Kolmogorov–Arnold Networks: Representation and Architecture

Peter W. Hoffman*
LIDS – MIT
Cambridge, MA
hoffmanp@mit.edu

Farrell Eldrian Wu*
MIT Operations Research Center
Cambridge, MA
farrellw@mit.edu

Petros Prastakos*
MIT Operations Research Center
Cambridge, MA
pprastak@mit.edu

1 Literature Review

Let us first introduce the Kolmogorov-Arnold representation theorem that serves as the foundation of the research in this area. The original theorem, proved by Kolmogorov and Arnold in (1), states that, if $f : [0,1]^n \to \mathbb{R}$ is a continuous function, there exist functions $\phi_{q,p} : [0,1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$ such that

$$f(x_1, ..., x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right). \tag{1}$$

Thus, the $n(2n+1)$ univariate functions $\phi_{p,q}$ and $2n+1$ univariate functions Φ_q are enough for an exact representation of a n-variate continuous function.

A series of variants of the original theorem then followed, which managed to simplify the number of inner and outer functions needed.

In 1965, Spretcher proved in (4) that one can replace the $n(2n+1)$ functions $\phi_{p,q}$ with a single inner function ϕ with a shift in the input, and the $2n+1$ outer functions Φ_q with a single function Φ. Specifically, in Theorem 1, he showed that, for any continuous function f, there exist real values ϵ, $\lambda_1, ..., \lambda_n$, a continuous function $\Phi : \mathbb{R} \to \mathbb{R}$, and a real, monotonically increasing continuous function $\phi : [0,1] \to [0,1]$ with the Hölder-continuous property that $|\phi(x) - \phi(y)| \leq |x-y|^{\frac{\ln(2)}{\ln(2N+2)}}$ $\forall x, y \in [0,1]$ for $N \geq n \geq 2$ such that

$$f(x_1, ..., x_n) = \sum_{q=0}^{2n} \Phi \left(\sum_{p=1}^{n} \lambda_p \phi(x_p + \epsilon q) + q \right). \tag{2}$$

More recently, Schmidt-Heiber in (3) constructed a new representation that allowed for the transfer of smoothness properties from f to Φ without assuming the inner function ϕ is continuous. Letting $\mathcal{C}$ denote the Cantor set and $0.a_1^x a_2^x ...$ denote the binary representation of x, in Theorem 2, he constructed a monotone function $\phi : [0,1] \to \mathcal{C}$ defined as $\phi(x) = \sum_{j=1}^{\infty} \frac{2a_j^x}{3^{1+d(j-1)}}$ such that for any function $f : [0,1]^d \to \mathbb{R}$, there exists a function $\Phi : \mathcal{C} \to \mathbb{R}$ such that $f(x_1, ..., x_n) = \Phi \left(3 \sum_{p=1}^{n} 3^{-p} \phi(x_p) \right)$. Replacing ϕ with ϕ_K, a finite sum analogue that uses the first K terms of the binary representation, and assuming that f is Hölder continuous (i.e. $|f(x) - f(y)| \leq Q\|x-y\|_\infty^\beta$ with $\beta \leq 1$), he

*Equal contribution.

Preprint. Under review.

showed that f can be well approximated with error $2Q2^{-\beta K}$. Furthermore, the interior function $3\sum_{p=1}^{n} 3^{-p}\phi_K(x_p)$ can then be computed using a deep ReLU network, where $\phi_K(x_i)$ is computed using a network with K hidden layers and width 3 for each $i \in [n]$, and an additional hidden layer is used to combine the results, yielding an overall network with $K+1$ hidden layers and width $3n$.

Inspired by the KA representation theorem, a novel neural network architecture was proposed in (2) as an alternative framework to multi-layer perceptrons (MLPs) that are inspired by the universal approximation theorem: Kolmogorov-Arnold Networks, or KANs. KANs present the significant advantage of usually allowing for much smaller computation graphs than MLPs, while also being fully connected. Instead of learning weights on edges and having fixed activation functions (e.g. ReLU) on nodes like MLPs, KANs learn activation functions on edges and simply do a sum operation on nodes.

Observe first that the KA representation result in Eq. 1 can then be formulated as a fully-connected KAN with 3 layers, where the first layer has n nodes, the second has $2n+1$ nodes, and the output layer has 1 node, with learned function $\phi_{q,p}$ on the edge connecting node p in the first layer to node q in the second layer and learned function Φ_q on the edge connecting node q in the second layer to the output layer (with sum operation applied to all the incoming post-activation values of the nodes in the second and output layer).

The overall structure of a KAN generalizes the original NN architecture of the KA representation theorem in Eq. 1 to $L+1$ layers instead of 3. Let $n_0, n_1, ..., n_L$ denote the number of nodes at each layer. Between any two layers l and $l+1$ and corresponding nodes $i \in [n_l]$ and $j \in [n_{l+1}]$ there exists a learned function $\phi_{l,j,i}$ that takes as input $x_{l,i}$ (i.e. the post activation value of the i-th node in the l-th layer) and outputs $\phi_{l,j,i}(x_{l,i})$. The post-activation of the j-th neuron in the $l+1$-th layer is then simply the sum of all incoming post-activations across all n_l nodes connected to it. That is,

$$x_{l+1,j} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}).$$

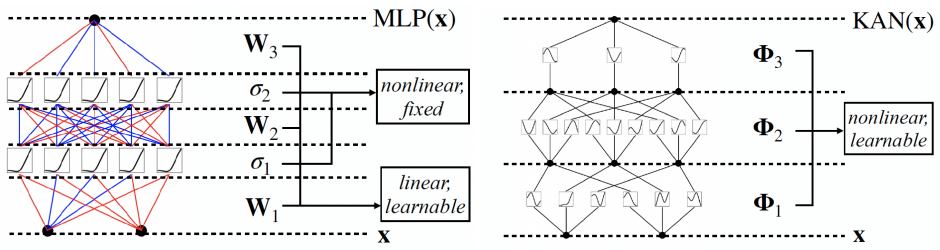
We can then define the outer function Φ_l corresponding to the l-th KAN layer as a function matrix of dimension $n_{l+1} \times n_l$ of $(\phi_{l,j,i})_{i\in[n_l],j\in[n_{l+1}]}$ that takes as input the vector of post activation values of all the nodes in the l-th layer and outputs the vector of post-activation values of all the nodes in the $l+1$-th layer. In matrix form, we have

$$\begin{pmatrix} x_{l+1,1} \\ x_{l+1,2} \\ \vdots \\ x_{l+1,n_{l+1}} \end{pmatrix} = \begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix} \begin{pmatrix} x_{l,1} \\ x_{l,2} \\ \vdots \\ x_{l,n_l} \end{pmatrix}$$

