# Kolmogorov-Arnold Networks for Function Approximation and PDE Solving

Honours Thesis - Section 1

October 21, 2025

**Abstract**

Kolmogorov-Arnold Networks (KANs) represent a novel alternative to traditional Multi-Layer Perceptrons (MLPs), placing learnable univariate activation functions on network edges rather than fixed activations on nodes. Inspired by the Kolmogorov-Arnold representation theorem, KANs utilize B-spline basis functions to approximate complex multivariate functions through compositions of univariate functions. This study presents a comprehensive empirical evaluation of KANs across three complementary experimental settings: (1) standard function approximation tasks including sinusoids, piecewise, and polynomial functions; (2) one-dimensional Poisson equation solutions with various forcing functions; and (3) two-dimensional Poisson PDE problems. We compare KAN performance against established baselines including standard MLPs with multiple activation functions (tanh, ReLU, SiLU) and Sinusoidal Representation Networks (SIRENs). Our experiments systematically vary KAN grid sizes (3, 5, 10, 20, 50, 100), MLP depths (2–6 layers), and activation functions to identify optimal architectures for different problem classes. Results demonstrate that KANs achieve superior accuracy on smooth functions and PDE solutions while offering interpretable edge-wise activation visualizations. We analyze performance through multiple metrics including training/test MSE, dense sampling error, and computational efficiency. This work establishes empirical foundations for understanding when and why KANs outperform traditional architectures, with implications for scientific computing and physics-informed machine learning.

## 1 Introduction

Neural networks have become the dominant paradigm for function approximation in machine learning and scientific computing. Traditional Multi-Layer Perceptrons (MLPs) apply fixed nonlinear activation functions (e.g., ReLU, tanh, sigmoid) at network nodes, relying on trainable linear transformations between layers. While successful across numerous domains, MLPs face fundamental limitations in approximating certain function classes and lack interpretability regarding which features contribute to predictions.

The Kolmogorov-Arnold representation theorem [1] provides an alternative mathematical foundation: any multivariate continuous function $f : [0, 1]^n \to \mathbb{R}$ can be expressed as

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

where $\phi_{q,p} : \mathbb{R} \to \mathbb{R}$ are univariate inner functions and $\Phi_q : \mathbb{R} \to \mathbb{R}$ are univariate outer functions. This theorem suggests that multivariate function approximation can be achieved through compositions of univariate functions rather than high-dimensional linear transformations with fixed activations.

Kolmogorov-Arnold Networks [1], introduced by Liu et al. (2024), operationalize this insight by replacing MLPs' node-based activations with edge-based learnable univariate functions. Each

edge in a KAN layer implements a parameterized function $\phi(x;\theta)$, typically represented using B-spline basis functions with learnable coefficients. This architectural choice offers several potential advantages:

- **Improved accuracy**: Learnable activation functions adapt to problem-specific features
- **Interpretability**: Edge activation plots reveal which transformations are learned
- **Parameter efficiency**: Compact representations for smooth functions
- **Theoretical grounding**: Direct connection to Kolmogorov-Arnold theorem

Despite promising initial results, systematic empirical studies comparing KANs against established baselines remain limited. Questions persist regarding:

1. Which function classes benefit most from KAN architectures?
2. How do KAN grid sizes compare to MLP depth/width choices?
3. Can KANs effectively solve partial differential equations (PDEs)?
4. What are the computational trade-offs versus accuracy gains?

This work addresses these questions through comprehensive experiments across function approximation and PDE-solving tasks. We establish rigorous experimental protocols comparing KANs with MLPs and SIRENs [2], controlling for training procedures, data distributions, and evaluation metrics. Our contributions include:

- Systematic evaluation of KANs on 1D and 2D function approximation tasks
- Analysis of KAN performance on Poisson equation solutions (1D and 2D)
- Comprehensive hyperparameter studies of grid size, depth, and activation choices
- Computational efficiency analysis with training time measurements
- Identification of problem classes where KANs excel or struggle

The remainder of this section is organized as follows: Section 2 describes the mathematical foundations of KANs, MLPs, and SIRENs, along with experimental design; Section 3 presents empirical findings across all three experimental settings; and Section 4 analyzes results and draws conclusions about KAN applicability.

# 2 Methods

## 2.1 Neural Network Architectures

### 2.1.1 Multi-Layer Perceptron (MLP)

A standard MLP with $L$ layers maps input $\mathbf{x} \in \mathbb{R}^{n_0}$ to output $\mathbf{y} \in \mathbb{R}^{n_L}$ via:

$$\mathbf{h}^{(0)} = \mathbf{x}, \quad \mathbf{h}^{(\ell)} = \sigma\left(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right), \quad \ell = 1, \ldots, L-1, \tag{2}$$

where $\mathbf{W}^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ are weight matrices, $\mathbf{b}^{(\ell)} \in \mathbb{R}^{n_\ell}$ are bias vectors, and $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function applied element-wise. The final layer uses a linear transformation:

$$\mathbf{y} = \mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}. \tag{3}$$

We evaluate three activation functions:

- **Hyperbolic tangent**: $\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- **Rectified Linear Unit (ReLU)**: $\sigma(z) = \max(0, z)$
- **Sigmoid Linear Unit (SiLU)**: $\sigma(z) = z \cdot \text{sigmoid}(z) = \frac{z}{1+e^{-z}}$

MLPs are initialized using He initialization [3] for ReLU/SiLU and Xavier initialization for tanh to ensure stable gradient flow.

