KAN:Kolmogorov–Arnold Networks

1st Haoyu Dong(hd2573)
Electrical Engineering
Columbia University
New York, USA

2nd Jinfan Xiang(jx2598)
Electrical Engineering
Columbia University
New York, USA

3rd Yunfei Ke(yk3108)
Data Science
Columbia University
New York, USA

***Abstract*—This report reproduces and analyzes the Kolmogorov–Arnold Networks (KAN), a novel neural network architecture inspired by the Kolmogorov–Arnold Representation Theorem (KART). KAN addresses limitations of traditional multilayer perceptrons (MLPs), such as inefficiency in high-dimensional tasks and lack of interpretability. By leveraging trainable univariate spline functions and dynamic grid refinement, KAN achieves efficient function approximation, adaptability, and symbolic interpretability. Experimental results validate KAN's equivalent accuracy and superior scalability compared to MLPs, particularly in complex compositional tasks. Challenges, including computational overhead and grid optimization, are explored alongside potential improvements to enhance the practicality of KAN for diverse applications in machine learning. We also studied the limitation of KAN in many applications and therefore analyzed the reasons behind them. We hope that this work can contribute to the study of KAN based models in the future.**

I. Introduction

In modern deep learning, the Multilayer Perceptron (MLP) serves as a cornerstone, whose foundational importance cannot be overstated. Since its proposal by Frank Rosenblatt in the mid-20th century [1], MLP has been extensively studied and has become the backbone of most modern deep learning applications. This has significantly contributed to the rapid advancements and thriving success of artificial intelligence (AI) in recent years. The key theoretical underpinning of MLP lies in the Universal Approximation Theorem (UAT). Proposed in seminal works such as [2], UAT demonstrates that an appropriately constructed MLP, with a sufficient number of hidden units and a non-linear activation function, can approximate any continuous function on a compact domain to arbitrary accuracy. This theoretical guarantee, coupled with the well-tested structure of MLP [3], has enabled modern deep learning models to excel across diverse tasks, showcasing remarkable learning capabilities.

However, alongside the notable benefits brought by UAT and the affine-activation structure of MLP, several intrinsic limitations arise:

- **Curse of Dimensionality (COD):** While UAT ensures MLP's universality, the exponential growth in the required number of parameters for high-dimensional problems often leads to inefficiencies [4].
- **Interpretability Issues:** The large number of parameters and hidden layers make MLP-based models inherently difficult to interpret. Therefore, post-analysis tools are needed to illustrate the results obtained by neural networks [5].
- **Training Challenges:** Deep MLP architectures are prone to issues such as gradient vanishing and exploding [6], leading to unstable training processes.
- **Task-Specific Limitations:** For certain specialized tasks, such as AI applications in scientific computing, these limitations can hinder performance or make learning infeasible [7]–[9].

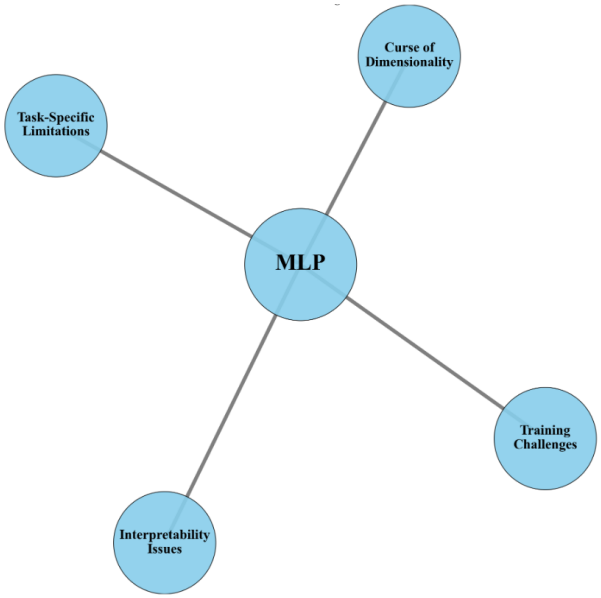

Fig. 1: Weakness of MLP

These challenges significantly impact the performance of MLP-based models in general tasks and further exacerbate difficulties in tasks requiring specialized solutions. Consequently, these drawbacks motivate researchers to explore alternative theoretical frameworks for neural network design.

One promising approach is the integration of the Kolmogorov–Arnold Representation Theorem (KART) as a replacement for UAT. Proposed in 1957 by Kolmogorov [10]–[12], KART states that any multivariable continuous function can be represented as a finite sum of univariate continuous functions. This theorem introduces potential advantages for deep learning models:

- **Dimensionality Reduction:** By transforming multivari-

able functions into univariate components, KART offers a way to mitigate the curse of dimensionality.
- **Efficient Representation:** KART-based networks allow for efficient function decomposition, potentially reducing the number of parameters required.
- **Foundational Flexibility:** Unlike affine-activation-based MLPs, KART enables the construction of architectures tailored to specific tasks.

Since KART's proposal, many researchers have attempted to incorporate it into neural network architectures, with works such as [13]–[20]. These efforts aimed to leverage KART's theoretical advantages in practical applications. However, several limitations hindered these models:

- **Non-Smooth Representations:** The inner functions in KART decompositions are often highly non-smooth, making them difficult to approximate with standard architectures.
- **Fixed Functional Forms:** Traditional KART-based networks often imposed rigid structures, reducing their adaptability to diverse tasks.
- **Implementation Complexity:** Determining and optimizing the internal functions of KART models involved cumbersome calculations, limiting their practical feasibility.
- **Generalization Challenges:** KART-based networks struggled with generalizing to unseen data, particularly in high-dimensional domains.

Building on the limitations of prior KART-based models, the Kolmogorov–Arnold Networks (KAN) proposed in [21], [22] introduce several improvements to make KART more practical in modern deep learning. Key innovations include:

- **Arbitrary Function Shapes:** KAN allows the internal functions to be represented with learnable parameters, such as splines, overcoming the rigidity of traditional forms.
- **Advanced Training Techniques:** KAN incorporates modern optimization strategies, such as backpropagation and regularization, to address training challenges.
- **Enhanced Architectures:** By designing architectures with adaptable widths and depths, KAN extends KART's applicability to a broader range of tasks.
- **Efficient Implementation:** KAN leverages grid extensions and other mechanisms to balance efficiency and performance.

These improvements significantly enhance the practicality and scalability of KART-based neural networks. The methodology and innovations behind KAN will be explored in detail in the following sections.

As a groundbreaking work published in 2024, Kolmogorov–Arnold Networks (KAN) seamlessly combine classical theoretical foundations with innovative advancements. Revisiting key concepts such as the Universal Approximation Theorem (UAT) and recently prominent scaling laws [23], KAN also explores state-of-the-art methods that are driving modern neural network research. This project serves as a bridge for deep learning learners, offering both exposure to cutting-edge research and practical experience in implementing advanced architectures. By reproducing KAN, we further develop our engineering skills through the programming of deep learning models using PyTorch and TensorFlow, the two leading frameworks in the field.

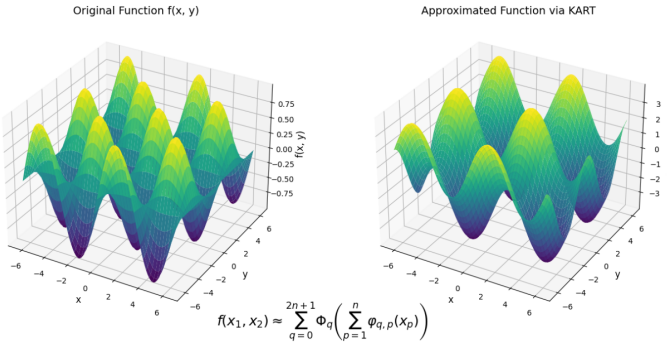

Fig. 2: Example of KART Approximation

In this report, we document our efforts to reproduce KAN and share insights drawn from the original work and related literature. Part 2 reviews KAN's methodology and results, while Part 3 outlines our approach to the project. Part 4 details the implementation process and key considerations in code structure, followed by a comparison of results in Part 5. Part 6 provides a discussion of the findings and observations. Finally, Part 7 concludes the project and explores potential future improvements.

II. Review on Original Work

A. *Materials & Methods*

*1) **Kolmogorov-Arnold Representation Theorem**:*
For Multi-Layer Perceptrons, one of the core theorems is the universal approximation theorem (UAT), while the core concept of the original paper is the Kolmogorov-Arnold representation theorem, which states that any multivariate continuous function $f(x_1, \ldots, x_n)$ can be expressed as a finite composition of univariate functions and additions:

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

where $\phi_{q,p}$ and Φ_q are univariate functions. A good example for visualized KART approximation made by us can be found in Fig. 2.One problem of this formula is that univariate functions may be non-smooth, which means they may be non-learnable. However, most functions are smooth and have sparse compositional structures, so we can have a smooth Kolmogorov-Arnold representation. This inspired the development of KANs, which generalize the theorem to deeper architectures.