The full KAN output is then the composition of $\Phi_1, ..., \Phi_{L-1}$ across all the KAN layers in the network, ie $KAN(x) = (\Phi_{L-1} \circ \Phi_{L-2} \circ ... \circ \Phi_1 \circ \Phi_0)(x)$ or alternatively, in order to relate it more to the form in Eq.1,

$$KAN(x) = \sum_{i_{L-1}=1}^{n_{L-1}} \phi_{L-1,i_L,i_{L-1}} \left(\sum_{i_{L-2}=1}^{n_{L-2}} \phi_{L-2,i_{L-1},i_{L-2}} \left(... \left(\sum_{i_1=1}^{n_1} \phi_{1,i_2,i_1} \left(\sum_{i_0=1}^{n_0} \phi_{0,i_1,i_0} \right) \right) \right) \right).$$

A visual comparison of a 3-layer KAN vs MLP is shown in figure 1.



(a) (b)

Figure 1: (a) 3-layer MLP. (b) 3-layer KAN. (Source: (2))

To make this network easier to optimize, Liu et al. use linear combinations of silu and B-spline functions for the learned functions ϕ in the edges of the network. In other words,

$$\phi(x) = w_1 \frac{x}{1+e^{-x}} + w_2 \sum_i c_i B_i(x)$$

where $B_i(x)$ is a B-spline, and w_1, w_2, c_i are learnable.

2 Representation Results

2.1 Full Form of Approximation Result (Theorem 2.1 in (2))

We consider a KAN with L layers. Among the $n_\ell \times n_{\ell+1}$ potential functions in a given layer Φ_ℓ, some may be identically 0, indicating sparsity. We approximate the original function $f(x) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_1 \circ \Phi_0)(x)$ by the spline-approximated function $f^G(x) = (\Phi^G_{L-1} \circ \Phi^G_{L-2} \circ \cdots \circ \Phi^G_1 \circ \Phi^G_0)(x)$. Our goal is to bound $\|f - f^G\|_{C^0}$.

For $t \geq 1$, define $\overline{M}_{D^t} := \max_{l,i,j} \|D^t \phi_{l,i,j}\|_{C^0}$. By standard one-dimensional spline approximation results (Theorem 49 in (5)), we have $\|\phi_{l,i,j} - \phi^G_{l,i,j}\|_{C^0} \leq C_k M_{D^{k+1}} G^{-(k+1)} := \overline{M}_a$ and $\|D\phi_{l,i,j} - D\phi^G_{l,i,j}\|_{C^0} \leq C_k M_{D^{k+1}} G^{-k}$, where C_k is the order-k spline approximation constant.

By a triangle inequality, for the first derivative we get:

$$\|D\phi^G_{l,i,j}\|_{C^0} \leq \|D\phi_{l,i,j}\|_{C^0} + \|D\phi_{l,i,j} - D\phi^G_{l,i,j}\|_{C^0} \leq \overline{M}_{D^1} + C_k M_{D^{k+1}} G^{-k} := \overline{M}_b.$$

As done in Theorem 2.1 in (2), we next bound the term R_l defined below, where the difference is only at the l-th layer:

$$R_l := \|(\Phi^G_{L-1} \circ \cdots \circ \Phi^G_{\ell+1} \circ \Phi_\ell \circ \Phi_{\ell-1} \circ \cdots \circ \Phi_0) - (\Phi^G_{L-1} \circ \cdots \circ \Phi^G_{\ell+1} \circ \Phi^G_\ell \circ \Phi_{\ell-1} \circ \cdots \circ \Phi_0)\|_{C^0}.$$

Define the partial composition $\tilde{\Phi}_\ell := \Phi_{\ell-1} \circ \cdots \circ \Phi_0$, then rewrite: $R_\ell = \|(\Phi^G_{L-1} \circ \cdots \circ \Phi^G_{\ell+1} \circ \Phi_\ell \circ \tilde{\Phi}_\ell) - (\Phi^G_{L-1} \circ \cdots \circ \Phi^G_{\ell+1} \circ \Phi^G_\ell \circ \tilde{\Phi}_\ell)\|_{C^0}$. Focusing on the layer Φ_ℓ, we evaluate:

$$\|(\Phi_l)_i - (\Phi^G_l)_i\|_{C^0} = \sup_x \|\sum_{j=1}^{n_l} \phi_{l,j,i}(x) - \phi^G_{l,j,i}(x)\|_\infty \leq \overline{M}_a \sum_{j=1}^{n_l} 1[\phi_{l,j,i} \neq 0] = \overline{M}_a \text{paths}^k_{l,l},$$

where $\text{paths}^c_{a,b}$ is the count of paths from any node right before layer a to the index-c node after layer b. Then, define $\Phi'_{l,l'} := \Phi^G_{l'} \circ \cdots \circ \Phi^G_{l+1} \circ \Phi_l \circ \tilde{\Phi}_{l-1}$, and similarly, $\Phi'^G_{l,l'} := \Phi^G_{l'} \circ \cdots \circ \Phi^G_{l+1} \circ \Phi^G_l \circ \tilde{\Phi}_{l-1}$.