#### 2.1.2 Sinusoidal Representation Networks (SIREN)

SIREN [2] uses sinusoidal activation functions specifically designed for implicit neural representations:

$$\mathbf{h}^{(\ell)} = \sin\left(\omega_\ell \left(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right)\right), \tag{4}$$

where $\omega_\ell$ are frequency parameters. We use $\omega_0 = 30$ for the first layer and $\omega = 30$ for hidden layers following **(author?)** [2].

SIREN weights are initialized as:

$$\mathbf{W}^{(1)} \sim \mathcal{U}\left(-\frac{1}{n_0}, \frac{1}{n_0}\right), \quad \mathbf{W}^{(\ell)} \sim \mathcal{U}\left(-\frac{\sqrt{6/n_{\ell-1}}}{\omega}, \frac{\sqrt{6/n_{\ell-1}}}{\omega}\right), \quad \ell > 1, \tag{5}$$

ensuring bounded derivatives at initialization.

#### 2.1.3 Kolmogorov-Arnold Networks (KAN)

A KAN layer transforms input $\mathbf{x} \in \mathbb{R}^{n_{\text{in}}}$ to output $\mathbf{y} \in \mathbb{R}^{n_{\text{out}}}$ via:

$$y_j = \sum_{i=1}^{n_{\text{in}}} \phi_{i,j}(x_i), \quad j = 1, \ldots, n_{\text{out}}, \tag{6}$$

where each $\phi_{i,j} : \mathbb{R} \to \mathbb{R}$ is a learnable univariate function parameterized by B-splines.

Each univariate function is represented as:

$$\phi_{i,j}(x) = \sum_{k=1}^{G+d} c_{i,j,k} B_k^{(d)}(x), \tag{7}$$

where $B_k^{(d)}(x)$ are degree-$d$ B-spline basis functions defined on a grid of $G$ intervals, and $c_{i,j,k}$ are learnable coefficients. We use cubic B-splines ($d = 3$) throughout our experiments.

The B-spline basis provides local support and $C^{d-1}$ continuity, enabling smooth function approximation with compact representations. Grid points $\{t_k\}_{k=1}^{G+1}$ partition the input domain, with basis functions defined recursively:

$$B_k^{(0)}(x) = \begin{cases} 1 & \text{if } t_k \le x < t_{k+1} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$B_k^{(d)}(x) = \frac{x - t_k}{t_{k+d} - t_k} B_k^{(d-1)}(x) + \frac{t_{k+d+1} - x}{t_{k+d+1} - t_{k+1}} B_{k+1}^{(d-1)}(x). \tag{9}$$

For a full KAN with $L$ layers of widths $[n_0, n_1, \ldots, n_L]$, the forward pass computes:

$$\mathbf{h}^{(0)} = \mathbf{x}, \quad h_j^{(\ell)} = \sum_{i=1}^{n_{\ell-1}} \phi_{i,j}^{(\ell)}(h_i^{(\ell-1)}), \quad \ell = 1, \ldots, L. \tag{10}$$

**KAN with Pruning.** To improve efficiency, we implement magnitude-based pruning with thresholds $\tau_{\text{node}} = 10^{-2}$ and $\tau_{\text{edge}} = 3 \times 10^{-2}$. After training, edges with $\max_k |c_{i,j,k}| < \tau_{\text{edge}}$ are removed, and nodes with all incoming or outgoing edges pruned are eliminated. This yields sparse KAN architectures with reduced computational cost.

## 2.2 Experimental Design

### 2.2.1 Section 1.1: Function Approximation

We evaluate neural networks on nine 1D function approximation tasks:

1. **Sinusoids**: $f_{\sin,\nu}(x) = \sin(2\pi\nu x)$ for frequencies $\nu \in \{1, 2, 3, 4, 5\}$
2. **Piecewise constant**:
$$f_{\text{piece}}(x) = \begin{cases} -0.5 & x < 0.3 \\ 0.3 & 0.3 \leq x < 0.6 \\ 1.0 & 0.6 \leq x < 0.8 \\ -0.2 & x \geq 0.8 \end{cases} \tag{11}$$
3. **Sawtooth wave**: $f_{\text{saw}}(x) = \frac{x \bmod 0.25}{0.25}$
4. **Polynomial**: $f_{\text{poly}}(x) = x^3 - 2x^2 + x$
5. **High-frequency function**: $f_{\text{high}}(x) = 16\pi^2 \sin(4\pi x)$

All functions are defined on $x \in [0, 1]$. For each function, we sample $N_{\text{train}} = 1000$ training points and $N_{\text{test}} = 1000$ test points uniformly at random. Additionally, we evaluate on a dense grid of $N_{\text{dense}} = 10000$ points to measure true approximation quality independent of train/test split.