*2) **KAN Architecture**:*

By the inspiration of MLPs, which stacks many layers to obtain a deeper model, the original authors define the KAN layer and stack layers to obtain a deeper model. KANs are neural networks where weights are replaced by trainable univariate spline functions. A KAN layer with n_{in} inputs and n_{out} outputs is defined as:

$$\Phi = \{\phi_{q,p}\}, \quad q = 1, \ldots, n_{out}, \quad p = 1, \ldots, n_{in}.$$

This formula works since we can set first layer $n_{in} = n, n_{out} = 2n+1$ and the second layer $n_{in} = 2n+1, n_{out} = 1$, which exactly represent the Kolmogorov-Arnold representation (1). Moreover, in matrix form, each layer can be represented as:

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

For a neuron at layer $l+1$ and position j, the value can be computed:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}), \quad j = 1, \ldots, n_{l+1}$$

In matrix form,

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

After defining layers, we can stack more KAN layers to obtain a deeper model. The network's output is a composition of these layers:

$$\text{KAN}(x) = (\Phi_L \circ \Phi_{L-1} \circ \cdots \circ \Phi_0)(x).$$

*3) **Approximation Theorem**:*

The original author provide a theorem that **Theorem (Approximation theory, KAT).** Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$. Suppose that a function $f(\mathbf{x})$ admits a representation:

$$f = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_1 \circ \Phi_0)(\mathbf{x}), \tag{2}$$

where each $\Phi_{l,i,j}$ is $(k+1)$-times continuously differentiable. Then there exists a constant C depending on f and its representation, such that we have the following approximation bound in terms of the grid size G: there exist k-th order B-spline functions $\Phi^G_{l,i,j}$ such that for any $0 \le m \le k$, we have the bound:

$$\|f - (\Phi^G_{L-1} \circ \Phi^G_{L-2} \circ \cdots \circ \Phi^G_1 \circ \Phi^G_0)(\mathbf{x})\|_{C^m} \le CG^{-k-1+m}. \tag{3}$$

The authors proved the correctness of the theorem. By this theory, KAN beats curse of dimensionality because the theory states that KANs can approximate the function well independent of dimension.

*4) **Techniques**:*

To improve accuracy, spline grids can be fine-grained during training. This process adjusts spline coefficients to match a finer grid, enhancing the network's resolution without full retraining. This is not feasible in traditional MLPs since we cannot increase the length and width of the model during the training. However, foe KANs, we can first train the model with fewer parameters and then add some parameters by extending the grid during the training.

To make KANs more interpretable, the original paper uses techniques include:

- **Sparsification**: The authors define the L1 norm and additional entropy regularization to favor sparsity.
- **Pruning**: The authors define two scores incoming scores and outgoing scores:

  $$I_{l,i} = \max_k \left(\|\phi_{l-1,i,k}\|_1\right), \quad O_{l,i} = \max_j \left(\|\phi_{l+1,j,i}\|_1\right).$$

  They will consider the node important if and only if both two scores are grater than the threshold. They will remove unimportant neurons based on activation scores.
- **Symbolification**: An interface (fix-symbolic(l, i, j, f)) is provided to set the activation function of a specific connection (l, i, j) to a symbolic function f . To address the issue of shifts and scalings, the symbolic function is adjusted to align with the model's internal transformations using the formula:

  $$y \approx c \cdot f(a \cdot x + b) + d,$$

  where x represents preactivations (inputs) and y represents postactivations (outputs).

B. Results

*1) **Accuracy**:*

The original authors rigorously examines the accuracy of Kolmogorov-Arnold Networks (KANs), establishing them as a more effective alternative to multi-layer perceptrons (MLPs) for representing complex functions. KANs and MLPs are compared on various tasks, focusing on accuracy, parameter efficiency, and scalability, using both theoretical analysis and empirical experiments.

To evaluate the function-fitting capability of KANs compared to MLPs, especially in tasks requiring high-dimensional or compositional function representations. The comparison is based on test RMSE and number of parameters.

In experimental design, the experiments involve five carefully chosen functions with known smooth Kolmogorov–Arnold (KA) representations:

1) $f(x) = J_0(20x)$: A univariate Bessel function.
2) $f(x, y) = \exp(\sin(\pi x) + y^2)$: A two-dimensional function with both sinusoidal and exponential components.
3) $f(x, y) = x \cdot y$: A simple multiplicative function.
4) $f(x_1, \ldots, x_{100}) = \exp\left(\frac{1}{100}\sum_{i=1}^{100} \sin^2\left(\frac{\pi x_i}{2}\right)\right)$: A high-dimensional compositional function.

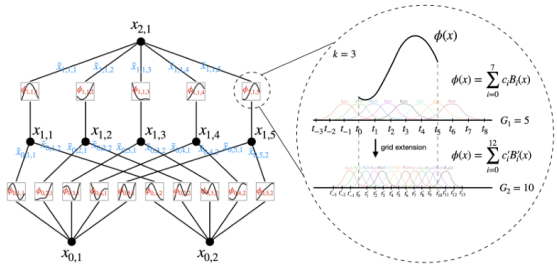

Fig. 3: Illustration of Spline Function from Original Work

5) A four-dimensional function requiring layered compositional structures:

$$f(x_1, x_2, x_3, x_4) =$$
$$\exp\left(\sin\left(\pi(x_1^2 + x_2^2)\right) + \sin\left(\pi(x_3^2 + x_4^2)\right)\right)$$

Through the whole experiment, KAN architectures varies in depths and widths and employ grid extension, incrementally refining the resolution of their spline-based activation functions during training. Baseline models include MLPs with depths ranging from 2 to 5 and varying node configurations per layer. Both KANs and MLPs are trained using the LBFGS optimizer for 1800 steps.

From the experiments, the result shows the superiority of KANs.

KANs achieve lower test RMSE than MLPs across all five tasks. Their scaling behavior matches the theoretical prediction ($\ell \propto N^{-4}$), where ℓ is the RMSE and N is the parameter count. KANs consistently outperform MLPs in terms of accuracy given the same number of parameters, particularly in high-dimensional tasks. For tasks requiring compositional structures, deeper KANs significantly outperform shallower ones, similar to the expressive power gains observed in deeper MLPs.

On the contrary, MLPs exhibit slower scaling laws, plateauing quickly in accuracy as the parameter count increases. Despite increased depth, MLPs struggle to approximate high-dimensional compositional structures efficiently.

Unlike MLPs, which require retraining larger models for better performance, KANs achieve finer accuracy by refining the spline grids dynamically during training, as Fig. 3 from original paper vividly illustrates. KANs' unique architecture, combining external and internal degrees of freedom, allows them to effectively learn compositional structures and univariate functions simultaneously. KANs achieve similar or better accuracy with fewer parameters than MLPs, reducing memory and computation requirements.

*2) **Interpretability**:*

KANs allow users to directly visualize and modify their internal components, making them inherently more interpretable than MLPs. The architecture simplifies the understanding of learned relationships, as KANs utilize learnable activation functions represented as splines, which can often map directly to symbolic expressions.

- **Interaction with KANs for Symbolic Regression** The original authors illustrates how users can interact with KANs to uncover symbolic relationships in data. Starting with a larger network, users can iteratively apply sparsification, pruning, and symbolification to derive interpretable symbolic expressions. An example is given where a regression task reveals the symbolic formula $f(x, y) = \exp(\sin(\pi x) + y^2)$, demonstrating KANs' effectiveness in symbolic regression tasks.
- **Applications in Scientific Discovery** KANs are particularly effective collaborators for scientists in discovering underlying mathematical or physical laws. Two examples are provided.
  In mathematics, KANs could be used for tasks like knot theory, where their structure simplifies complex relationships. KANs help identify relationships in the study of knots, uncovering functional dependencies between knot invariants. This assists mathematicians in understanding the geometric and topological properties of knots. By using symbolic regression techniques, KANs provide interpretable results that align with known theoretical properties or suggest new conjectures.
  In Physics, aiding in identifying key functional dependencies, KANs are applied to problems like Anderson localization. Anderson localization is a phenomenon in condensed matter physics where waves become localized due to disorder in the medium. KANs are applied to analyze datasets related to this phenomenon, revealing key symbolic relationships between system parameters and localization effects. Their symbolic outputs provide deeper insights into the interplay between variables, helping physicists validate existing theories or propose new mechanisms.
- **Comparison to Symbolic Regression** KANs are contrasted with traditional symbolic regression (SR) approaches by offering several advantages. They are robust due to continuous optimization using gradient descent, enabling them to handle noisy or complex data effectively. KANs can approximate non-symbolic functions using splines when no symbolic form exists, showcasing their flexibility and adaptability. Additionally, their transparency allows users to modify and debug the network easily, which is critical for interpretability and usability. In contrast, SR methods are often brittle and lack intermediate outputs, making them less interpretable and harder to debug, which limits their practical applicability in many real-world scenarios.