We prove that $\|(\Phi'_{l,l'})_i - (\Phi'^G_{l,l'})_i\|_{C^0} \leq \overline{M}_a \overline{M}_b^{l'-l} \text{paths}^i_{l,l'}$ by induction on $l' \geq l$, noting that the base case $l' = l$ has been covered. For the induction step,

$$\begin{aligned}
\|(\Phi'_{l,l'+1})_i - (\Phi'^G_{l,l'+1})_i\|_{C^0} &= \|(\Phi^G_{l'+1} \circ \Phi'_{l,l'})_i - (\Phi^G_{l'+1} \circ \Phi'^G_{l,l'})_i\|_{C^0} \\
&= \|\sum_{j=0}^{n_{l'}} \Phi^G_{l'+1,j,i} \circ (\Phi'_{l,l'})_j - \sum_{j=0}^{n'_l} \Phi^G_{l'+1,j,i} \circ (\Phi'^G_{l,l'})_j\|_{C^0} \\
&= \|\Phi^G_{l'+1,j,i} \sum_{j=0}^{n_{l'}} \circ [(\Phi'_{l,l'})_j - (\Phi'^G_{l,l'})_j]\|_{C^0} \\
&\leq \|\sum_{j=0}^{n_{l'}} 1[\phi_{l,j,i} \neq 0] \overline{M}_b (\overline{M}_a \overline{M}_b^{l'-l} \text{paths}^j_{l,l'})\|_{C^0} \text{ (by MVT and the induct. hypothesis)} \\
&= \overline{M}_a \overline{M}_b^{l'+1-l} \text{paths}^i_{l,l'+1}.
\end{aligned}$$

Taking $l' = L-1$ yields the bound $R_l \leq \overline{M}_a \overline{M}_b^{L-1-l} \text{paths}^1_{l,L-1}$, as the final layer only has 1 output. Taking the sum over R_l, as done in (2), yields

$$R = \sum_{l=0}^{L-1} R_l \leq \sum_{l=0}^{L-1} \overline{M}_a \overline{M}_b^{L-1-l} \text{paths}^1_{l,L-1} \leq L \overline{M}_a \max(1, \overline{M}_b)^L \text{paths}^1_{0,L-1},$$

where we used the monotonicity of paths. For G sufficiently large $\max(1, \overline{M}_b) \leq \max(1, 2M_{D^1})$, deriving the asymptotic constant $C = LC_k M_{D^{k+1}} \max(1, 2M_{D^1})^L \text{paths}_{0,L-1}^1$. The implications of this result is that a tighter approximation is guaranteed for sparse KAN's (which have exponentially fewer paths), and where function derivatives are bounded. (The first derivative, however, matters more as the bound for this derivative is raised to the Lth power.)

2.2 KAN Approximation Through Schmidt Heiber's Space-filling Curve Construction

In this section, we prove the following result:

For every $\epsilon > 0$ and every Hölder continuous f (i.e. there exists $Q \geq 0, \beta \in (0, 1]$ such that $|f(x) - f(y)| \leq Q\|x - y\|_\infty^\beta$ for every $x, y \in [0, 1]^n$), there exists an L-layer KAN such that

$$\|f(x) - (\Phi_{L-1} \circ \Phi_{L-2} \circ ... \circ \Phi_1 \circ \Phi_0)(x)\|_\infty < \epsilon$$

where Φ_ℓ are smooth functions with bounded derivatives up to order k (i.e. $\max_{i \in [k]} \sup_{x \in [0,1]} |D^i \Phi_\ell(x)| \leq M$).

Proof:

From Theorem 2 in (3), we have that given function $f : \mathbb{R}^n \to \mathbb{R}$ that is Hölder continuous with parameters Q, β, then it has a representation $f = g(3 \sum_{p=1}^n 3^{-p} \phi(x_p))$. Here, the inner function ϕ is defined as $\phi(x) := \sum_{j=1}^\infty \frac{2a_j^x}{3^{1+d(j-1)}}$ and the outer function g is Hölder continuous over the Cantor set, i.e. $|g(x) - g(y)| \leq 2^\beta Q_g |x - y|^{\beta_g} \forall x, y \in \mathcal{C}$, with parameters $Q_g := 2^\beta Q$ and $\beta_g = \frac{\beta \log 2}{n \log 3}$.

We first extend the function g, defined over $\mathcal{C}$ to a function $\tilde{g} : [0, 1] \to \mathbb{R}$ which is also Hölder continuous with parameters Q_g, β_g. To do so, define $\tilde{g}(x) = \inf_{c \in \mathcal{C}}(g(c) + Q_g |x - c|^{\beta_g})$. The function $\tilde{g}$, by its definition and the Hölder continuity of g, coincides with g on $\mathcal{C}$.

Next, we verify the Hölder continuity of $\tilde{g}$. For any $0 \leq x \leq y \leq 1$, and any $\epsilon > 0$, pick $c_x \in \mathcal{C}$ such that $\tilde{g}(x) \leq g(c_x) + Q_g |x - c_x|^{\beta_g} - \epsilon$. Then, $\tilde{g}(y) \leq g(c_x) + Q_g |y - c_x|^{\beta_g}$, and also $\tilde{g}(y) - \tilde{g}(x) \leq Q_g(|y - c_x|^{\beta_g} - |x - c_x|^{\beta_g}) + \epsilon$. Since the function $t \to |t|^{\beta_g}$ is non-decreasing and sub-additive, we have that $|y - c_x|^{\beta_g} - |x - c_x|^{\beta_g} \leq |y - x|^{\beta_g}$, so $\tilde{g}(y) - \tilde{g}(x) \leq Q_g |y - x|^{\beta_g} + \epsilon$. Taking $\epsilon \to 0$, $\tilde{g}(y) - \tilde{g}(x) \leq Q_g |y - x|^{\beta_g}$ and by symmetry, $|\tilde{g}(y) - \tilde{g}(x)| \leq Q_g |y - x|^{\beta_g}$ as desired. Thus, we now have a Hölder continuous outer function $\tilde{g}$ on $[0, 1]$.

Now, let g_d be the Bernstein polynomial approximation of $\tilde{g}$ of order d, defined by

$$g_d(x) = \sum_{i=1}^d \tilde{g}\left(\frac{i}{d}\right) \binom{d}{i} x^i (1 - x)^{d-i}.$$

By section 4 in (6), $|g_d(x) - \tilde{g}(x)| \leq 2 \sup_{|x-x'| < d^{-1/2}} |\tilde{g}(x) - \tilde{g}(x')|$ and by Hölder continuity we have $|g_d(x) - \tilde{g}(x)| \leq 2Q_g \sup_{|x-x'| < d^{-1/2}} |x - x'|^{\beta_g} \leq 2Q_g d^{-\beta_g/2}$. Now, observe that since polynomials and their derivatives are bounded in a closed domain, $\forall \epsilon > 0$, there exist $M_\epsilon < \infty, d$ such that for all $x \in [0, 1]$ and $i \in [k]$, $|g_d(x) - g(x)| \leq \epsilon$ and $|D^i g_d(x)| \leq M_\epsilon$.