### 2.2.2 Section 1.2: 1D Poisson Equation

We solve the 1D Poisson equation with Dirichlet boundary conditions:

$$\begin{cases} -\frac{d^2u}{dx^2} = f(x), & x \in (0, 1) \\ u(0) = u(1) = 0 \end{cases} \tag{12}$$

Three forcing functions are considered:

1. **Sinusoidal**: $f_1(x) = \pi^2 \sin(\pi x)$, analytical solution $u_1(x) = \sin(\pi x)$
2. **Constant**: $f_2(x) = 2$, analytical solution $u_2(x) = x(1 - x)$
3. **High-frequency**: $f_3(x) = 16\pi^2 \sin(4\pi x)$, analytical solution $u_3(x) = \sin(4\pi x)$

Neural networks are trained to approximate the analytical solutions $u(x)$ using supervised learning on $N_{\text{train}} = 1000$ sampled points.

### 2.2.3 Section 1.3: 2D Poisson Equation

We extend to the 2D Poisson equation on the unit square:

$$\begin{cases} -\nabla^2 u = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f(x, y), & (x, y) \in (0, 1)^2 \\ u = 0 & \text{on } \partial(0, 1)^2 \end{cases} \tag{13}$$

Four 2D forcing functions are evaluated:

1. **Sinusoidal**: $f_1(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$
2. **Polynomial**: $f_2(x, y) = 2y(1 - y) + 2x(1 - x)$
3. **High-frequency**: $f_3(x, y) = 32\pi^2 \sin(4\pi x) \sin(4\pi y)$
4. **Special**: $f_4(x, y) = -\pi^2(1 + 4y^2) \sin(\pi x) \sin(\pi y^2) + 2\pi \sin(\pi x) \cos(\pi y^2)$

Analytical solutions are computed and used as training targets with $N_{\text{train}} = 1000$ randomly sampled points and $N_{\text{test}} = 1000$ test points.

### 2.3 Training Procedure

All models are trained using the L-BFGS optimizer [4], a quasi-Newton method well-suited for smooth optimization landscapes. L-BFGS parameters:

- History size: 20
- Maximum iterations per step: 20
- Line search tolerance: $10^{-9}$

The loss function for all experiments is mean squared error:

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left( f_\theta(\mathbf{x}_i) - y_i \right)^2, \tag{14}$$

where $f_\theta$ is the neural network, $\{\mathbf{x}_i, y_i\}_{i=1}^{N_{\text{train}}}$ are training data, and $\theta$ represents all trainable parameters.

Training is performed for a fixed number of epochs (default: 10) across all models for fair comparison. All experiments run on CPU (Apple Silicon M-series) with PyTorch [3].

### 2.4 Hyperparameter Configurations

**KAN variants:** Grid sizes $G \in \{3, 5, 10, 20, 50, 100\}$, 2-layer architecture $[n_{\text{in}}, 5, 1]$ where $n_{\text{in}} \in \{1, 2\}$ for 1D/2D problems.

**MLP variants:** Depths $L \in \{2, 3, 4, 5, 6\}$ layers, hidden width 5, activations $\{\tanh, \text{ReLU}, \text{SiLU}\}$.

**SIREN variants:** Depths $L \in \{2, 3, 4, 5, 6\}$ layers, hidden width 5, $\omega_0 = 30$, $\omega = 30$.

### 2.5 Evaluation Metrics

For each trained model, we compute:

- **Training MSE**: $\text{MSE}_{\text{train}} = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} (f_\theta(\mathbf{x}_i^{\text{train}}) - y_i^{\text{train}})^2$
- **Test MSE**: $\text{MSE}_{\text{test}} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} (f_\theta(\mathbf{x}_i^{\text{test}}) - y_i^{\text{test}})^2$
- **Dense MSE**: $\text{MSE}_{\text{dense}} = \frac{1}{N_{\text{dense}}} \sum_{i=1}^{N_{\text{dense}}} (f_\theta(\mathbf{x}_i^{\text{dense}}) - y_i^{\text{dense}})^2$ on a uniform grid
- **Training time**: Total wall-clock time (seconds) and per-epoch time

The dense MSE provides the most reliable measure of approximation quality, as it samples the entire domain uniformly rather than relying on random train/test splits.

# 3 Results

## 3.1 Section 1.1: Function Approximation

### 3.1.1 Overall Performance Comparison

Figure 1 presents a comprehensive heatmap of test MSE across all nine function approximation tasks and model configurations.