III. Methodology

A. *Project Objective*

The primary objective of this project is to reproduce and analyze the Kolmogorov–Arnold Networks (KAN), a groundbreaking neural network architecture proposed in 2024

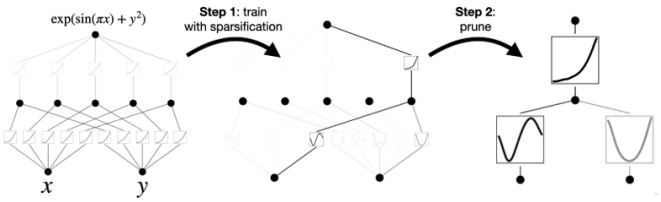

Fig. 4: KAN's Evolution

as we have mentioned above. KAN builds upon the Kolmogorov–Arnold Representation Theorem (KART) while addressing the limitations of traditional MLPs as well as earlier KART-based models. Specifically, our goals include:

*1) **Reproducing the Original Results**:*
The first objective is to faithfully implement the KAN architecture as described in the original paper, including its spline-based activation functions, grid extension strategy, and interpretability mechanisms. This involves constructing KAN layers where weights are replaced by learnable spline functions and ensuring the composition of these layers adheres to the theoretical underpinnings of KART. By conducting experiments on benchmark tasks, we aim to validate the authors' claims regarding KAN's superior accuracy, parameter efficiency, and scaling behavior when compared to traditional MLPs.

*2) **Performance Validation**:*
To comprehensively evaluate KAN's capabilities, our second objective is to assess its performance across tasks requiring compositional and high-dimensional function representations. Specifically, we will Compare the approximation accuracy and parameter efficiency of KAN and baseline MLPs using test RMSE on benchmark functions. We'll also examine KAN's robustness under different levels of noise and investigate the impact of grid extension on its ability to refine learned representations dynamically. This analysis will allow us to verify the theoretical claims regarding KAN's resilience to the curse of dimensionality and its adaptive learning capabilities.

*3) **Comparative Analysis**:*
A key component of this project is the comparative analysis of KAN with conventional neural network architectures, particularly MLPs. We aim to answer critical questions, such as: How does KAN scale with increasing parameters and task complexity compared to MLPs? What are the computational trade-offs associated with KAN's spline-based activations and grid extension mechanism? By analyzing convergence behavior, parameter efficiency, and computational costs, we aim to provide insights into the practical advantages and challenges of KAN over baseline models.

*4) **Technical Innovations Exploration**:*
The final objective is to explore and analyze the key technical innovations introduced in KAN, which include grid extensions, sparsification, pruning, and symbolic regression capabilities. Specifically, we aim to: Evaluate the impact of grid refinement on model accuracy and parameter growth. And we'll implement interpretability techniques such as L1-based sparsification and activation-based pruning to identify and visualize critical components of the network. And we'll also investigate KAN's potential for symbolic regression by extracting interpretable relationships from learned models. These explorations will enable us to better understand KAN's flexibility, interpretability, and scalability, offering valuable insights into its potential applications in scientific discovery and other fields.

*5) **Identification of KAN's Limitations**:*
While KAN introduces significant advancements in function approximation and interpretability, we aim to objectively analyze its potential limitations. These include increased computational overhead due to spline-based activations, implementation complexity arising from dynamic grid extensions, and uncertainties regarding scalability and generalization to large-scale tasks. By evaluating these aspects, we provide a balanced perspective on KAN's strengths and areas for improvement.
By addressing these objectives, this project aims to rigorously validate the theoretical and empirical contributions of KAN, while providing a detailed analysis of its strengths, limitations, and practical utility. The outcomes of this work will not only reproduce the original findings but also pave the way for further improvements and applications of KAN in modern neural network research.

*B. **Challengenges***

The reproduction of Kolmogorov–Arnold Networks (KAN) presented considerable challenges due to its large codebase, technical intricacies, and lack of standardized tools.

*1) **Codebase Size & Implications**:*
First, KAN's repository demonstrates a significant scale compared to similar works, reaching over 6000 lines of .py files alone, far surpassing well-known projects such as Residual Attention Networks (about 1030 lines) and SimCLR (about 2500 lines) as shown in Fig. 5. While code size does not inherently determine the complexity or quality of a project, it is an undeniable factor that correlates with reproduction difficulty. Larger codebases tend to introduce layers of interdependence and abstraction, requiring careful reverse-engineering to decipher their functional relationships. Additionally, KAN, as a novel 2024 work, lacks the encapsulated design and mature tooling typically seen in more established frameworks. This absence exacerbates the effort needed to dissect and reconstruct the core components of the architecture, making the reproduction process far more laborious than for earlier, better-documented models.

*2) **PyTorch to TensorFlow**:*
Second, a significant challenge arises from the need to port KAN from PyTorch to TensorFlow, which substantially increases the reproduction workload. PyTorch's dynamic computational graph and flexibility are core to the original implementation, yet TensorFlow, particularly when leveraging its Keras abstraction, often requires static graph definitions and explicit handling of intermediate operations. Such discrepancies necessitate a manual reimplementation of the KAN architecture. Most critically, the KAN layers and multi-layer

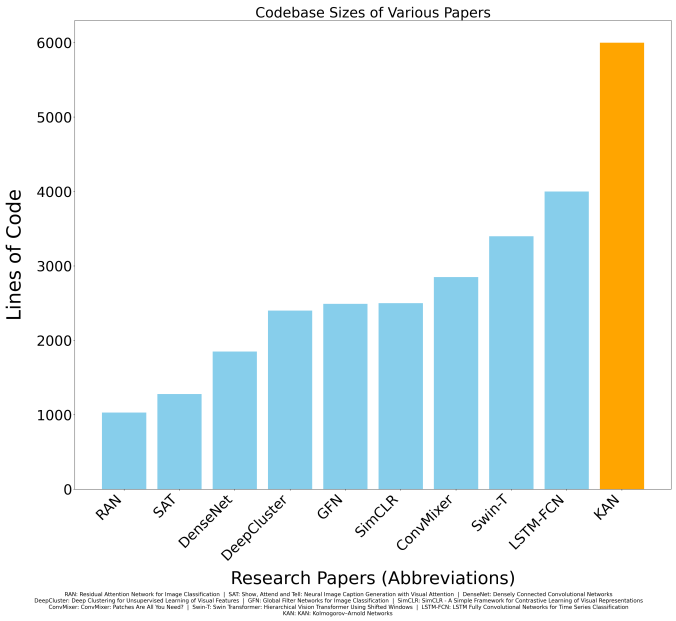

Fig. 5: Original Codebase Size for All Papers

KAN composition—which rely on custom spline-based activations—cannot take advantage of TensorFlow's existing high-level tools like functional APIs or pre-built layers. Unlike common architectures such as ResNets or LSTMs, where TensorFlow equivalents are readily available, KAN demands the reconstruction of its layers entirely from scratch. This lack of reusable templates places a greater coding burden on the implementation team, requiring precision to align with the original design while maintaining computational efficiency and correctness.

3) ***Grid Extension Mechanism****:*

Third, KAN's grid extension mechanism introduces additional technical challenges during implementation. Unlike conventional networks with static parameters, KAN dynamically refines its spline activations by expanding the resolution of the underlying grid during training. While this innovation is essential to KAN's success, it complicates reproduction due to the need for careful handling of parameter updates to ensure smooth transitions and numerical stability. At the same time, the grid extension process increases computational overhead, both in terms of memory usage and training time. Achieving an efficient implementation under limited hardware resources requires a delicate balance between grid refinement and overall model scalability.

4) ***Interpretability Techniques****:*

Finally, KAN's interpretability features, including sparsification, pruning, and symbolification, further elevate the complexity of the reproduction process. These techniques aim to enhance the transparency of the network by identifying and isolating critical neurons or connections. However, implementing these features demands robust scoring mechanisms to evaluate node importance while ensuring consistency with the original architecture. Aligning learned spline activations with symbolic representations introduces an additional layer of difficulty, as it requires preserving accuracy and interpretability without undermining the network's performance.

In summary, reproducing KAN is not merely a matter of following existing code but requires overcoming its extensive codebase, the complexities of PyTorch-to-TensorFlow migration, and the technical demands of implementing grid extension and interpretability mechanisms from scratch. These challenges, while significant, also provide an opportunity to deeply understand KAN's architecture and its novel contributions. In the following sections, we will demonstrate how we addressed these difficulties, ensuring fidelity to the original work while leveraging the flexibility of TensorFlow to adapt KAN's methodology. Through this process, we aim not only to faithfully reproduce the results but also to highlight the strengths of KAN's design and explore the potential advantages that arise from our implementation approach.

