfaces corroborates that HybridKAN not only achieves higher r^2 scores on regression tasks but also exhibits more consistent optimization behavior across diverse tasks.

This study represents a first investigation into the potential of HybridKAN, and future work will extend this analysis to include classification tasks, additional benchmark datasets, and a significantly broader range of experiments to further validate and refine the model’s capabilities across various machine learning domains. While our current evaluation focuses on KAN architectures, future work will include comparisons with standard MLPs to contextualise HybridKAN’s general performance advantages. It must be noted that, while HybridKAN outperforms KAN in terms of performance, it remains less explainable. A deeper study on the explainability of HybridKAN will be subject of future work.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT-4o in order to: Grammar and spelling check. After using these tools, the author(s) reviewed and edited the content as needed and take full responsibility for the publication’s content.

Acknowledgements

The numerical simulations have been realized on the HPC cluster of the Department of Information Engineering, Computer Science and Mathematics (DISIM) at the University of L’Aquila. The work is partially funded by the European Union - NextGenerationEU under the Italian Ministry of University and Research (MUR) National Innovation Ecosystem grant ECS00000041 - VITALITY - CUP E13C22001060006, by National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) - Project: “SoBigData.it - Strengthening the Italian RI for Social Mining and Big Data Analytics” - Prot. IR0000013 - Avviso n. 3264 del 28/12/2021, and by the “ICSC – Centro Nazionale di Ricerca in High Performance Computing, Big Data and Quantum Computing.”

References

- [1] Q. Zheng, C. Zhang, J. Sun, Sa-mlp: A low-power multiplication-free deep network for 3d point cloud classification in resource-constrained environments, 2025. URL: <https://arxiv.org/abs/2409.01998>. arXiv:2409.01998.

- [2] H. Mu, B. Ul Tayyab, N. Chua, SpiralMLP: A Lightweight Vision MLP Architecture , in: 2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), IEEE Computer Society, Los Alamitos, CA, USA, 2025, pp. 8627–8637. URL: <https://doi.ieeecomputersociety.org/10.1109/WACV61041.2025.00836>. doi:10.1109/WACV61041.2025.00836.
- [3] D. Alvarez-Melis, T. S. Jaakkola, Towards robust interpretability with self-explaining neural networks, 2018. URL: <https://arxiv.org/abs/1806.07538>. arXiv:1806.07538.
- [4] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, M. Tegmark, Kan: Kolmogorov-arnold networks, 2025. URL: <https://arxiv.org/abs/2404.19756>. arXiv:2404.19756.
- [5] V. D. Tran, T. X. H. Le, T. D. Tran, H. L. Pham, V. T. D. Le, T. H. Vu, V. T. Nguyen, Y. Nakashima, Exploring the limitations of kolmogorov-arnold networks in classification: Insights to software training and hardware implementation, 2024. URL: <https://arxiv.org/abs/2407.17790>. arXiv:2407.17790.
- [6] K. Shukla, J. D. Toscano, Z. Wang, Z. Zou, G. E. Karniadakis, A comprehensive and fair comparison between mlp and kan representations for differential equations and operator networks, Computer Methods in Applied Mechanics and Engineering 431 (2024) 117290. URL: <https://www.sciencedirect.com/science/article/pii/S0045782524005462>. doi:<https://doi.org/10.1016/j.cma.2024.117290>.
- [7] V. I. Arnold, On the representation of continuous functions of several variables by superpositions of continuous functions of one variable, Uspekhi Mat. Nauk 18 (1963). An influential contribution to the development of the theorem.
- [8] A. N. Kolmogorov, On the representation of continuous functions of several variables by superpositions of continuous functions of one variable and addition, Doklady Akademii Nauk SSSR 114 (1957). Original paper (in Russian).
- [9] C. J. Vaca-Rubio, L. Blanco, R. Pereira, M. Caus, Kolmogorov-arnold networks (kans) for time series analysis, 2024. URL: <https://arxiv.org/abs/2405.08790>. arXiv:2405.08790.
- [10] R. C. Yu, S. Wu, J. Gui, Residual kolmogorov-arnold network for enhanced deep learning, 2025. URL: <https://arxiv.org/abs/2410.05500>. arXiv:2410.05500.
- [11] A. Jamali, S. K. Roy, D. Hong, B. Lu, P. Ghamisi, How to learn more? exploring kolmogorov-arnold networks for hyperspectral image classification, 2024. URL: <https://arxiv.org/abs/2406.15719>. arXiv:2406.15719.
- [12] E. Poeta, F. Giobergia, E. Pastor, T. Cerquitelli, E. Baralis, A benchmarking study of kolmogorov-arnold networks on tabular data, in: 2024 IEEE 18th International Conference on Application of Information and Communication Technologies (AICT), 2024, pp. 1–6. doi:10.1109/AICT61888.2024.10740444.
- [13] A. Dahal, S. A. Murad, N. Rahimi, Efficiency bottlenecks of convolutional kolmogorov-arnold networks: A comprehensive scrutiny with imagenet, alexnet, lenet and tabular classification, 2025. URL: <https://arxiv.org/abs/2501.15757>. arXiv:2501.15757.
- [14] S. Somvanshi, S. A. Javed, M. M. Islam, D. Pandit, S. Das, A survey on kolmogorov-arnold network, ACM Comput. Surv. (2025). URL: <https://doi.org/10.1145/3743128>. doi:10.1145/3743128, just Accepted.
- [15] Z. Li, Kolmogorov-arnold networks are radial basis function networks, 2024. URL: <https://arxiv.org/abs/2405.06721>. arXiv:2405.06721.
- [16] S. T. Seydi, Z. Bozorgasl, H. Chen, Unveiling the power of wavelets: A wavelet-based kolmogorov-arnold network for hyperspectral image classification, 2024. URL: <https://arxiv.org/abs/2406.07869>. arXiv:2406.07869.
- [17] A. D. Bodner, A. S. Tepsich, J. N. Spolski, S. Pourteau, Convolutional kolmogorov-arnold networks, 2024. URL: <https://arxiv.org/abs/2406.13155>. arXiv:2406.13155.
- [18] M. Moradi, S. Panahi, E. Bollt, Y.-C. Lai, Kolmogorov-arnold network autoencoders, 2024. URL: <https://arxiv.org/abs/2410.02077>. arXiv:2410.02077.
- [19] Kaggle, Kaggle: Your machine learning and data science community, 2025. URL: <https://www.kaggle.com>.