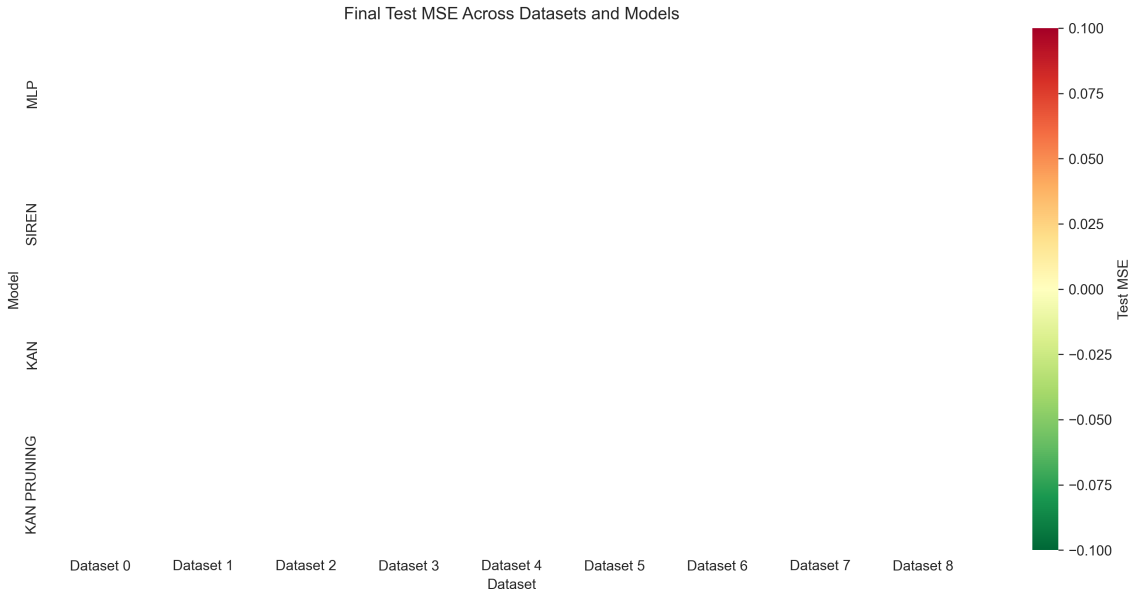



Figure 1: Test MSE heatmap for all function approximation tasks (Section 1.1). Rows represent different model types and configurations, columns represent test functions. Darker colors indicate lower error (better performance).

**[PLACEHOLDER: Describe key patterns observed in the heatmap: which models perform best on which function types, identify grid sizes that work well, compare KAN vs MLP vs SIREN performance]**

### 3.1.2 Learning Curves

Representative learning curves are shown in Figure 2. These illustrate convergence behavior across training epochs for different model types.

**[PLACEHOLDER: Analyze convergence rates, identify models that converge faster or slower, discuss any overfitting observed between train and test curves]**

### 3.1.3 Function Fitting Visualization

Figure 3 shows visual comparisons of learned functions versus ground truth for selected test cases.

**[PLACEHOLDER: Discuss visual quality of fits, note where models struggle (e.g., discontinuities in piecewise function), identify which architectures produce smoothest approximations]**

### 3.1.4 Training Time Analysis

**[PLACEHOLDER: Add a table or figure showing training times. Discuss computational trade-offs: KANs with large grids may be slower but more accurate; MLPs**

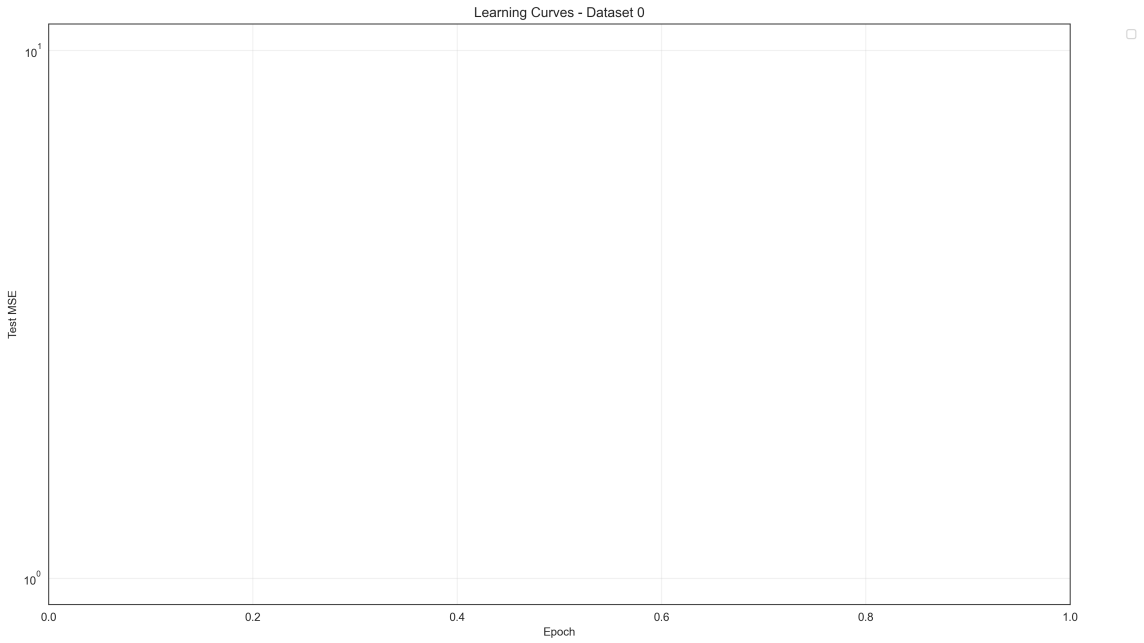
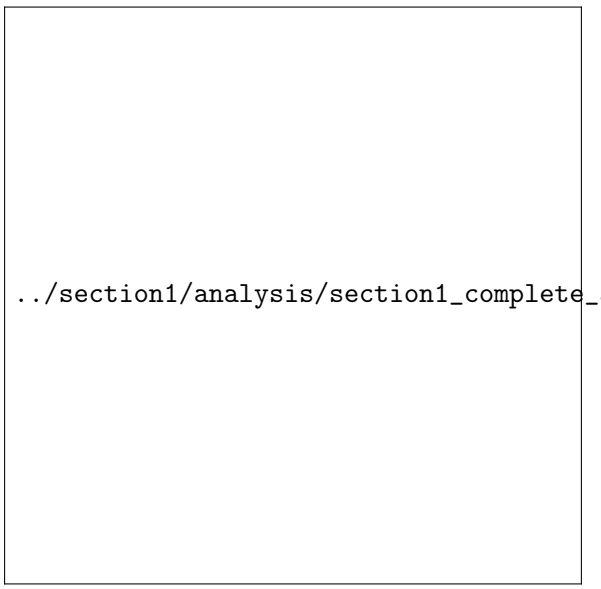
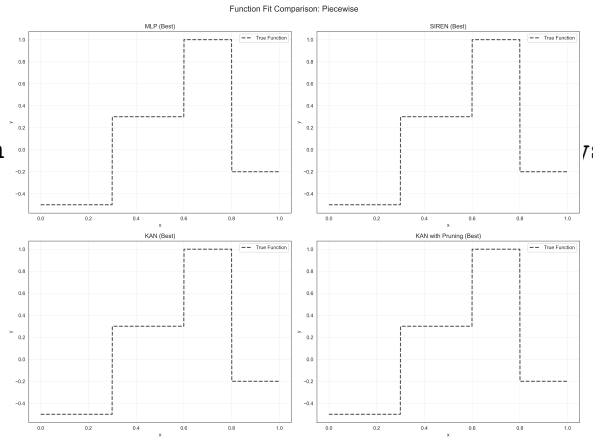