C. Key Ideas for Project

To implement the TensorFlow-based KAN model, we focus on achieving a minimal yet functional design while preserving the core principles of the original PyTorch implementation. Compared to PyTorch, which uses a dynamic computation graph and simplifies gradient handling, TensorFlow requires a more structured approach to ensure equivalent functionality. The model architecture combines numerical and symbolic components to replicate the original behavior. At its core, the numerical computations rely on grid-based spline interpolation to approximate activation functions. This mechanism enables the model to capture complex nonlinear relationships. In TensorFlow, we redefine this functionality using adaptive B-splines. Grid points are updated dynamically based on input data, ensuring the model can adapt effectively. Although the PyTorch implementation benefits from its flexibility, TensorFlow achieves the same dynamic behavior through operations like `tf.concat` and `tf.linalg.solve`, which efficiently handle grid updates.

In addition to numerical components, the symbolic pathway provides explicit symbolic mappings for greater interpretability. Unlike PyTorch, where symbolic transformations are easier to integrate due to its dynamic execution, TensorFlow requires careful coordination between numerical operations and algebraic manipulations. To address this, we adopt a modular design where both computations occur in parallel. The outputs from the numerical and symbolic components are merged seamlessly during the forward pass, ensuring consistency with the original implementation.

The optimization process introduces further differences. PyTorch offers a built-in LBFGS optimizer with support for advanced line search. However, TensorFlow requires a custom implementation to achieve similar results. In our version, the LBFGS algorithm is adapted using Wolfe condition-based line search and manual gradient updates. TensorFlow's automatic differentiation computes gradients efficiently, allowing the optimizer to handle the model's high-dimensional and non-convex optimization challenges.

To evaluate the TensorFlow-based implementation, we designed experiments focusing on a minimal reproduction of the original model's key features. These include spline-based activation functions, adaptive grid updates, and symbolic reasoning. Instead of replicating the entire PyTorch setup, we concentrated on essential functionalities to ensure clarity and performance. For comparison, we tested the TensorFlow KAN against baseline models such as Keras-based MLPs. The results show that our implementation successfully fits symbolic functions and models complex relationships while maintaining simplicity.

IV. Implementation

A. *Dataset*

As a novel architecture, Kolmogorov–Arnold Networks (KAN) have garnered attention for their remarkable interpretability. However, their applicability is by no means limited to a specific domain. On the contrary, as an emerging approach, KAN's potential should be thoroughly explored across diverse deep learning tasks to validate its versatility and robustness. In this reproduction study, we focus on KAN's **scientific applications**, where its exceptional functional approximation and interpretability capabilities can be most prominently demonstrated.

Unlike other works that focus solely on single-domain tasks such as image recognition, our objective is to expand the scope of KAN by applying it to multiple classical scientific machine learning tasks. To achieve this, we carefully prepared a collection of datasets that are not only representative of their respective tasks but also capable of producing insightful and demonstrative results using KAN. The datasets are introduced as follows:

*1) **Special Function Datasets**:*

To evaluate KAN's ability to approximate complex mathematical functions, we construct datasets based on well-known special functions. These include the **Bessel function** J_0, logarithmic combinations, and simple trigonometric functions. For instance, one of the target functions is defined as:

$$f(x) = \exp(J_0(20x_0) + x_1^2),$$

where J_0 is the zeroth-order Bessel function of the first kind. Another example involves logarithmic and sinusoidal components:

$$f(x) = \sin\left(2(\log(x_0) + \log(x_1))\right).$$

These datasets serve as benchmarks to assess the accuracy, parameter efficiency, and functional approximation capabilities of KAN.

*2) **Moon Dataset (Classification Task)**:*

The *Moon* dataset, a standard two-class classification problem, whose distribution can be found in Fig. 6, is employed to test KAN's ability to learn non-linear decision boundaries. By integrating custom functional mappings (e.g., Bessel functions and exponentials), we introduce additional complexity to the classification task. This dataset provides a straightforward yet insightful demonstration of KAN's performance in handling non-linear classification challenges.

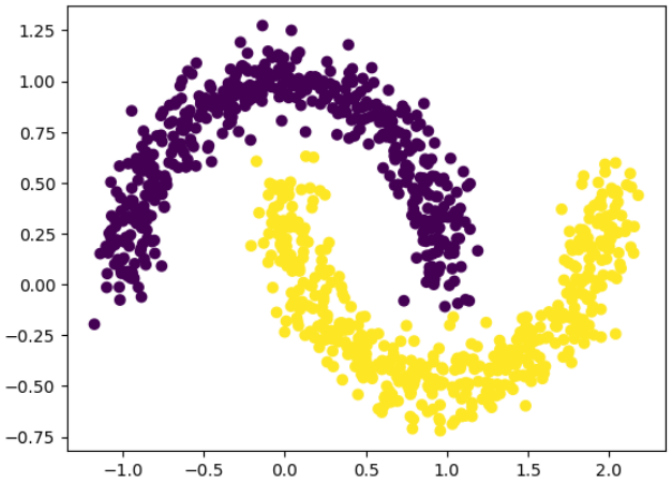

Fig. 6: Moon Dataset Distribution

*3) **PDE Solving Dataset**:*

For scientific applications, we employ a dataset derived from solving the second-order elliptic partial differential equation (Laplace's equation) over a square domain:

$$-\Delta u(x, y) = f(x, y), \quad \Omega = [-1, 1] \times [-1, 1],$$

where Δ is the Laplace operator:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}.$$

The exact solution is given by:

$$u(x, y) = \sin(\pi x)\sin(\pi y),$$

and the corresponding source term is:

$$f(x, y) = -2\pi^2 \sin(\pi x)\sin(\pi y).$$

This dataset evaluates KAN's ability to approximate solutions to PDEs, a fundamental problem in scientific computing, and highlights its strength in modeling continuous domains.

*4) **Gaussian Peaks Dataset (Continual Learning)**:*

To demonstrate KAN's potential in **continual learning** scenarios, we construct a Gaussian Peaks dataset featuring multiple peaks distributed across the input space. The peaks are sequentially introduced to the model, allowing us to evaluate KAN's ability to adapt to new data distributions while retaining knowledge of previously learned patterns. This dataset is particularly suited to test KAN's capacity for learning in dynamic and evolving tasks, a crucial aspect of modern machine learning models.

Based on the datasets described above, we aim to accomplish a series of **cross-domain scientific machine learning tasks** that include function approximation, classification, numerical PDE solving, and continual learning. This comprehensive

evaluation not only verifies KAN's outstanding performance but also demonstrates its versatility and applicability in various scientific domains.

B. Software Structure

The project develops a tensorflow 2.14.0 based architecture leveraging Kolmogorov-Arnold Networks (KAN) to approximate complex multivariate functions with high precision. This is implemented through TensorFlow tools and includes innovative layers and optimization techniques. Below is a detailed breakdown of the key components of the codebase.

*1) **KANLayer Implementation (KANLayer.py)**:*
The KANLayer class serves as the fundamental building block of the KAN architecture, introducing a grid-based approach for function approximation. Its implementation uniquely supports trainable coefficients, leveraging B-spline techniques (curve2coef) for smooth curve reconstruction, while offering optional sparse connectivity through a mask mechanism. The forward pass integrates pre-activation transformations and splines, ensuring efficient computation and adaptability across various input dimensions.

*2) **Optimization Framework (LBFGS.py)**:*
The LBFGS.py module provides an advanced line search optimizer tailored for high-dimensional KAN models. A standout feature is its robust cubic interpolation for determining optimal step sizes, integrated seamlessly with TensorFlow operations. The implementation emphasizes precision through the Strong Wolfe conditions, ensuring stability and convergence, making it ideal for handling the intricate parameter spaces inherent in KAN architectures.

*3) **Symbolic Representation (Symbolic_KANLayer.py)**:*
The Symbolic_KANLayer extends KAN's utility by incorporating symbolic computation via SymPy, enabling mathematical interpretability alongside numerical evaluations. Its key innovation lies in supporting singularity-robust activation functions, affine transformations of the form , and symbolic function manipulation, catering to tasks where explicit formulae are critical for verification or theoretical analysis.

*4) **Multi-Layer KAN Architecture (MultKAN.py)**:*
The MultKAN class builds upon individual KANLayer instances to create a deep, multi-layered KAN model. This implementation supports dynamic grid refinement for improved approximation accuracy and integrates multiplication nodes for complex feature interactions. A notable feature is the ability to checkpoint and restore model states, enabling iterative model enhancements and robustness during extensive training procedures. We also incorporated many tools for KAN analysis in MultKAN which played key roles in our experiments.

*5) **Spline-Based Utilities (spline.py)**:*
This module encapsulates the mathematical backbone of KAN layers, implementing core functionalities for B-spline evaluations and transformations. Functions like B_batch and coef2curve ensure seamless transitions between spline coefficients and curves, critical for the accurate representation of multivariate functions. Its recursive computation structure highlights flexibility and efficiency in basis evaluation.