A. Equations

As mentioned in Section 5, for this study, we selected the same 27 equations sampled from the Feynman dataset by the authors of the original KAN paper [4]. Here we report in detail what those equations are and their IDs.

Equation	Formula	Input Variables
2	$\exp(-\frac{\theta^2}{2\sigma^2})/\sqrt{2\pi\sigma^2}$	2
3	$\exp(-\frac{(\theta-\theta_1)^2}{2\sigma^2})/\sqrt{2\pi\sigma^2}$	3
5	$\frac{Gm_1m_2}{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$	9
12	$q(E_f + Bv\sin\theta)$	5
14	$Gm_1m_2(\frac{1}{r_2} - \frac{1}{r_1})$	5
17	$\frac{x-ut}{\sqrt{1+(\frac{u}{c})^2}}$	4
20	$\frac{u+v}{1+\frac{uv}{c^2}}$	3
21	$\frac{m_1r_1+m_2r_2}{m_1+m_2}$	4
26	$\arcsin(n\sin\theta_2)$	2
27	$\frac{1}{\frac{1}{d_1} + \frac{n}{d_2}}$	3
29	$\sqrt{x_1^2 + x_2^2 - 2x_1x_2\cos(\theta_1 - \theta_2)}$	4
30	$I_{*,0} \frac{\sin^2(\frac{n\theta}{2})}{\sin^2(\frac{\theta}{2})}$	3
31	$\arcsin(\frac{\lambda}{nd})$	3
38	$I_* = I_1 + I_2 + 2\sqrt{I_1I_2}\cos\delta$	4
43	$n_0\exp(-\frac{mgx}{k_bT})$	6
48	$nk_bT\ln(\frac{V_2}{V_1})$	5
51	$x_1(\cos(wt) + \alpha\cos^2(wt))$	4
52	$\frac{k(T_2-T_1)A}{d}$	5
56	$\frac{3}{4\pi\epsilon} \frac{p_d z}{r^5} \sqrt{x^2 + y^2}$	5
62	$n_0(1 + \frac{p_d E_f \cos\theta}{k_b T})$	6
64	$\frac{n\alpha}{1-n\alpha} \epsilon E_f$	4
80	$\frac{\hbar_0}{\exp(\frac{\mu_m B}{k_b T}) + \exp(-\frac{\mu_m B}{k_b T})}$	5
82	$\frac{\mu_m B}{k_b T} + \frac{\mu_m \alpha M}{\epsilon c^2 k_b T}$	8
84	$\frac{Y A x}{d}$	4
90	$\frac{p_d E_f}{h} \frac{\sin^2((w-w_0)t/2)}{((w-w_0)t/2)^2}$	6
91	$\mu_m \sqrt{B_x^2 + B_y^2 + B_z^2}$	4
98	$\beta(1 + \alpha\cos\theta)$	3

Table 1: List of equations we selected for our experiments, with corresponding formulas and the number of input variables.