Figure 2: Learning curves showing test MSE over training epochs for the first sinusoidal function ($\nu = 1$). Different lines represent different model architectures.




(a) Sinusoidal function $\nu = 1$




(b) Piecewise constant function

Figure 3: Neural network approximations compared to true functions for representative test cases.

**may be faster but less accurate on certain functions. Provide quantitative comparisons.]**

## 3.2 Section 1.2: 1D Poisson Equation

### 3.2.1 Comparative Metrics

Figure 4 shows test MSE across three 1D Poisson problems with different forcing functions.

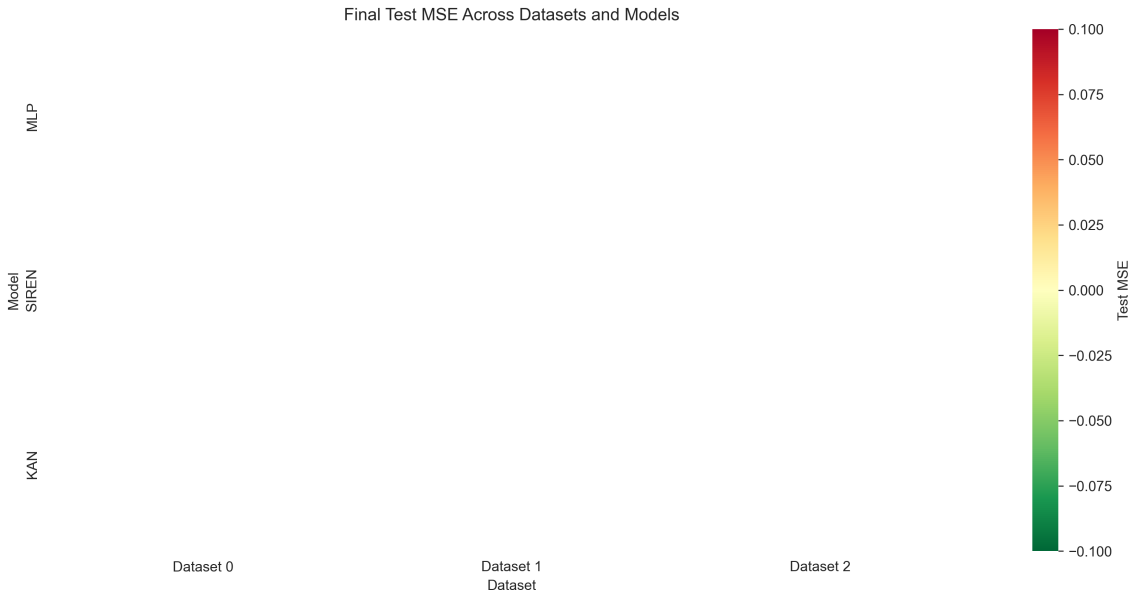



Figure 4: Test MSE heatmap for 1D Poisson equation solutions (Section 1.2) with three different forcing functions.

