The Journal of Open Source Software

jaxKAN: A unified JAX framework for Kolmogorov-Arnold Networks

Spyros Rigas [1¶] **and Michalis Papachristou** [2]

1 Department of Digital Industry Technologies, School of Science, National and Kapodistrian University of Athens **2** Department of Physics, School of Science, National and Kapodistrian University of Athens ¶ Corresponding author

DOI: 10.21105/joss.07830

Software

- Review
- Repository
- Archive

Editor: Fabian Scheipl

Reviewers:

- @kazewong
- @ColCarroll

Submitted: 09 January 2025
Published: 18 March 2025

Summary

jaxKAN is a JAX-based library for building and training Kolmogorov–Arnold Networks (KANs) (Liu et al., 2024), built on Flax's NNX (Heek et al., 2024) with Optax (DeepMind et al., 2020) for optimization. It provides a broad selection of layer implementations - from the original KAN design to more recent or efficient variants - and unifies them under a single interface. Beyond basic model instance definition and training, jaxKAN supports class-inherent adaptive training methods (e.g., grid updates) and provides utilities that address performance limitations in the original KAN framework. KANs from jaxKAN can be used in any setting where standard multilayer perceptrons (MLPs) would otherwise be employed as the underlying architecture, although the library includes specialized utilities for adaptive Partial Differential Equation (PDE) solving tasks, thus placing emphasis on scientific applications with Physics-Informed Kolmogorov-Arnold Networks (PIKANs) (Shukla et al., 2024).

Statement of need

The recent introduction of KANs (Liu et al., 2024) in machine learning provided an alternative to traditional MLPs by making the network activation functions trainable, through expansions in terms of basis functions - originally B-splines. This novel idea proved to be effective across diverse applications, from engineering to scientific computing. Consequently, KANs were extended to incorporate alternative basis functions, such as polynomial (Sidharth et al., 2024) or trigonometric (Gist Noesis, 2024) functions. Moreover, more efficient architectures were proposed to overcome performance bottlenecks in the original design (Blealtan, 2024). As a result, a growing body of research has explored new ways to build, refine and utilize KANs, yet a comprehensive, high-performance software framework that integrates these ideas under one roof has been lacking.

Existing libraries only partially address this need. For instance, pykan (Liu, 2024), the original PyTorch-based (Paszke et al., 2019) KAN implementation, does not provide expandability to multiple layer types. Similarly, frameworks like NeuroMANCER (Drgona et al., 2023), while valuable for physics-informed applications, restrict KAN usage mainly to scientific computing tasks via the use of high-level APIs, thus limiting their applicability for other types of problems or low-level integration into novel deep learning architectures. Additionally, while decentralized repositories exist for individual KAN layer types, there is no unified approach to constructing and training KANs, making it challenging to combine them into hybrid architectures or incorporate them into broader workflows. This fragmented landscape perpetuates the tendency towards assembling custom, case-specific solutions by researchers and practitioners.

jaxKAN fills this gap by providing a JAX-native codebase that encompasses multiple KAN layer types, from the original B-spline-based approach to alternative variants. Through its unified

interface, users can not only choose from various basis expansions but also utilize built-in adaptive routines - such as grid updates, adaptive state transitions, or basis order extension - to tackle problems where static representations underperform. Moreover, it delivers specialized utilities for the adaptive training of PIKANs, a field where KANs have proven to be equally or even more efficient than MLPs in several occasions (Rigas et al., 2024; Shukla et al., 2024). At the time of writing, `jaxKAN` has been used in a series of academic works relevant to PDE problems (Howard, Jacob, Murphy, et al., 2024; Howard, Jacob, & Stinis, 2024; Jacob et al., 2024; Rigas et al., 2024), and has showcased significantly faster performance compared to the original `pykan` implementation (Rigas et al., 2024).

Core Functionality

In the following, a brief discussion on the core functionality of `jaxKAN` is provided.

Layer Selection

In `jaxKAN`, KANs are built as instances of the `KAN` class, which comprises one or more Kolmogorov–Arnold layers. These layers are defined within the `jaxkan.layers` module, which currently includes five types:

- `base`: the original B-spline-based layer.
- `spline`: the efficient variant of the original B-spline-based layer (Blealtan, 2024).
- `cheby`: a layer using Chebyshev polynomials.
- `mod-cheby`: a layer using Chebyshev polynomials without incorporating trigonometric functions for their calculation (Shukla et al., 2024).
- `fourier`: a layer utilizing sines and cosines as basis functions.