B. Detailed performance for best hyperparameters

Table 2 reports the R_2 score, Test loss and training time of the best performing KAN and HybridKAN architectures for each equation, along with their hyperparameters. See section 5 for more details on the

hyperparameter space.

Eq	Width		Hyperparameters		R2 Score		Training Time (s)		Test Loss	
	HybridKAN	KAN	HybridKAN Neurons	KAN Grid Size	HybridKAN	KAN	HybridKAN	KAN	HybridKAN	KAN
2	(2,3,1)	(2,5,5,1)	(10,10)	5	99.90%	99.13%	45.86 s	145.04 s	0.0034	0.0103
3	(3,3,3,1)	(3,5,1)	(8,10)	10	99.92%	97.66%	63.79 s	109.39 s	0.0038	0.0204
5	(9,5,3,1)	(9,5,3,1)	(4,10)	5	99.08%	0.02%	183.20 s	112.19 s	0.0028	0.0290
12	(5,5,5,1)	(5,5,5,1)	(8,10)	5	98.20%	98.16%	187.58 s	116.71 s	0.0488	0.0488
14	(5,3,1)	(5,3,1)	(10,10)	10	99.63%	95.98%	61.93 s	58.57 s	0.0063	0.0209
17	(4,5,3,1)	(4,3,3,1)	(6,10)	3	99.77%	99.42%	121.39 s	85.71 s	0.0381	0.0610
20	(3,3,1)	(3,3,3,1)	(8,10)	3	99.90%	99.15%	49.21 s	89.16 s	0.0071	0.0210
21	(4,3,1)	(4,3,3,1)	(4,8)	3	99.98%	99.83%	47.19 s	65.20 s	0.0051	0.0166
26	(2,5,1)	(2,3,1)	(8,10)	5	99.91%	99.91%	59.97 s	52.72 s	0.0132	0.0133
27	(3,3,1)	(3,5,3,1)	(8,8)	3	99.91%	99.32%	47.47 s	130.20 s	0.0053	0.0149
29	(4,5,3,1)	(4,5,5,1)	(8,10)	10	97.49%	98.24%	123.18 s	201.11 s	0.0598	0.0500
30	(3,5,3,1)	(3,5,3,1)	(10,6)	5	96.66%	99.31%	150.16 s	105.79 s	0.0739	0.0335
31	(3,3,1)	(3,3,3,1)	(4,8)	3	99.93%	99.88%	41.72 s	66.71 s	0.0102	0.0138
38	(3,5,3,1)	(3,5,1)	(10,8)	5	99.96%	99.94%	105.39 s	89.66 s	0.0160	0.0203
43	(6,3,3,1)	(6,5,3,1)	(8,6)	10	99.95%	98.39%	130.66 s	103.19 s	0.0091	0.0037
48	(5,3,3,1)	(5,5,1)	(10,6)	10	99.71%	95.44%	123.59 s	135.36 s	0.0056	0.0222
51	(4,3,3,1)	(4,5,5,1)	(10,10)	10	99.22%	99.53%	146.55 s	284.97 s	0.0428	0.0203
52	(5,3,3,1)	(5,3,1)	(10,8)	15	99.86%	98.05%	140.71 s	104.41 s	0.0051	0.0193
56	(6,3,1)	(6,5,5,1)	(10,8)	3	99.00%	99.69%	120.58 s	85.27 s	0.0425	0.0227
62	(6,5,3,1)	(6,5,5,1)	(4,10)	10	96.97%	96.34%	217.40 s	155.12 s	0.0559	0.0614
64	(4,3,1)	(4,5,3,1)	(4,10)	3	99.88%	99.89%	93.13 s	134.86 s	0.0114	0.0111
80	(5,3,1)	(5,3,1)	(6,8)	10	99.91%	98.02%	98.28 s	87.67 s	0.0041	0.0197
82	(8,3,1)	(8,5,3,1)	(10,10)	5	99.14%	99.52%	111.96 s	146.69 s	0.0051	0.0277
84	(4,3,1)	(4,3,3,1)	(4,10)	5	99.85%	99.69%	58.61 s	66.86 s	0.0058	0.0085
90	(6,3,1)	(6,5,1)	(10,8)	10	99.21%	98.10%	80.38 s	66.77 s	0.0127	0.0197
91	(4,3,1)	(4,3,3,1)	(4,6)	5	99.75%	99.71%	48.64 s	77.87 s	0.0161	0.0171
98	(3,3,3,1)	(3,5,3,1)	(6,6)	10	99.77%	99.83%	78.95 s	111.40 s	0.0178	0.0146

Table 2: Best Performance on All Equations