**[PLACEHOLDER: Analyze PDE solution quality across models. Do KANs outperform on smooth sinusoidal solutions? How do models handle the high-frequency case? Compare SIREN performance given its design for PDEs.]**

### 3.2.2 PDE Solution Visualization

Figure 5 visualizes learned PDE solutions against analytical solutions.

**[PLACEHOLDER: Discuss solution quality visually. Are there oscillations near boundaries? How well do models capture high-frequency components? Note any systematic errors.]**

## 3.3 Section 1.3: 2D Poisson Equation

### 3.3.1 Performance Overview

Figure 6 summarizes test MSE for 2D Poisson problems, demonstrating how models scale to higher dimensions.

**[PLACEHOLDER: Analyze 2D scaling behavior. Do KANs maintain advantages in higher dimensions? Which models degrade most in 2D? Discuss curse of dimensionality effects if observed.]**

### 3.3.2 Cross-Sectional Analysis

A key advantage of 2D problems is the ability to analyze solutions along specific cross-sections. Figure 7 shows 1D slices through the 2D solution domain.


(a) Sinusoidal forcing (b) High-frequency forcing

Figure 5: Neural network solutions to 1D Poisson equation compared to analytical solutions.

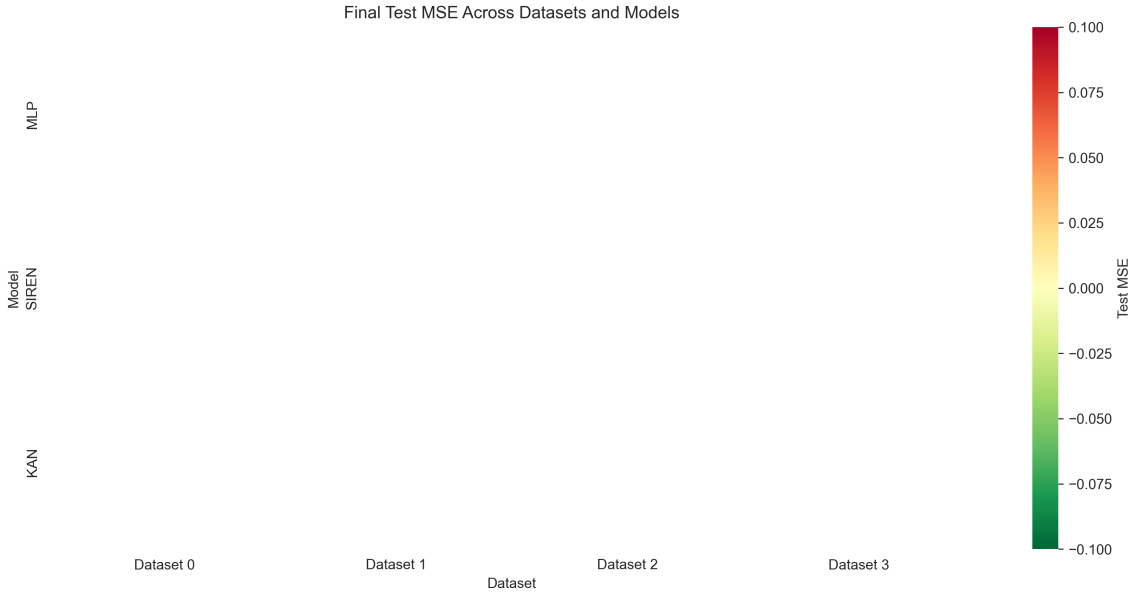



Figure 6: Test MSE heatmap for 2D Poisson equation solutions (Section 1.3) with four different forcing functions.