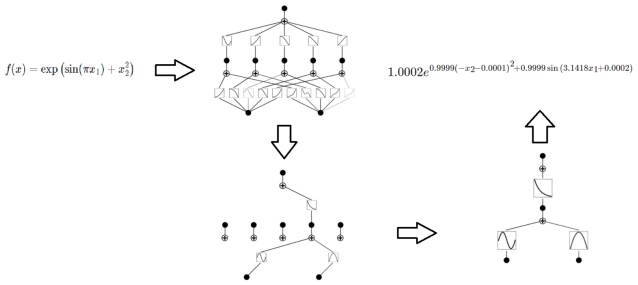

Fig. 7: Pipeline of Our KAN Approximation of $e^{\sin(\pi x_1)+x_2^2}$

*6) **Utility Functions (utils.py)**:*
The utils.py module offers versatile helper functions to streamline dataset preparation, input augmentation, and sparsity induction. Its create_dataset function generates symbolic data with customizable normalization and range settings, while augment_input enriches feature spaces with symbolic transformations. Sparse masks generated by sparse_mask align inputs and outputs, optimizing computational efficiency without sacrificing accuracy.

*7) **Integration and Experimentation**:*
The components described above are integrated to form a flexible framework for testing KAN architectures. The following experimentation steps are recommended:

- **Model Initialization:** Use MultKAN with appropriate hyperparameters to define the architecture.
- **Training:** Train the model using LBFGS or standard optimizers with spectral or symbolic objectives.
- **Evaluation:** Evaluate performance using custom datasets or benchmarks, leveraging symbolic capabilities for analysis.
- **Visualization:** Use matplotlib and symbolic outputs for visual validation of results.

V. Results

A. Project Results

*1) **KAN Approximation on Math Formula**:* In the first experiment, we explored the foundational capabilities of the KAN in approximating data distribution functions with minimal complexity. The results demonstrates that even a minimal-width KAN architecture (2, 5, 1) is capable of flexibly approximating the target function

$$f(x) = e^{\sin(\pi x_1)+x_2^2}$$

after limited epoch os basic training. Remarkably, the network autonomously identified and effectively "pruned" unnecessary neurons, focusing only on the key components needed for accurate approximation. This dynamic adaptability is evident from the training results, where the model aligns closely with the ground truth without overfitting or underutilizing its structure. Further enhancements to accuracy were achieved through iterative pruning and refinement techniques. The additional

refinement stages demonstrated the network's ability to push its approximation capabilities to higher precision, underlining its superiority over traditional MLPs in terms of efficiency and flexibility. This success in faithfully reproducing KART sets the stage for establishing KAN as a viable replacement for conventional MLP architectures in complex approximation tasks.

Who Pipeline of results from this experiment can be found in Fig. 7, The results provide a strong foundation for further exploration of KAN's theoretical limits. Specifically, the potential for infinite approximation via grid refinement, as discussed in the next experiment, builds on this demonstration of adaptability and precision.

*2) **Impact of Grid Refinement**:* The second experiment delves into the relationship between grid size and KAN's approximation performance. Incrementally increasing the grid resolution revealed a clear pattern: each increase in grid size caused the model's loss to rapidly decline from previously plateaued values, achieving a new level of accuracy before plateauing again. This behavior provides strong empirical evidence supporting KART's assertion that KAN can approximate any function to arbitrary precision, given a sufficiently large grid.

However, while theoretically compelling, this capability comes with practical engineering considerations. Our analysis identified three critical trade-offs that limit the utility of infinite grid refinement in real-world applications–and in engineering especially:

- **Diminishing Returns**
  Although increasing grid size improves approximation accuracy, the incremental improvement becomes progressively smaller as grid resolution grows. This is evident in the experimental data of Fig. 8, where the rate of loss reduction slows significantly after each grid increment.
- **Increased Computational Cost**
  Higher grid resolutions substantially increase the total number of trainable parameters, thereby requiring more computational resources. This imposes practical limits on grid refinement, particularly in resource-constrained environments.
- **Generalization Limits**
  While the fit to the training data improves with grid size, the generalization loss reaches a plateau at a certain grid level as shown in Fig. 8, and, in some cases, deteriorates as grid size becomes excessively large. This phenomenon underscores the risk of overfitting when prioritizing precision over generalization.

Based on these observations, we conclude that while grid refinement is a powerful tool for improving approximation accuracy, it must be balanced against computational efficiency and generalization performance. These considerations provide valuable guidelines for determining optimal grid sizes in practical implementations.

*3) **KAN Depth Analysis**:* In this experiment, we analyze the effect of network depth on the Kolmogorov-Arnold Network's

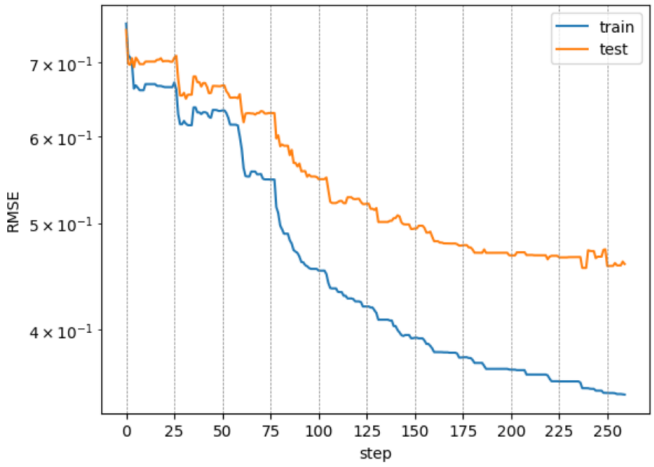

Fig. 8: How Grid changes Performance
(Each 25 steps for a larger Gird)

(KAN) ability to approximate the complex function:

$$f(x) = \exp\left(\frac{\sin\left(\pi r_1^2\right) + \sin\left(\pi r_2^2\right)}{2}\right),$$

$$\text{where} \quad r_1^2 = x_1^2 + x_2^2 \text{ and } r_2^2 = x_3^2 + x_4^2$$

The results reveal that the depth of the network plays a critical role in determining KAN's approximation capacity for any certain math formula (distribution of dataset). Specifically, the three-layer KAN (width=[4,2,1,1]) outperforms the two-layer KAN (width=[4,9,1]) by a significant margin, despite the latter having more neurons and parameters. This finding underlines a fundamental property of KAN: depth enhances the network's ability to model compositional structures, even in cases where a shallower network is substantially wider.

The superior performance of the three-layer KAN in Fig. 9 highlights the expressive power of deeper architectures. By structuring its layers hierarchically, the three-layer KAN effectively partitions and captures the compositional components of the target function compared with the two-layer KAN. This outcome demonstrates that merely increasing the number of parameters or neurons in a shallow network cannot substitute for the hierarchical representational capabilities afforded by deeper architectures. This property is consistent with observations in traditional Multi-Layer Perceptrons (MLPs), suggesting that KAN shares some key characteristics with MLPs. As a result, the findings provide theoretical support for KAN's potential to replicate many functionalities typically associated with MLPs, broadening the applicability of KAN in machine learning tasks. We conclude the expressive result of this experiment on depth in Fig. 10. Beyond depth's role in enhancing expressiveness, the experiment also emphasizes the importance of selecting an appropriate network structure for specific functions. The success of the three-layer KAN underscores the significance of architectural design in achieving optimal performance. In this case, the function's compositional complexity aligns well with the three-layer network's

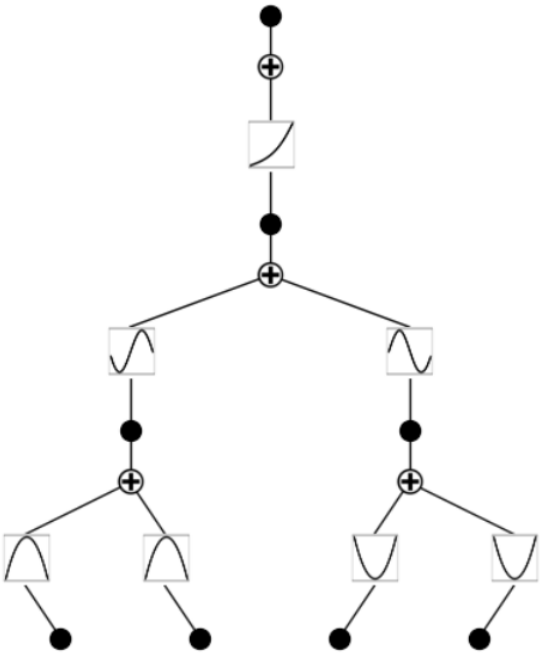

Fig. 9: width=[4,2,1,1]) Performance

hierarchical capabilities, enabling it to achieve significantly better results than the two-layer counterpart. Conversely, the two-layer KAN's subpar performance highlights the risks of inadequate depth, even when computational resources are heavily invested in wider layers.

This finding has important implications for engineering and practical implementations of KAN as when designing KAN architectures, a one-size-fits-all approach is insufficient—optimal performance depends on selecting the right depth for the task at hand. For problems where the data distribution or function structure is known, prior knowledge can guide the design of an architecture that aligns with the function's compositional properties. This experiment not only validates the theoretical advantages of deeper KAN architectures but also reinforces the practical need for careful depth selection in model design. These findings pave the way for future research into leveraging KAN's depth-based properties in specialized applications, particularly those where a priori knowledge about data distribution can inform model design for optimal performance.