While these cover the most commonly used basis functions, the library will remain under active development, aiming to add further variants as research continues and identifies promising new options.

Network Methods

Each KAN instance provides three primary methods: initialization for customizing layer parameters, a forward pass for passing inputs through the network, and an `update_grids` method for in-place adjustments to each layer's grid. In particular, the `update_grids` method can be called during training with a new grid size as its argument to extend the network's current grid and subsequently adapt the extended grid to the training data. Although grid updates are naturally interpreted in spline-based layers, they also extend to other types: for Chebyshev layers, the method increases the degree of the polynomial basis functions, while in Fourier layers the method adds more terms to the Fourier sums. Notably, this method requires a technique to solve batched least-squares problems in parallel - an operation currently not supported natively by JAX libraries but implemented internally in `jaxKAN` for optimal performance.

PIKAN utilities

Beyond its low-level KAN functionality, `jaxKAN` supplies dedicated utilities for PIKANs. Users wishing to fine-tune every detail can extend the `KAN` class to their own needs, as illustrated in the documentation's tutorials. Alternatively, a higher-level `train_PIKAN` function automates the end-to-end adaptive training loop, allowing a PIKAN to solve a forward PDE problem with minimal overhead.

Case Study

As a case study, the Allen-Cahn equation, defined by

$$\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2} + 5\left(u^3 - u\right) = 0, \tag{1}$$

for $D = 10^{-3}$ in the $\Omega = [0, 1] \times [-1, 1]$ domain, subject to the boundary conditions

$$u\left(t = 0, x\right) = x^2 \cos\left(\pi x\right),$$

$$u\left(t, x = -1\right) = u\left(t, x = 1\right) = -1,$$

is solved, by training a network of `spline` layers for $5 \cdot 10^4$ epochs.

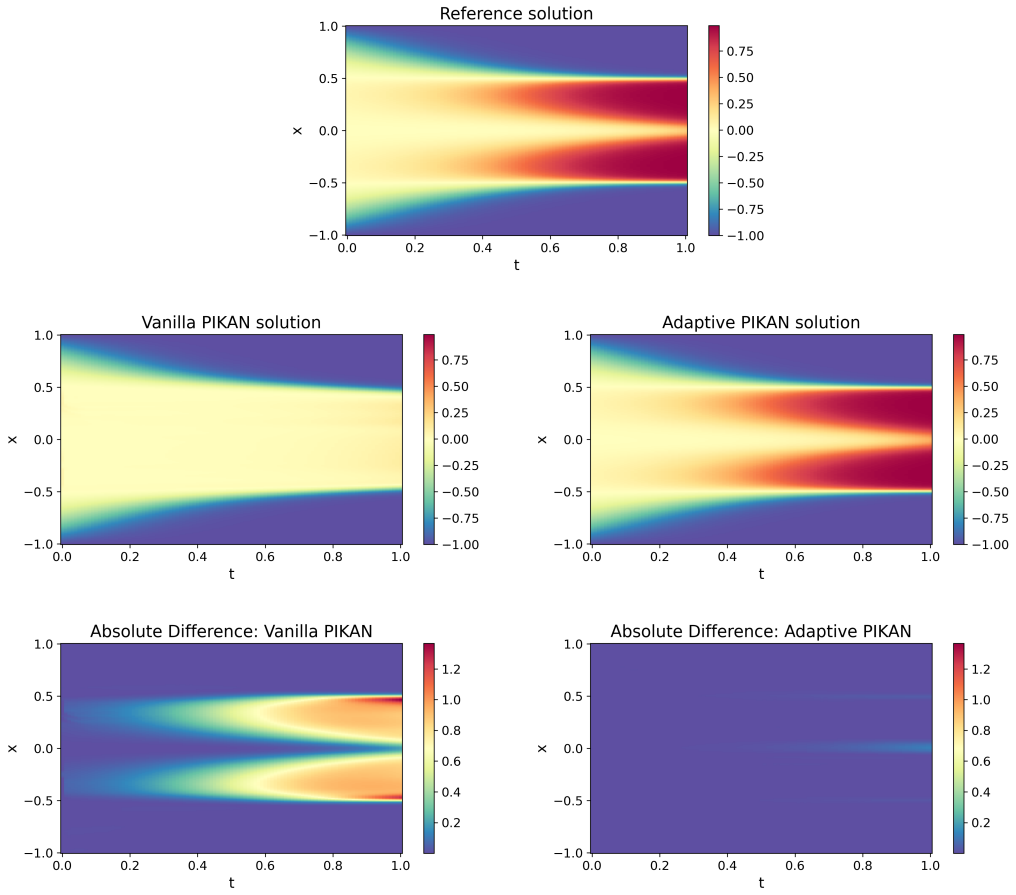

Figure 1: Upper row: reference solution to Equation 1. Middle row: approximation by vanilla PIKAN (left) and adaptive PIKAN (right). Lower row: absolute errors relative to the reference solution.