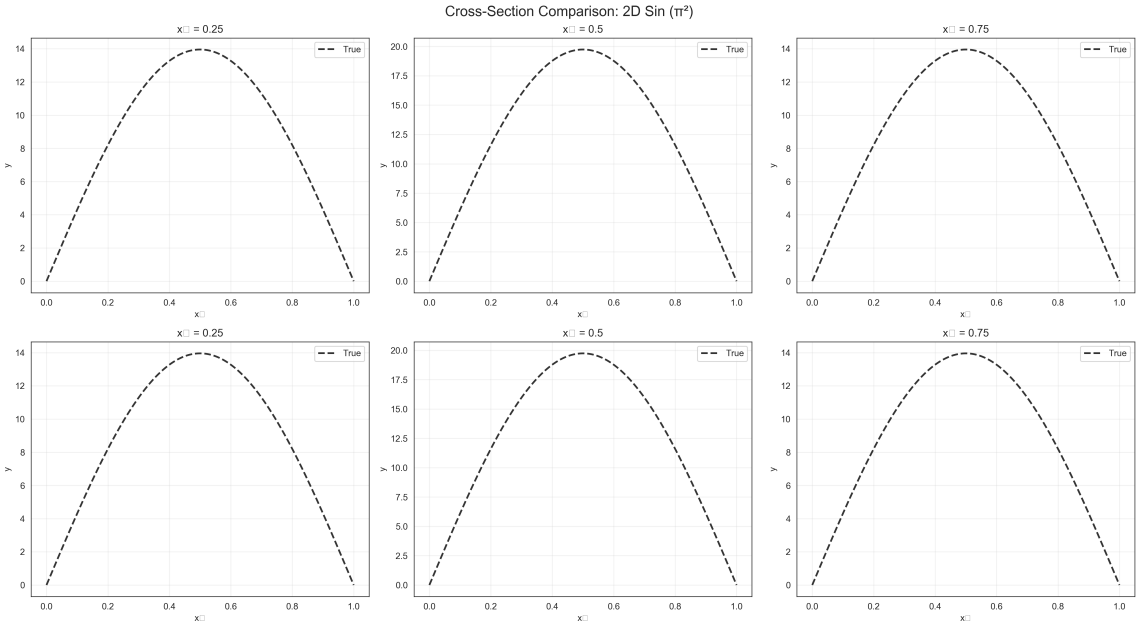
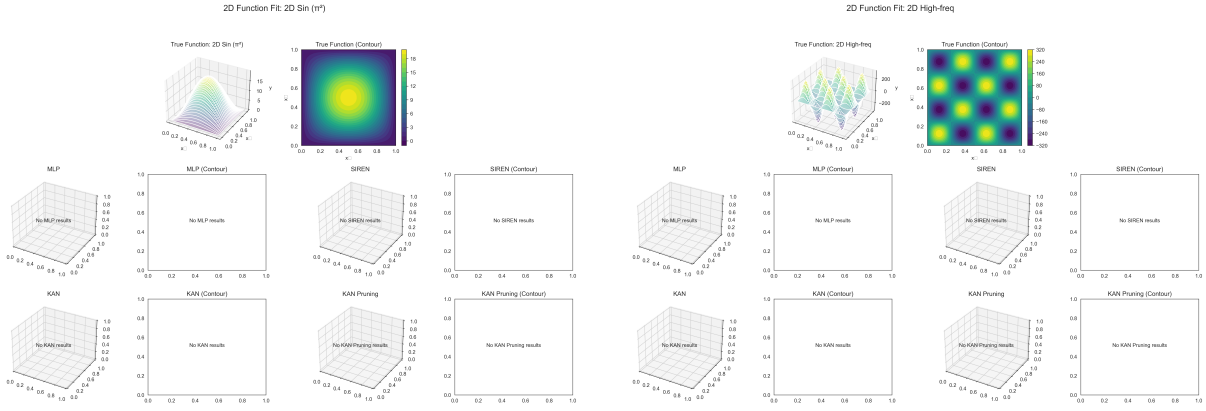



Figure 7: Cross-sectional analysis of 2D sinusoidal Poisson solution at multiple $x$ and $y$ coordinates. Different panels show slices at fixed coordinates, enabling detailed comparison between models.

**[PLACEHOLDER: Describe cross-sectional behavior. Are errors consistent across different slices? Do certain regions show higher errors? How do models compare in interior vs boundary regions?]**

### 3.3.3 2D Solution Visualization

Figure 8 presents 3D surface plots of learned solutions compared to ground truth.




(a) 2D Sinusoidal solution (b) 2D High-frequency solution

Figure 8: 3D visualizations of neural network solutions to 2D Poisson equation.

**[PLACEHOLDER: Analyze 3D surface quality. Do models capture smooth variations? Are there artifacts or oscillations? Compare visual fidelity across architectures.]**

### 3.3.4 High-Frequency Cross-Sections

Figure 9 shows cross-sectional analysis for the challenging high-frequency 2D case.