We then proceed to approximating the inputs to g. As done in Lemma 4 in (3), we truncate the binary representation to K places, yielding the approximation $|f(x) - g(3 \sum_{p=1}^n 3^{-p} \phi_K(x_p))| \leq 2Q2^{-\beta K}$. which can be taken to 0 as $K \to \infty$.

Now finally, note that, in theorem 3 in (3), they approximate $\phi_K(x) = \sum_{i=1}^k \frac{2a_i}{3^{1+d(i-1)}}$ using a deep ReLU network. In our case, we approximate similarly using a modified version of the softplus function, defined by $\text{softplus}_\epsilon(x) = \epsilon \log(1 + e^{x/\epsilon})$.

3 Theory-Informed KAN Architecture

Equation 1 guarantees the existence of Φ_{inner} and Φ_{outer}, but says nothing about how easy it is to find them using optimization. In this section, we ask whether architecture motivated by theoretical results increase the ability of a network to learn a function's KA representation.

We draw on three insights to design our proposed architecture. First, Sprecher's result in 2 suggests that wider networks have inner functions with tighter Lipschitz properties. Since functional behavior

between known data points is controlled by the gradient, interpolation under tighter Lipschitz properties is easier, and thus these smoother representations should be easier to learn. Secondly, in the previous section, we showed that that increasing sparsity, measured by the number of paths from input layer to output layer, decreases the worst-case error incurred by using a spline KAN to approximate the KA representation. Thirdly, what if we apply the KA theorem on itself by using an additional KA representation to decompose $\phi_{q,p}(x_p)$ into even simpler pieces, as suggested by Daniel Mitropolsky. Suppose first that we project $x_p \mapsto (x_p, \ldots, x_p) \in \mathbb{R}^e$. Then applying the KA theorem in Equation 1 on $\phi'_{q,p}(x_p, \ldots, x_p)$ to obtain the KA-KA decomposition of $f(\vec{x})$ gives the following; the corresponding architecture is shown in Figure 7 in Appendix A.1:

$$f(\vec{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi'_{q,p}(x_p, \ldots, x_p) \right) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \left(\sum_{q'=1}^{2(e)+1} \Phi_{q'} \left(\sum_{p'=1}^{e} \phi_{q',p'}(x_p) \right) \right) \right).$$

While the preconditions of the KA theorem require that f is of at least two variables, $(x_p, \ldots, x_p)$ is still a one dimensional subspace of $\mathbb{R}^e$. Referring again to Figure 7 in Appendix A.1, we observe that the width of the second layer in the KAN recurrence architecture is given by $(2e+1)(n)(2n+1)$ since we have $(n)(2n+1)$ functions ϕ and each is broken into a sub-network with width $(2e+1)$. So the benefit of this projection is that it facilitates a wider, and hopefully smoother representation of each of these ϕ.

4 Architecture-Related Experiments

We now evaluate this KA recurrence architecture on the supervised learning of functions motivated by scientific-related tasks.

4.1 Dataset Generation

Suppose we wish to approximate the functions

$$f_1(x_1, x_2) = \exp(\exp(\cos(5x_1 + 2)) - x_1$$
$$f_2(x_1, x_2) = \cos(\exp(\sin(x_1 + x_2)x_1 + x_2))$$
$$f_3(x_1, x_2) = \cos(\sin(10x_2 \cos(2 * x_1) + 2x_2)).$$

These functions align with the physics-inspired tasks that KANs are frequently used for and contain components, such as trigonometric and exponential functions, that MLPs struggle with. These target functions are shown in Figure 2.

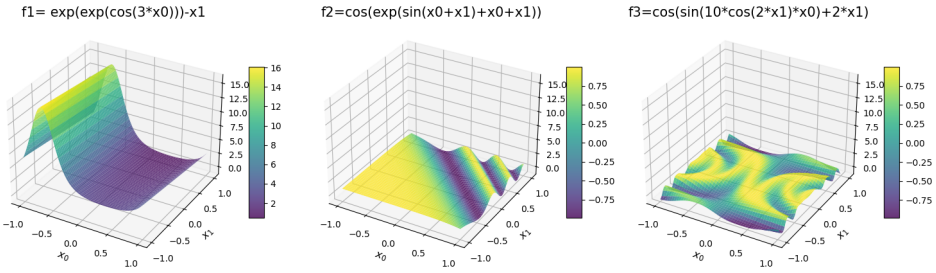


Figure 2: Visualization of target functions used for supervised approximation experiments.

For our training dataset generation, we drew 10,000 uniformly random points from $[-1, 1]^2$ and evaluated each of f_1, f_2, f_3 on this $10{,}000 \times 2$-dimensional vector to produce the corresponding $10{,}000 \times 1$ vectors of labels. We then ran the same procedure for our validation dataset.

4.2 Architecture

The question we seek to answer is whether the KA-recurrence architecture offers advantages over fully connected architectures. To this end, we introduce the first three networks we will examine

(Setup 1). The first is the KAN recurrence architecture with an expansion $e = 2$. The second architecture is a fully connected KAN with 6 hidden layers and 5 nodes per hidden layer. The third architecture is a fully connected KAN with 3 hidden layers and 10, 10, 4 nodes per hidden layer, respectively. It is trivial to check that each network has the same number of functions ± 1[1].

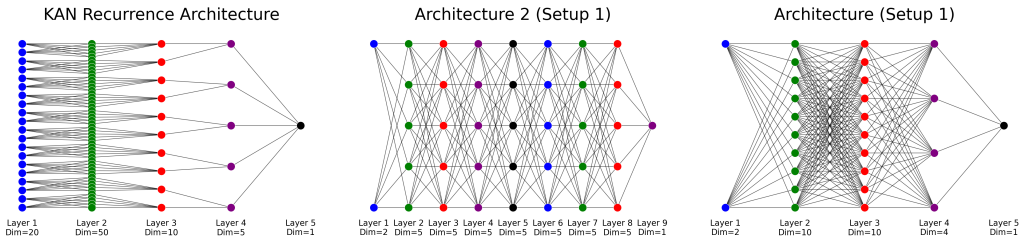


Figure 3: Architecture 1 is a KAN recurrence network with expansion $e = 2$. Architecture 2 is a 6-layer fully connected KAN with width 5. Architecture 3 is a 3-layer fully connected KAN with layer widths 10, 10, and 4, respectively.