*4) **Capability of Classification**:* In this experiment, we applied KAN to solve a binary classification task using Moon Dataset. We used a shallow KAN architecture with 2 layers (width: [2, 2]) and a grid size $k = 3$, which controls the complexity of the activation functions. For optimization, we chose the LBFGS optimizer with a learning rate of 0.001 and performed 5 optimization steps. The train accuracy and test accuracy were used to evaluate the model's performance.

The experiment achieved promising results as is mentioned in the original paper. The shallow KAN obtained a train

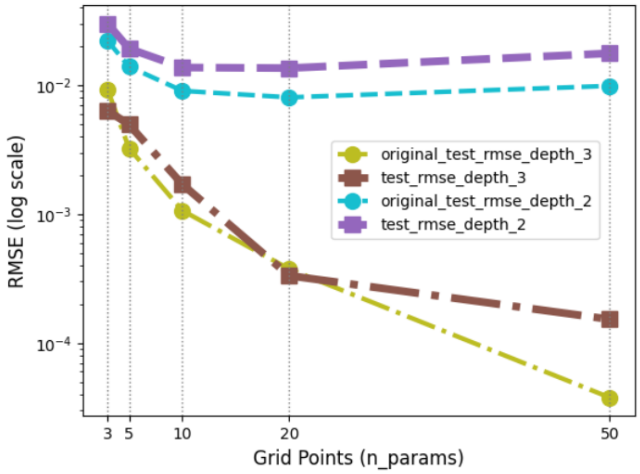

Fig. 10: Depth & Loss for Both Models and Both Implementations

accuracy of 0.879 and a test accuracy of 0.888, showing that it can accurately learn the non-linear decision boundaries of the dataset. Additionally, the KAN model generated interpretable symbolic formulas that approximated the decision boundary.

TABLE I: Decision Boundaries and Accuracy of PyTorch-KAN

Boundary 1	-15.0316x1 + 177.9349x2 – 63.0716
Boundary 2	60.4718x1 – 156.0295x2 + 16.9
Training Accuracy	0.887
Testing Accuracy	0.881

TABLE II: Decision Boundaries and Accuracy of TensorFlow-KAN

Boundary 1	-215.5785x1 + 758.3895x2 – 103.973
Boundary 2	214.1306x1 - 755.6062x2 + 98.0888
Training Accuracy	0.879
Testing Accuracy	0.888

This result highlights KAN's ability to provide both high accuracy and symbolic interpretability. Unlike traditional neural networks, which produce black-box outputs, KAN can reveal the underlying mathematical structure of the data, making it more transparent and insightful for analysis.

At the same time, it is important to recognize the limitations of a shallow architecture in classification tasks. While the two-layer KAN achieved strong performance on this relatively simple two-moon dataset, deeper architectures may be necessary for more complex classification problems, such as those with highly non-linear or hierarchical decision boundaries. Previous experiments on KAN depth analysis have demonstrated that a three-layer KAN can significantly outperform a wider two-layer version. This improvement occurs because deeper networks are better at learning and representing compositional decision boundaries, even when the shallower networks have a larger number of parameters. This finding is consistent with results observed in traditional MLPs, where increasing network depth often leads to better performance by enabling the model to capture higher-level feature relationships in the data.

From this experiment, it is shown that KAN has the capability of handling tasks in different scenarios. Nevertheless, after we further compare different settings of our model, analysis points out some factors which should be considered in classification applications:

- **Network Depth**
  Shallow networks, like the two-layer KAN used here, can perform well on relatively simple tasks such as the two-moon dataset. However, for more complex or hierarchical data, deeper networks are needed to capture compositional structures effectively. Deeper networks can achieve better expressiveness, whereas wider shallow networks may reach their limits on such tasks.
- **Grid Size and Complexity of Activation Functions**
  Increasing the grid size and depth of the KAN improves accuracy and allows the model to better approximate complex decision boundaries. However, this comes at the cost of higher computational complexity and longer training times, which need to be carefully managed depending on the application.
- **Noise and Data Variability**
  The presence of noise in the data can affect the network's performance. While KAN performs well on noisy datasets (as shown in the two-moon experiment), careful regularization or fine-tuning may be needed for more challenging data distributions.

*5) **Performance on Special Functions**:* To see if multivariate special functions could be written in KA representations , we evaluate the KAN model for its ability to approximate the target function $f(x, y) = \exp\left(J_0(20x) + y^2\right)$, involving the zero-order Bessel function $J_0(20x)$. The ground truth data was generated using 'scipy.special.j0', while a Taylor series approximation of J_0 was used during training. Symbolic regression analyzed the recovered structure, and a custom J_0 approximation was added to the symbolic library to improve interpretability. Performance was assessed through test loss and symbolic recovery.

However, when compared to the result shown in the original paper, the symbolic regression using our model did not explicitly recover the Bessel structure, and the fitting loss remained relatively high. To make it clear, we analyzes the reasons behind the observed limitations and identifies possible areas for improvement.

TABLE III: Top 5 Suggested Symbolic Function using PyTorch-KAN

RANK	**Function**	**Fitting Loss**
0	J0	-1.166097
1	0	0.000003
2	x	0.799540
3	cos	1.346963
4	sin	1.346963

TABLE IV: Top 5 Suggested Symbolic Function using TensorFlow-KAN

RANK	**Function**	**Fitting Loss**
0	sin	-0.874493
1	x^2	-0.874220
2	cos	-0.873947
3	0	0.000003
4	gaussian	0.089003

- **Coefficient Range Issue**
  The coefficient 20 in $J_0(20x)$ falls outside the default symbolic search range of $(-10, 10)$. KAN's symbolic regression process is constrained to this range, which makes it difficult for the model to identify the correct structure. While expanding the search range to $(-80, 80)$ improved the fitting, the exact J_0 structure still did not emerge. This highlights the importance of adjusting search boundaries when working with functions involving large coefficients.
- **Approximation Precision**
  The ground truth data for J_0 was generated using the high-precision implementation from 'scipy.special.j0', whereas training relied on a Taylor series approximation of J_0. The approximation introduces slight inaccuracies, particularly for larger values like $20x$, which deviate from the ground truth. These inconsistencies may have affected KAN's ability to recognize the correct symbolic representation.
- **Data Inconsistency**
  A mismatch exists between the target function's ground truth data and the approximated version used during train-

ing. While the symbolic regression process attempts to minimize error based on the training data, discrepancies between the approximated and true values of J_0 introduce noise, reducing the accuracy of symbolic recovery.

- **Model Bias Toward Simplicity**
  KAN exhibits a tendency to prioritize simpler symbolic terms, such as $\sin(x)$, $\cos(x)$, and polynomials like x^2. Even after expanding the search range, the model favored low-complexity solutions that minimize the total loss efficiently. This behavior reflects KAN's bias toward lower complexity terms, which helps reduce overfitting but may limit its ability to represent highly specialized functions like J_0.

The observed limitations also offer valuable insights for improving KAN's performance on similar tasks. For target functions with large coefficients, it becomes necessary to expand the symbolic search range to identify higher-scale components effectively. Automating this process could make the model more robust and adaptable. At the same time, using a more accurate J_0representation—such as higher-order Taylor expansions or more precise implementations—could help reduce inconsistencies and improve symbolic recovery.

*6) **Solving Partial Differential Equation (PDE)**:* To introduce the application of solving PDEs, we use the KAN model to solve a 2D Poisson equation:

$$\nabla^2 f(x, y) = -2\pi^2 \sin(\pi x) \sin(\pi y),$$

with boundary conditions $f(-1, y) = f(1, y) = f(x, -1) = f(x, 1) = 0$ and ground-truth solution $f(x, y) = \sin(\pi x) \sin(\pi y)$. The KAN architecture had a width of $[2, 2, 1]$ and polynomial order $k = 3$. The first layer activations were set as linear functions, and the second layer used sine functions. Training utilized the LBFGS optimizer with a loss combining interior PDE constraints and boundary conditions, alongside dynamic grid refinement to enhance resolution during training.

The loss function combines the PDE loss, obtained via the Laplacian (second derivative) of the KAN output using automatic differentiation, and the boundary loss, which minimizes discrepancies at boundary points. Interior points were generated using a mesh grid, while boundary points were explicitly defined. The LBFGS optimizer was employed for efficient convergence, and dynamic grid updates refined the grid resolution during training to improve accuracy. The results showed that KAN approximated the solution with a PDE loss of 7.47×10^{-2} and an L2 loss stabilizing at 4.58×10^{-3}.

From the comparison between our experimental results and those reported in the original paper, several factors could explain the differences in PDE loss and training time. One possible reason lies in the optimizer settings. While the original paper employs the LBFGS optimizer with specific line search configurations, subtle differences in how step sizes or gradients are managed in the TensorFlow implementation may lead to slower convergence and higher loss.