The reference solution to Equation 1 is depicted in the upper row of Figure 1; since the Allen-Cahn equation does not have an analytical solution, the reference solution used by Wu et al. (2023) is adopted herein. The plots in the middle row of Figure 1 depict two approximate solutions, corresponding to the results obtained after training a vanilla PIKAN, i.e., a PIKAN trained without adopting adaptive techniques, (left) and an adaptively trained PIKAN (right).

Finally, the lower row of Figure 1 showcases the absolute error corresponding to each PIKAN's approximation, relative to the reference. This example highlights the benefits of adaptive training, as the vanilla PIKAN fails to capture the details of the reference solution, especially in the $t \geq 0.3$ region.

References

Blealtan. (2024). An Efficient Implementation of Kolmogorov-Arnold Network. In *GitHub repository*. GitHub. https://github.com/Blealtan/efficient-kan

DeepMind, Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, D., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Dedieu, A., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., … Viola, F. (2020). The DeepMind JAX Ecosystem. In *GitHub repository*. GitHub. https://github.com/google-deepmind

Drgona, J., Tuor, A., Koch, J., Shapiro, M., Jacob, B., & Vrabie, D. (2023). NeuroMANCER: Neural Modules with Adaptive Nonlinear Constraints and Efficient Regularizations. In *GitHub repository*. GitHub. https://github.com/pnnl/neuromancer

Gist Noesis. (2024). FourierKAN. In *GitHub repository*. GitHub. https://github.com/GistNoesis/FourierKAN

Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., & van Zee, M. (2024). Flax: A neural network library and ecosystem for JAX. In *GitHub repository*. GitHub. https://github.com/google/flax

Howard, A. A., Jacob, B., Murphy, S. H., Heinlein, A., & Stinis, P. (2024). Finite basis Kolmogorov-Arnold networks: domain decomposition for data-driven and physics-informed problems. *ArXiv e-Prints*. https://arxiv.org/abs/2406.19662

Howard, A. A., Jacob, B., & Stinis, P. (2024). Multifidelity Kolmogorov-Arnold Networks. *ArXiv e-Prints*. https://arxiv.org/abs/2410.14764

Jacob, B., Howard, A. A., & Stinis, P. (2024). SPIKANs: Separable Physics-Informed Kolmogorov-Arnold Networks. *ArXiv e-Prints*. https://arxiv.org/abs/2411.06286

Liu, Z. (2024). Kolmogorov-Arnold Networks (KANs). In *GitHub repository*. GitHub. https://github.com/KindXiaoming/pykan

Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T. Y., & Tegmark, M. (2024). KAN: Kolmogorov-Arnold Networks. *ArXiv e-Prints*. https://arxiv.org/abs/2404.19756

Paszke, A., Gross, S., Massa, F., A., L., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *ArXiv e-Prints*. https://arxiv.org/abs/1912.01703

Rigas, S., Papachristou, M., Papadopoulos, T., Anagnostopoulos, F., & Alexandridis, G. (2024). Adaptive Training of Grid-Dependent Physics-Informed Kolmogorov-Arnold Networks. *IEEE Access*, *12*, 176982. https://doi.org/10.1109/ACCESS.2024.3504962

Shukla, K., Toscano, J. D., Wang, Z., Zou, Z., & Karniadakis, G. E. (2024). A comprehensive and FAIR comparison between MLP and KAN representations for differential equations and operator networks. *Computer Methods in Applied Mechanics and Engineering*, *431*, 117290. https://doi.org/10.1016/j.cma.2024.117290

Sidharth, S. S., Keerthana, A. R., Gokul, R., & Anas, K. P. (2024). Chebyshev Polynomial-Based Kolmogorov-Arnold Networks: An Efficient Architecture for Nonlinear Function Approximation. *ArXiv e-Prints*. https://arxiv.org/abs/2405.07200

Wu, C., Zhu, M., Tan, Q., Kartha, Y., & Lu, L. (2023). A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, *403*, 115671. https://doi.org/10.1016/j.cma.2022.115671