4.3 Experimentation

Table 1 below shows the final validation loss for each architecture-function pair. For each function f_1, f_2, f_3, Architecture 1 achieved the lowest validation loss by a wide margin. Meanwhile, Architectures 2 and 3 had similar validation losses but with Architecture 3 (wide, fully connected) slightly outperforming Architecture 2 (deep, fully connected) in every case. Furthermore, we can

Table 1: (T1) Validation loss for each architecture-function pair. For each function considered, Architecture 1 achieves the lowest validation loss by a wide margin, followed by Architecture 3, then Architecture 2.

	Arch1 (Nested KAN)	**Arch2** (FC Deep)	**Arch3** (FC Wide)
TF1 f_1	0.1917	22.5770	10.2054
TF2 f_2	0.0149	0.4669	0.2181
TF3 f_3	0.0197	0.3394	0.1580

verify the KAN recurrence architecture learned a good representation of the data by noticing the 1-to-1 linear relationship between actual and predicted validation data, shown in Figure 4. More notable is that Architectures 2 and 3 almost didn't learn at all. Additional training runs produced similar results. Please refer to Appendix A.2 for details on training, including use of basis functions, Python implementation, and hyperparameter settings.

[1]Function totals across networks. Architecture 1: 100 + 50 + 10 + 5 = 165. Architecture 2: 10 + 6(25) + 5 = 165. Architecture 3: 20 + 100 + 40 + 4 = 164.

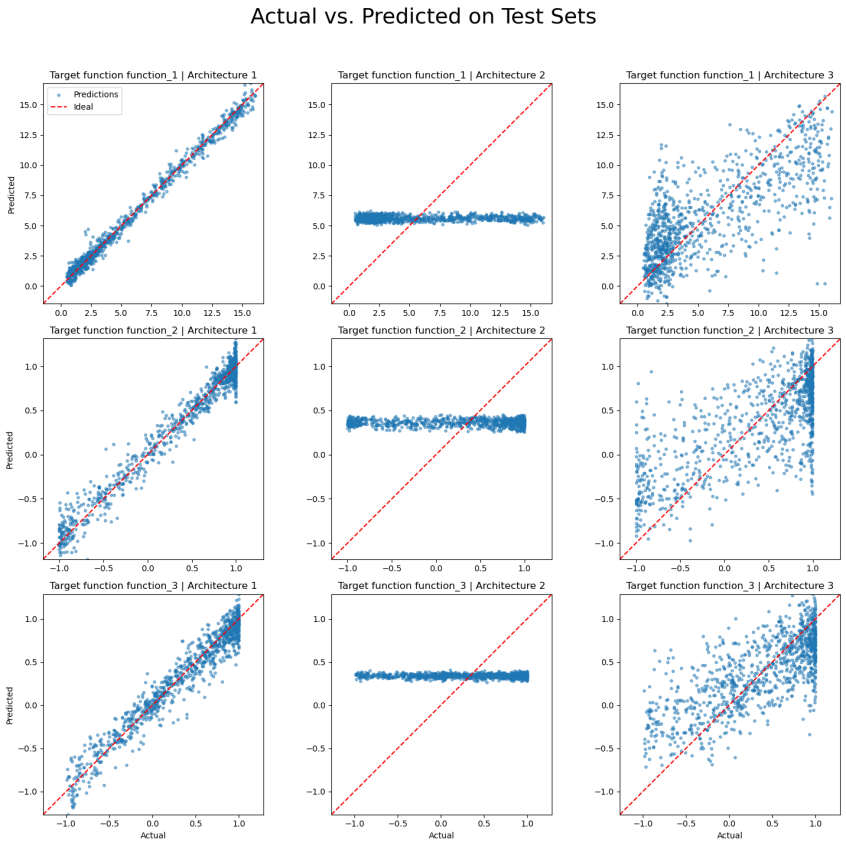


Figure 4: Actual vs. Predicted function values for each architecture-function pair. Architecture 1 consistently outperforms Architectures 2 and 3 across all functions.

4.4 Additional Experiments (Setup 2 and Setup 3)

We now devise two more Setups to investigate the role of width and sparsity in performance. Setup 2 compares the KAN recurrence architecture against two deep and sparse networks. Setup 3 compares the KAN recurrence architecture against two wide and sparse networks. The total number of parameters is held fixed across all architectures. The architectures for Setup 2 and Setup 3 are shown in Figure 5 and Figure 6, respectively. In Appendix A.4, Figure 9 and Figure 10 show training loss

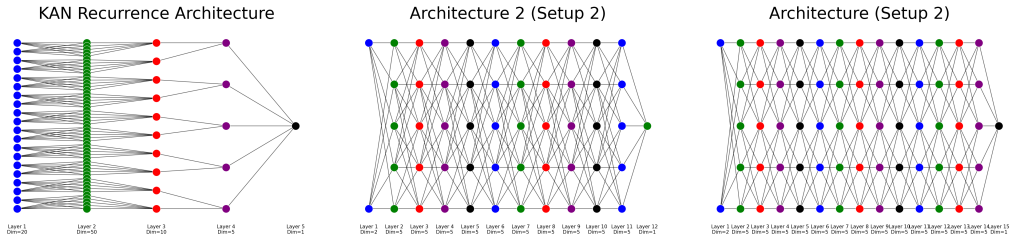


Figure 5: Setup 2 Architecture (Deep, sparse). Specifically, architecture 3 is deeper and more sparse than architecture 2.

for each setup, respectively, Table 2 and Table 3 show the validation loss for each setup, respectively, and Figure 11 and Figure 12 show the actual vs predicted plot for each setup, respectively.

Interestingly, Architecture 1 (KAN recurrence) performed best for all but one function: function 1 in Setup 3 (wide networks). However, this was not due to instability: the KAN recurrence architecture, which was held constant in all three setups, achieved nearly the *exact* same validation loss on each respective function across the different runs. Instead, this was due to Architecture 2 and Architecture

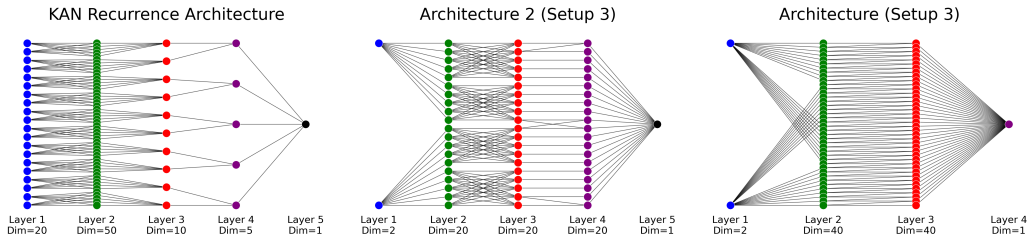


Figure 6: Setup 3 Architecture (Wide, sparse). Specifically, architecture 3 is wider and more sparse than architecture 2.