Another contributing factor could be the grid adaptivity mechanism. In the TensorFlow version, dynamic grid updates might not be as efficient as in the PyTorch implementation. Variations in numerical precision or the refinement strategy could cause the grid to misalign with the data distribution, increasing approximation errors in the spline-based activations.

The batch Jacobian computation may also introduce overhead in TensorFlow. Unlike PyTorch's more flexible dynamic graph, TensorFlow's static computation graph could lead to additional delays during each optimization step, which impacts training efficiency.

Finally, differences in hyperparameter settings, such as the number of training steps, learning rate, or the weighting factor `alpha` for loss balancing, may also contribute to the observed results. Fine-tuning these parameters, particularly the trade-off between PDE loss and boundary condition loss, could be key to improving performance.

TABLE V: Comparison of the Two Models in Solving PDE

Model	**Loss**	**L2 Loss**	**Training Speed**
PyTorch-KAN	$2.83e-02$	$3.78e-03$	$1.20s/it$
TensorFlow-KAN	$7.47e-02$	$4.58e-03$	$16.85s/it$

*7) **PDE Loss Accuracy Analysis** :* We aim to solve the same second-order elliptic partial differential equation (PDE) to test its accuracy. The loss function for our model combines the PDE residual loss $\mathcal{L}_{\text{PDE}}$ and the boundary condition loss $\mathcal{L}_{\text{BC}}$:

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{BC}},$$

where:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_i} \sum_{i=1}^{N_i} \left(\Delta u_\theta(x_i) - f(x_i) \right)^2,$$

$$\mathcal{L}_{\text{BC}} = \frac{1}{N_b} \sum_{i=1}^{N_b} \left(u_\theta(x_b) - u_{\text{true}}(x_b) \right)^2.$$

The model is optimized using the LBFGS optimizer over 50 steps. Internal grid points x_i are sampled uniformly across the domain, while boundary grid points x_b are extracted along the domain boundary.

During training:

- PDE residual loss $\mathcal{L}_{\text{PDE}}$ is minimized to enforce the equation constraints inside the domain.
- Boundary condition loss $\mathcal{L}_{\text{BC}}$ ensures the model satisfies the boundary conditions.
- The overall L_2-squared loss measures the total error between the model's prediction and the analytical solution.

The loss curves during training are shown below:

From the figure, we observe:

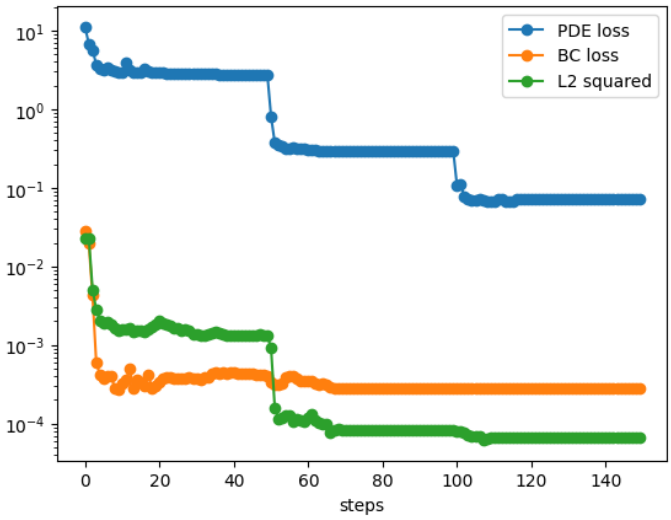

Fig. 11: Loss curves during training: PDE loss, boundary condition loss, and L_2 squared error.

- PDE Loss (blue curve): It decreases gradually, with step-wise drops corresponding to adaptive grid refinement. This can indicate improved approximation of PDE.
- Boundary Condition Loss (orange curve): It drops quickly and and become stable, which showed that the model effectively satisfies the boundary conditions.
- L2 Squared Loss (green curve): It also decreases gradually with step-wise drops. This shows the overall convergence towards the true solutio is quite good.

The results demonstrate that the KAN model successfully approximates the PDE solution. Grid refinement significantly improves the model's accuracy, as seen from the step-wise drops in PDE loss. The final losses converge to small values, which means that the model learns the solution effectively.

*8) **Continual Learning**:* This experiment uses a Kernel-based Adaptive Network (KAN) to approximate a function composed of multiple Gaussian peaks. The function is defined as the sum of Gaussian peaks:

$$y(x) = \sum_{i=1}^{n_{\text{peak}}} \exp(-300 \cdot (x - c_i)^2), \quad x \in [-1, 1],$$

where c_i represents the centers of the Gaussian peaks, calculated as:

$$c_i = \frac{2}{n_{\text{peak}}} \left(i - \frac{n_{\text{peak}}}{2} + 0.5 \right).$$

A continuous grid x_{grid} is defined with $n_{\text{sample}} = 500$ points uniformly spaced over the range $[-1, 1]$. For each peak, a subset of $n_{\text{num per peak}} = 100$ points is sampled locally around its center c_i. The sampled data points are used for training.
The KAN model is initialized with the following parameters: input dimension 1, output dimension 1, grid resolution 200, spline order $k = 3$, and noise scale 0.1. Both scaling parameters for the spline and base functions are set as non-trainable (`sp_trainable=False` and `sb_trainable=False`), and the base function is `zero`.
During training, the model sequentially learns data corresponding to each Gaussian peak. For each peak, a subset of sampled points x_{train} and corresponding labels y_{train} are extracted. The model is then trained for 50 steps using the LBFGS optimizer with a learning rate of 10^{-5}. The results show that the train and test losses converge to very small values (on the order of 10^{-6}), indicating quite good approximation accuracy.
This kind of sequential training approach enables the model to learn the localized features of each Gaussian peak without forgetting previously learned peaks. This demonstrates the effectiveness of continual learning with KAN. The grid resolution of 200 points allows the model to capture fine details of the Gaussian peaks. This experiment highlights the capability of KAN in continual learning.

*9) **Singularity Case**:* This document analyzes the ability of KAN to approximate functions containing singularities and their corresponding symbolic expressions. Two functions are studied to demonstrate the model's strengths and limitations.
Example 1: Logarithmic and Sine Singularities
In the first example, the target function contains logarithmic and sine components:

$$f(x, y) = \sin(\log(x) + \log(y)).$$

This function introduces singularities at $x = 0$ and $y = 0$, where the logarithmic terms become undefined. To address this issue, the dataset is constructed in the range $x, y \in [0.2, 5]$, avoiding the problematic singularity points.
The KAN model is initialized with:

- Input dimension: 2,
- Output dimension: 1,
- Grid size: 5,
- Spline order: $k = 3$.

The model is trained using the LBFGS optimizer with a learning rate of 10^{-4} for 20 steps. During training, the symbolic components are explicitly fixed to match the logarithmic and sine structure. After training, the learned symbolic formula is extracted and rounded for simplicity. The resulting expression is: $0.003 - 0.938 \sin(1.833 \log(x_1) - 0.397) + 1.903 \log(9.987 x_2 + 0.112) - 10.942$. This formula does approximate the original target function, which means in this case, singularity does not seem to be a problem.
Example 2: Square Root and Multiplicative Singularities
The second example considers a function with square root and multiplicative components:

$$f(x, y) = \sqrt{x^2 + y^2} \cdot x.$$

This function introduces a singularity at $y = 0$, where the square root term may cause numerical instability. The dataset is constructed in the range $x, y \in [-1, 1]$, covering the entire domain.
The KAN model is initialized with the same configuration as the first example. Despite these fixes, the extracted symbolic

formula does not fully match the original function. The learned expression is:

$$1.01\sqrt{x_1^2 + x_2^2} - 0.01.$$

While the square root term is captured, the explicit multiplicative term x is not accurately represented. This discrepancy highlights a limitation in the current symbolic fixing process.

- In the first example, the KAN model successfully approximates the target function by fixing the symbolic components (`log` and `sin`).
- In the second example, while the model captures the square root term, it fails to accurately represent the multiplicative component x.

The KAN model shows some capacity of dealing with singularity case. However, some functions such as functions with complex multiplicative terms, it can not output the correct function.

10) ***Comparison Among models****:* In this experiment of comparison, three architectures of deep learning models are used to further study the intrinsic magic of KAN and its weakness.

- KAN (Kernel-based Adaptive Network): A model consisting of adaptive grid-based kernel layers to approximate complex functions with higher interpretability and efficiency. It leverages spline-based interpolation techniques to handle nonlinear relationships.
- KANLayer with Keras: A simplified implementation of KAN layers integrated into the TensorFlow Keras framework. The model uses a Sequential architecture with KANLayer components, for instance:

  ```
  model = tf.keras.models.Sequential(
  KANLayer(in_dim=2, out_dim=5),
  KANLayer(in_dim=5, out_dim=1)).
  ```

  Here, each KANLayer replaces the traditional dense layers with spline-based functional approximations.
- MLP (Multi-Layer Perceptron): A fully connected neural network serving as the baseline model. The architecture typically includes dense layers with nonlinear activation functions (e.g., ReLU) for function approximation.