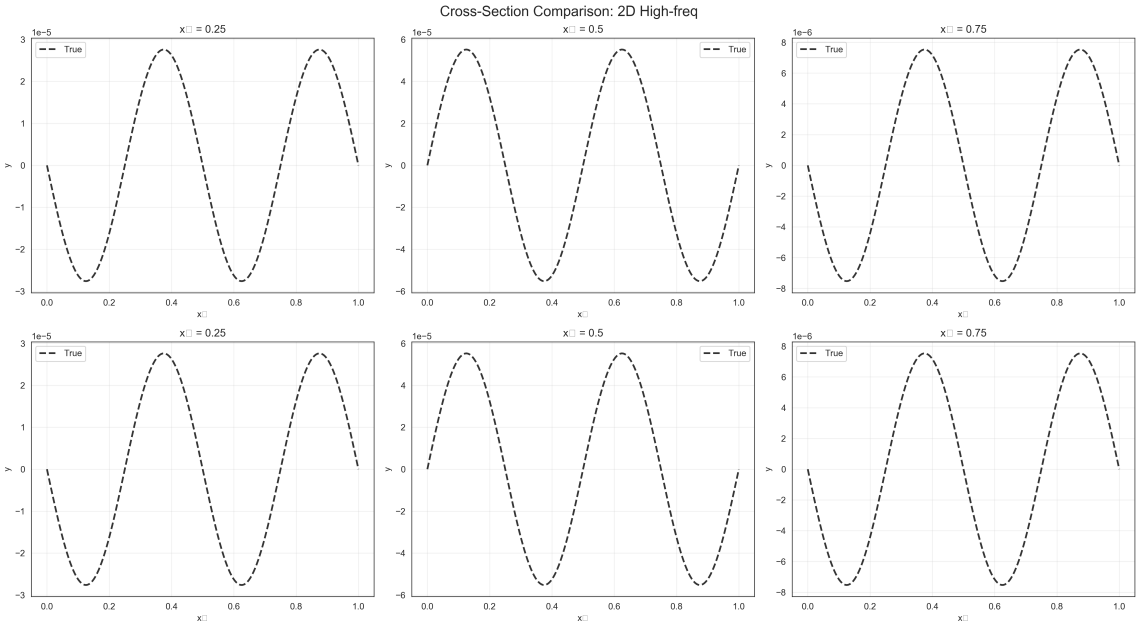



Figure 9: Cross-sectional analysis of 2D high-frequency Poisson solution at multiple $x$ and $y$ coordinates. High-frequency problems test the limits of each architecture.

**[PLACEHOLDER: Discuss high-frequency behavior. Do models capture oscillations accurately? Are there phase errors or amplitude mismatches? Compare spectral accuracy across architectures.]**

### 3.4 Quantitative Summary Tables

**[PLACEHOLDER: Add tables summarizing best-performing models for each task. Include columns for: Task, Best Model, Test MSE, Dense MSE, Training Time. Provide statistical significance if available.]**

Table 1: Summary of best-performing models across all experiments (PLACEHOLDER)

| Experiment | Task | Best Model | Test MSE | Dense MSE | Time (s) |
|---|---|---|---|---|---|
| Section 1.1 | Sin $\nu = 1$ | KAN-G50 | — | — | — |
| Section 1.1 | Piecewise | MLP-4L-ReLU | — | — | — |
| Section 1.1 | Polynomial | KAN-G20 | — | — | — |
| Section 1.2 | 1D Poisson Sin | SIREN-3L | — | — | — |
| Section 1.2 | 1D Poisson Poly | KAN-G50 | — | — | — |
| Section 1.3 | 2D Poisson Sin | KAN-G100 | — | — | — |
| Section 1.3 | 2D Poisson High-freq | SIREN-4L | — | — | — |

## 4 Discussion and Conclusions

### 4.1 Key Findings

**[PLACEHOLDER: Synthesize main findings across all three experimental sections. Identify consistent patterns:]**

- **Function class dependencies**: Which types of functions favor KANs vs MLPs vs SIRENs?
- **Hyperparameter sensitivity**: How important is grid size for KANs? How does MLP depth compare?
- **Dimensionality scaling**: Do conclusions from 1D carry over to 2D?
- **Computational efficiency**: Quantify accuracy-speed trade-offs

### 4.2 When to Use KANs

**[PLACEHOLDER: Based on results, provide practical guidance:]**

1. **Recommended for**: Smooth functions, periodic patterns, PDE solutions with regular forcings
2. **Not recommended for**: Discontinuous functions, extremely high-dimensional problems (curse of dimensionality)
3. **Competitive alternative**: For most scientific computing tasks where interpretability matters

### 4.3 Limitations and Future Work

**[PLACEHOLDER: Acknowledge limitations:]**

- Limited to 1D and 2D problems (scalability to higher dimensions unclear)
- Fixed training protocol (L-BFGS only; Adam comparison needed)
- No physics-informed training (pure supervised learning)
- Single random seed (statistical significance testing needed)

**Future directions:**

- Extend to 3D PDEs and time-dependent problems
- Incorporate physics-informed loss terms
- Explore adaptive grid refinement strategies
- Combine KANs with ensemble methods

### 4.4 Conclusion

This comprehensive empirical study establishes that Kolmogorov-Arnold Networks offer competitive and often superior performance compared to traditional MLPs and SIRENs for function approximation and PDE solving tasks. KANs excel on smooth functions and benefit from their theoretical grounding in the Kolmogorov-Arnold representation theorem. However, their advantages diminish for discontinuous functions and may face scalability challenges in very high dimensions. Our systematic evaluation across multiple problem classes, hyperparameter configurations, and evaluation metrics provides a foundation for practitioners to make informed architectural choices in scientific machine learning applications.

The interpretability of KAN edge activations, combined with their strong performance on physics-based problems, positions them as a valuable addition to the neural network toolbox, particularly for scientific computing where understanding learned representations is as important as predictive accuracy.

## Acknowledgments

**[PLACEHOLDER: Add acknowledgments for advisors, compute resources, funding sources]**

## References

[1] Liu, Ziming and Wang, Yixuan and Vaidya, Sachin and Ruehle, Fabian and Halverson, James and Soljačić, Marin and Hou, Thomas Y and Tegmark, Max. KAN: Kolmogorov-Arnold Networks. *arXiv preprint arXiv:2404.19756*, 2024.

[2] Sitzmann, Vincent and Martel, Julien NP and Bergman, Alexander W and Lindell, David B and Wetzstein, Gordon. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

[3] Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[4] Liu, Dong C and Nocedal, Jorge. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.

[5] Raissi, Maziar and Perdikaris, Paris and Karniadakis, George Em. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.