3 achieving almost perfect validation accuracy on function 1 in Setup 3. Meanwhile, on the other two functions in Setup 3, Architecture 2 and Architecture 3 did about as poorly as in the other setups. Furthermore, these results are robust: we re-ran Setup 3 an additional 2 times, generating Setup 3a and 3b, but no difference in results occurred as shown in Table 4 and Table 5 displayed in Appendix A.4. To summarize, Architecture 1 performed well and is consistent. The only only architecture to learn anything, as evidenced by Figure 11 and Figure 12, was Architecture 2 and Architecture 3 in Setup 3, but *only* on function 1.

4.5 Main Findings

First, these results suggest that sparsity alone doesn't entirely dictate performance, as evidenced by the nearly identical performance of Architecture 2 and Architecture 3 across all setups despite Architecture 3 always being more sparse than Architecture 2. Additionally, these results also cast doubt on the benefit of width: in Setup 3, Architecture 2, which had half the width of Architecture 3, did almost identically.

Perhaps the most clear result is that Architecture 1 was the only architecture of the 7 considered to learn a good representation of the data for every function. When conducting small-scale machine learning, such as function learning of physics-inspired processes, perhaps the most obvious architecture would be a fully connected network that balances depth and width. However, put lightly, the above results suggest that this may not be the best choice, and put more strongly, the above results suggest that fully connected architectures might cause you to learn nothing at all.

References

[1] A. N. Kolmogorov and V. I. Arnold, *Representation of Multidimensional Continuous Functions as Superpositions of Univariate Functions*, Moscow University Press, 1957.

[2] Z. Liu, et al., *KAN: Kolmogorov-Arnold Networks*, arXiv preprint arXiv:2404.19756, 2024.

[3] J. Schmidt-Hieber, *The Kolmogorov-Arnold Representation Theorem Revisited*, arXiv preprint arXiv:2007.15884, 2021.

[4] D. Sprecher, *On the structure of continuous functions of several variables*, Transactions of the American Mathematical Society, 1965

[5] T. Lyche, et al., *Foundations of Spline Theory: B-Splines, Spline Approximation, and Hierarchical Refinement*, Springer Nature, 2017

[6] A. Pallini, *Bernstein-type approximation of smooth functions*, Statistica, 2005

A Appendix

A.1 Derivation of KA-recurrence architecture

The original KAN architecture (LEFT) represents an $n = 2$ dimension function using $2n + 1 = 5$ hidden nodes. Therefore, there are $(n)(2n + 1) = 10$ learned $\phi(x)$ in the first layer and $(2n + 1) = 5$ learned Φ in the second layer. In the KAN recurrence architecture (RIGHT), we represent each of these 10 learned $\phi(x)$ using a second KA decomposition with expansion $e = 2$, shown in the red box. The black nodes with $+$ denote the sum

operation. To preserve that each of the five downstream Φ in the final layer "see" both x_1 and x_2 we use the projection $[x_1, x_2] \mapsto [x_1, x_1, x_2, x_2, \ldots x_1, x_1, x_2, x_2]$.

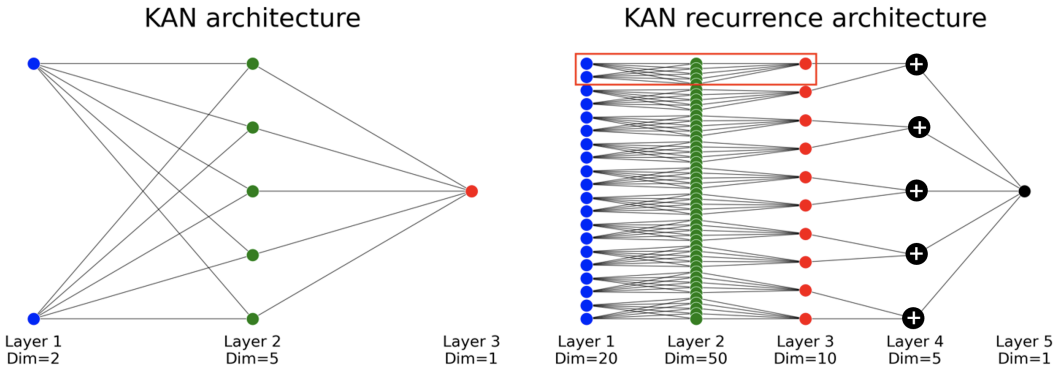


Figure 7: (LEFT) The original KAN architecture and the KA recurrence architecture.

A.2 KAN implementation

Since deep KAN architecture is still very new, having been first introduced in May 2024, the only existing Python package was Pykan. However, we found these methods generally unsuitable for our needs since they did not support masking or weight sharing. Therefore, we coded the KAN net and layer classes ourselves while maintaining compatibility with backpropagation and used the Adam optimizer. Our code is available here.

As opposed to polynomial basis functions, we used the trigonometric basis functions

$$B_{k,c}(x) = \cos^k(x) \tag{3}$$
$$B_{k,s}(x) = \sin^k(x) \tag{4}$$

for $k = 1, 2, 2, \ldots$. We found experimentally that this produced more stable results than spline basis functions, avoided the necessity of grid interval updates, yet still allowed the option to increase basis granularity during training.

We trained each network locally on the GPU for 50 epochs, with a batch size of 10, a learning rate of 10^{-3}, the Adam optimizer, and used the Mean Squared Error (MSE) loss. The total training time across all functions and all architectures was less than 1 hour.

A.3 KAN training and testing (Setup 1)

We first discuss the training loss, shown in Figure 8. In the initial epochs, the rate of training loss descent was consistent across Architecture 1 (BLUE), Architecture 2 (ORANGE), and Architecture 3 (GREEN) for every function investigated. However, after about epoch 15 for each function, Architectures 2 and 3 began to plateau while Architecture 1 continued to decrease. After 50 epochs, it was clear across all functions that Architecture 1 had achieved a lower training loss than Architectures 2 and 3. We also speculate, given the right-tail behavior of the blue lines, that the training loss of Architecture 1 would have continued to decrease if given additional epochs. Furthermore, it is encouraging that all three trend lines are smooth and non-intersecting, suggesting that the optimization process is free from unstable learning rates or exploding gradients.