The experiments were designed to evaluate the ability of these models to approximate a given target function $f(x)$. After trials of different functions, basically, we found that our KAN model cannot outperform MLP models, with training loss ranging from 10^{-1} to 10^{-5}. Moreover, for KANLayer with keras, the model has good results (loss $\approx 10^{-5}$) in some function, for instance, $f(x) = e^{\sin(\pi x_1)+x_2^2}$, but it has some poor results for other functions (loss $\approx 10^{-1}$), such as $p = \sqrt{1 + a^2 - 2a\cos(\theta_1 - \theta_2)}$. MLP always has loss around 10^{-4}. This shows that MLP still a better model to fit in our opinions, and the better performance of KAN in the original work simply results from proper dataset and shallow setting of MLP. But for KAN model, it has its unique interpatability and formula expression capacity, which can give us a direct insight about the dataset, and we therefore still consider KAN as a work with potential for future research fields.

Model	KAN	KANLayer with Keras	MLP
Loss	$10^{-2} \sim 10^{-4}$	$10^{-2} \sim 10^{-5}$	$10^{-3} \sim 10^{-5}$

TABLE VI: Loss Ranges for Different Models

VI. Further Discussions

In our experiments, we aimed to reproduce the results from the original paper and evaluate the KAN model under similar conditions. While the overall trends were consistent, we observed differences in PDE loss, convergence speed, and symbolic recovery, which revealed both strengths and limitations in our implementation.

For PDE loss, the original implementation achieved faster convergence and lower final values. In contrast, our version converged more slowly and resulted in higher residual losses. This difference is largely due to the behavior of the LBFGS optimizer. While both implementations used LBFGS, TensorFlow's static graph added overhead during gradient calculations and line searches. On the other hand, PyTorch's dynamic graph handled these operations more efficiently, which likely explains the smoother convergence in the original results.

Another key factor was computational resources. The original experiments utilized GPU acceleration, which provided faster computations and better numerical stability. In comparison, we conducted our experiments on a CPU due to resource limitations. This significantly reduced training efficiency, especially during grid refinement and symbolic regression, further contributing to slower convergence and suboptimal performance.

The grid refinement process in our implementation is also affected by limited performance of float dtype we adopted in the code. In some datasets , TensorFlow's static graph made dynamic grid updates more challenging to implement, resulting in less precise alignment with input data. However, we still got many good datasets on KAN performance as we shown above in part V after tons of experiments with KAN. Moreover, the model struggled with symbolic recovery, particularly for functions involving large coefficients or complex multiplicative terms. While the original implementation recovered these structures accurately, our version tended to focus on simpler terms, likely due to numerical limitations and the complexity of symbolic regression.

Despite these challenges, our experiments confirmed the importance of model depth. Deeper KAN architectures tends to perform better on complex tasks, as they captured hierarchical patterns in the target functions more effectively. But a wise rather than deep depth choice is more important since training is not the only metric we value. A wise depth

choice based on prior knowledge of the task makes KAN perform much better.

We state that KAN as an extraordinary work of 2024, does have a marvellous performance on the interpretability of deep learning, and therefore worths further study and more experiments, to broaden the boundary of fields like AI4Science. But it's not as good as what the authors of the original work said in their paper. However, we do agree with the last parts of their paper that **Choice really matters**: the readers should make wise choice on which network to choose, like what they do with the hyperparameters of any deep learning models they use. And we hope that more readers find KAN useful in a recent future.

VII. Conclusions & Future

In this report, we have thoroughly analyzed and implemented Kolmogorov–Arnold Networks (KAN) based on the Kolmogorov–Arnold Representation Theorem. The study demonstrates the effectiveness of KAN in addressing key challenges associated with multivariate function approximation, including scalability and interpretability. By designing and testing key components such as trainable univariate splines, grid refinement, and symbolic functionality, we verified the theoretical advantages of KAN and its performance in complex machine learning tasks. This work provides a comprehensive foundation for further exploration of KAN and its applications in both theoretical and practical domains.

Future efforts will focus on enhancing the optimization process by improving the performance of optimizers, particularly in high-dimensional training scenarios, to achieve faster convergence and more efficient parameter tuning. Addressing the issue of gradient explosion through advanced normalization techniques and better initialization strategies will be a priority to ensure stability during training. Additionally, further leveraging TensorFlow's built-in advantages, such as distributed training, automatic differentiation, and GPU acceleration, will help unlock KAN's full potential in large-scale and real-world applications. These advancements will aim to make KAN more robust, efficient, and widely applicable across diverse domains.

Acknowledgements

The authors would like to express their sincere gratitude to all those who contributed to the successful completion of this work. To finish this work, knowledge from course ECBM 4040 by Prof.Zoran Kostic from Columbia University helped a lot, so do the TAs of this course and all members that had great discussions over this topic with the authors.

References

[1] Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.

[3] George Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2(4):303–314, 1989.

[4] Altman, Naomi, and Martin Krzywinski. "The curse (s) of dimensionality." Nat Methods 15.6 (2018): 399-400.

[5] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. arXiv preprint arXiv:2309.08600, 2023.

[6] Ribeiro, Antônio H., et al. "Beyond exploding and vanishing gradients: analysing RNN training using attractors and smoothness." International conference on artificial intelligence and statistics. PMLR, 2020.

[7] Lan, Xinjie, and Kenneth E. Barner. "A Probabilistic Representation of Deep Learning for Improving The Information Theoretic Interpretability." arXiv preprint arXiv:2010.14054 (2020).

[8] Dong, Hangcheng, et al. "How to Explain Neural Networks: an Approximation Perspective." arXiv preprint arXiv:2105.07831 (2021).

[9] Lin, Ruiyuan, et al. "From two-class linear discriminant analysis to interpretable multilayer perceptron design." arXiv preprint arXiv:2009.04442 (2020).

[10] A.N. Kolmogorov. On the representation of continuous functions of several variables as superpositions of continuous functions of a smaller number of variables. Dokl. Akad. Nauk, 108(2), 1956.

[11] Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In Doklady Akademii Nauk, volume 114, pages 953–956. Russian Academy of Sciences, 1957.

[12] Jürgen Braun and Michael Griebel. On a constructive proof of kolmogorov's superposition theorem. Constructive approximation, 30:653–675, 2009.

[13] David A Sprecher and Sorin Draghici. Space-filling curves and kolmogorov superposition based neural networks. Neural Networks, 15(1):57–67, 2002.

[14] Mario Köppen. On the training of a kolmogorov network. In Artificial Neural Net works—ICANN 2002: International Conference Madrid, Spain, August 28–30, 2002 Pro ceedings 12, pages 474–479. Springer, 2002.

[15] Ji-Nan Lin and Rolf Unbehauen. On the realization of a kolmogorov network. Neural Com putation, 5(1):18–20, 1993.

[16] Ming-Jun Lai and Zhaiming Shen. The kolmogorov superposition theorem can break the curse of dimensionality when approximating high dimensional functions. arXiv preprint arXiv:2112.09963, 2021.

[17] Leni, Yohan D Fougerolle, and Frédéric Truchetet. The kolmogorov spline network for image processing. In Image Processing: Concepts, Methodologies, Tools, and Applications, pages 54–78. IGI Global, 2013.

[18] Daniele Fakhoury, Emanuele Fakhoury, and Hendrik Speleers. Exsplinet: An interpretable and expressive spline-based neural network. Neural Networks, 152:332–346, 2022.

[19] Hadrien Montanelli and Haizhao Yang. Error bounds for deep relu networks using the kolmogorov–arnold superposition theorem. Neural Networks, 129:1–6, 2020.

[20] Juncai He. On the optimal expressive power of relu dnns and its application in approximation with kolmogorov superposition theorem. arXiv preprint arXiv:2308.05509, 2023.

[21] Liu, Ziming, et al. "Kan: Kolmogorov-arnold networks." arXiv preprint arXiv:2404.19756 (2024).

[22] Liu, Ziming, et al. "Kan 2.0: Kolmogorov-arnold networks meet science." arXiv preprint arXiv:2408.10205 (2024).

[23] https://www.theinformation.com/articles/openai-shifts-strategy-as-rate-of-gpt-ai-improvements-slows?ref=platformer.news..

[24] **Link to this work:** https://github.com/ecbme4040/e4040-2024Fall-Project-KANY-hd2573-jx2598-yk3108.git

TABLE VII: Individual Student Contributions in Fractions

UNI	† **hd2573**	† **jx2598**	† **yk3108**
Last Name	Dong	Xiang	Ke
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Background Review	Results Review	Methods Review
What I did 2	Coding including Pipeline & Multkan	Coding including Multkan & LBFGS	Coding including Multkan, KanLayer, and Keras Kan
What I did 3	Experiments of V.1-V.4	Experiments of V.5-V.6	Experiments of V.7-V.10
What I did 4	Report Writing of Part I,III,IV,V,VII	Report Writing of Part II,III,V,VI	Report Writing of Part II,V,VI

† *All authors contributed equally.*