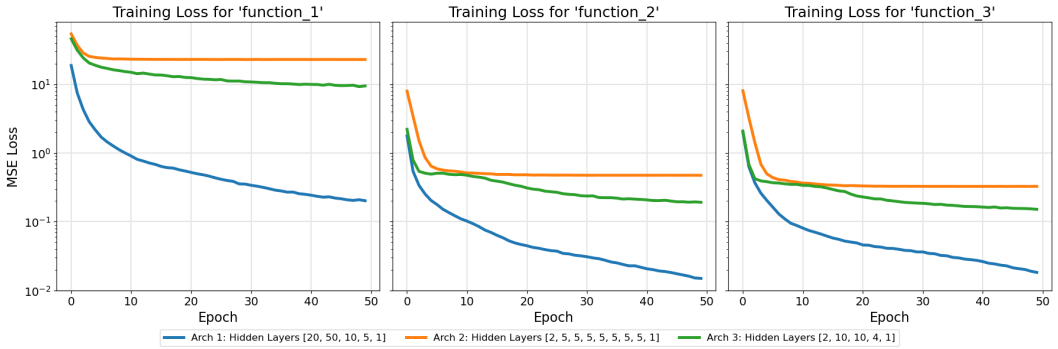


Figure 8: (Setup 1) Training loss across epochs for each architecture and function. In every function, Architecture 1 (BLUE) achieves a lower training loss than Architecture 3 (GREEN), which in turn outperforms Architecture 2 (ORANGE).

A.4 KAN training and testing (Setup 2,3,3a,3b)

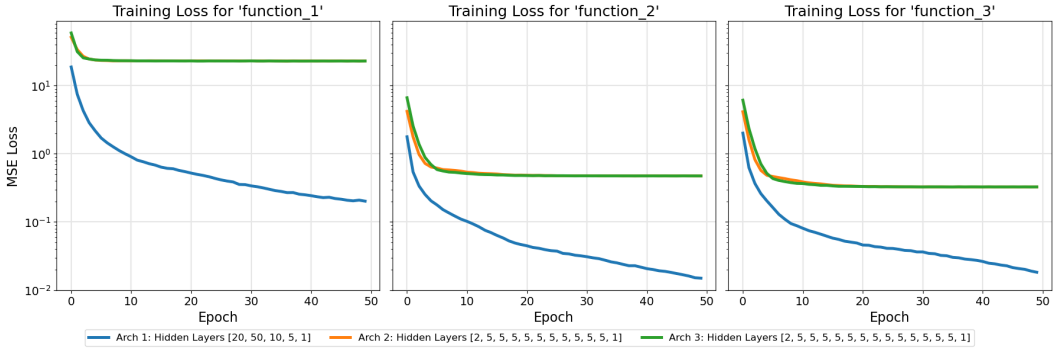


Figure 9: (Setup 2) Training loss across epochs for each architecture and function. In every function, Architecture 1 (BLUE) achieves the lowest training loss.

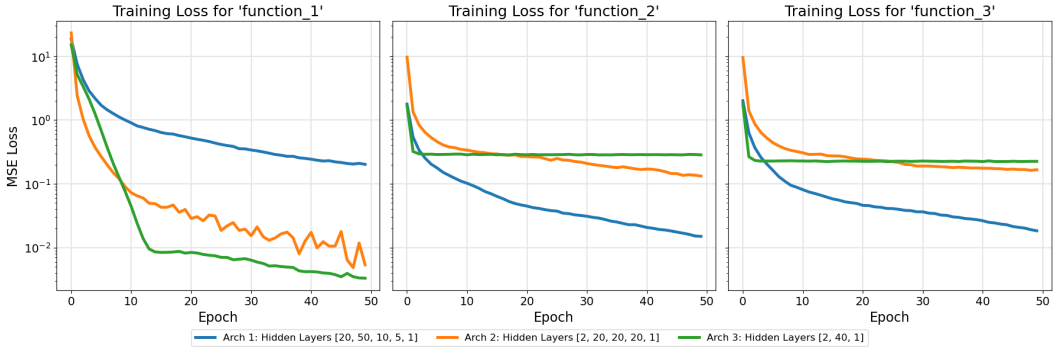


Figure 10: (Setup 3) Training loss across epochs for each architecture and function. Function 1 is the only instance where Architecture 1 (BLUE) does not achieve the lowest training loss (including across other Setups).

Table 2: (Setup 2 validation loss)

	Arch1 (Nested KAN)	**Arch2** (FC Deep)	**Arch3** (FC Wide)
TF1 f_1	0.1917	22.5545	22.5222
TF2 f_2	0.0149	0.4665	0.4633
TF3 f_3	0.0197	0.3372	0.3379

Table 3: (Setup 3 validation loss)

	Arch1 (Nested KAN)	**Arch2** (FC Deep)	**Arch3** (FC Wide)
TF1 f_1	0.1917	0.0060	0.0028
TF2 f_2	0.0149	0.1298	0.2620
TF3 f_3	0.0197	0.1677	0.2326

Table 4: (Setup 3a) Validation loss)

	Arch1 (Nested KAN)	**Arch2** (FC Deep)	**Arch3** (FC Wide)
TF1 f_1	0.1917	0.0061	0.0028
TF2 f_2	0.0149	0.1297	0.2620
TF3 f_3	0.0197	0.1677	0.2325

Table 5: (Setup 3b) Validation loss)

	Arch1 (Nested KAN)	**Arch2** (FC Deep)	**Arch3** (FC Wide)
TF1 f_1	0.1917	0.0060	0.0027
TF2 f_2	0.0145	0.1297	0.2620
TF3 f_3	0.0197	0.1679	0.2322

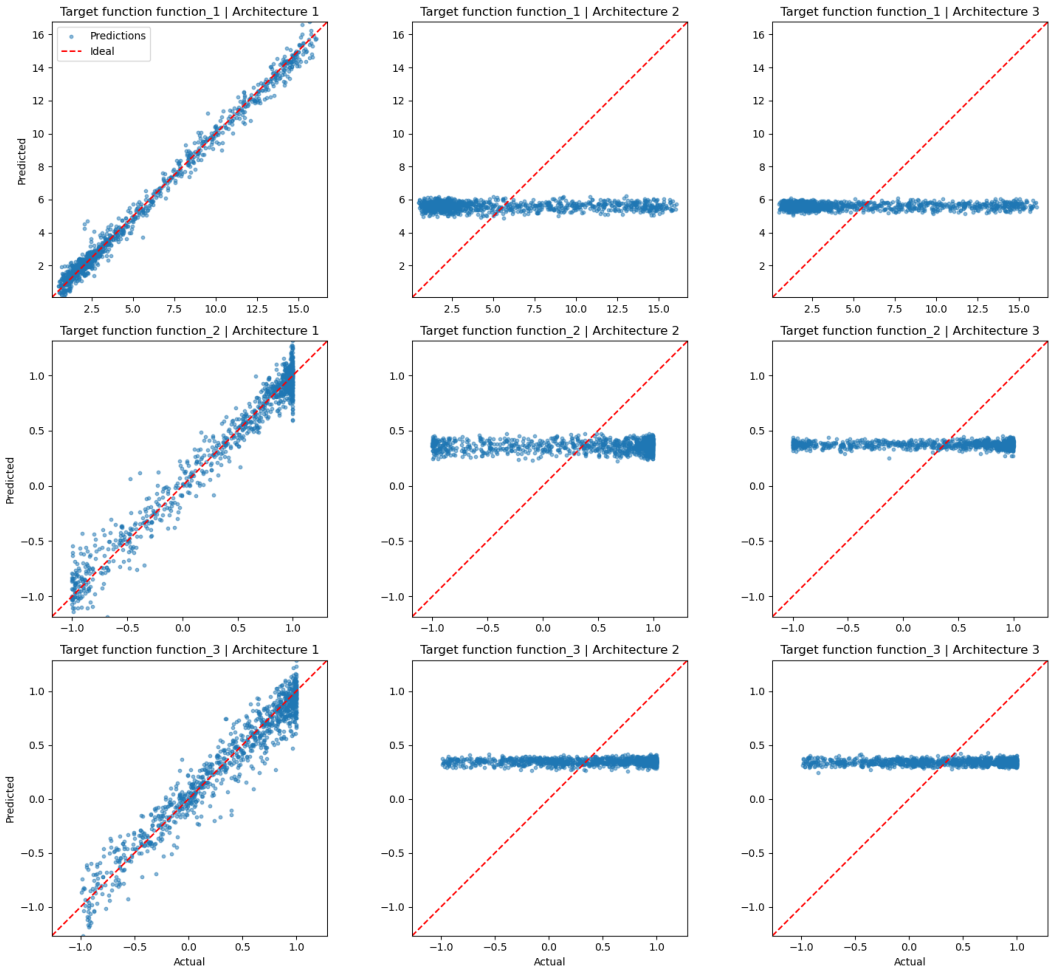


Figure 11: Setup 2

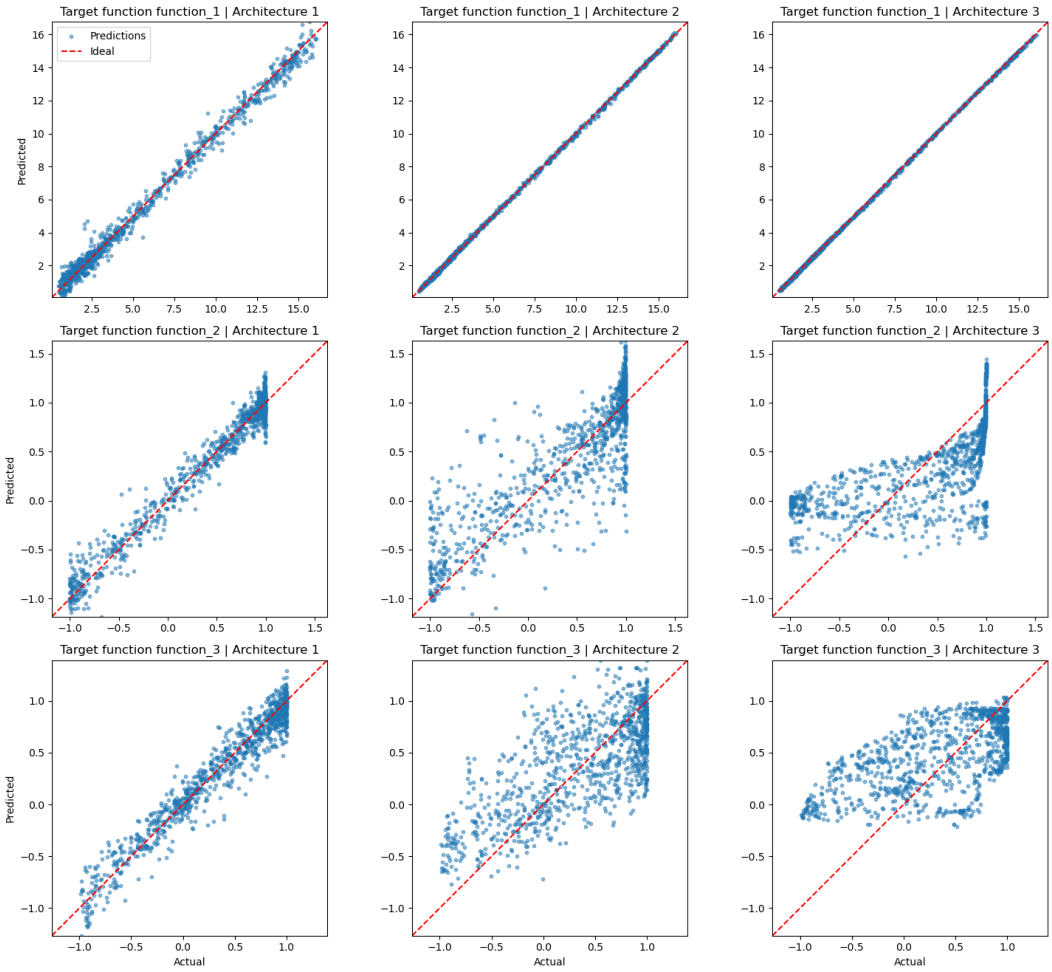


Figure 12: Setup 3