

Towards deep physics-informed Kolmogorov–Arnold networks

Spyros Rigas^{a,*}, Fotios Anagnostopoulos^b, Michalis Papachristou^c, Georgios Alexandridis^a

^a*Department of Digital Industry Technologies, School of Science, National and Kapodistrian University of Athens, 34400, Greece*

^b*Department of Informatics & Telecommunications, University of the Peloponnese, 22131, Greece*

^c*Department of Physics, School of Science, National and Kapodistrian University of Athens, 15784, Greece*

Abstract

Since their introduction, Kolmogorov–Arnold Networks (KANs) have been successfully applied across several domains, with physics-informed machine learning (PIML) emerging as one of the areas where they have thrived. In the PIML setting, Chebyshev-based physics-informed KANs (cPIKANs) have become the standard due to their computational efficiency. However, like their multilayer perceptron-based counterparts, cPIKANs face significant challenges when scaled to depth, leading to training instabilities that limit their applicability to several PDE problems. To address this, we propose a basis-agnostic, Glorot-like initialization scheme that preserves activation variance and yields substantial improvements in stability and accuracy over the default initialization of cPIKANs. Inspired by the PirateNet architecture, we further introduce Residual-Gated Adaptive KANs (RGA KANs), designed to mitigate divergence in deep cPIKANs where initialization alone is not sufficient. Through empirical tests and information bottleneck analysis, we show that RGA KANs successfully traverse all training phases, unlike baseline cPIKANs, which stagnate in the diffusion phase in specific PDE settings. Evaluations on seven standard forward PDE benchmarks under

*Corresponding author

Email addresses: spyrigas@uoa.gr (Spyros Rigas), fotisanagn@uop.gr (Fotios Anagnostopoulos), mixpap@phys.uoa.gr (Michalis Papachristou), gealexandri@uoa.gr (Georgios Alexandridis)

a fixed training pipeline with adaptive components demonstrate that RGA KANs consistently outperform parameter-matched cPIKANs and PirateNets – often by several orders of magnitude – while remaining stable in settings where the others diverge.

Keywords:

Kolmogorov–Arnold networks, physics-informed neural networks, partial differential equations, deep architectures, weight initialization, adaptive training

1. Introduction

The widespread adoption and integration of machine learning into computational science has profoundly influenced the way complex physical phenomena are modeled and analyzed. One of the most striking advances is the Physics-Informed Machine Learning (PIML) framework [1, 2], which offers a compelling alternative to traditional discretization-based solvers for both forward and inverse problems involving partial differential equations (PDEs). Within the PIML framework, the governing equations, alongside boundary and initial conditions, plus any observational data, are embedded into a differentiable loss function, while a neural network parametrizes the unknown solution field. Leveraging automatic differentiation [3] to evaluate differential operators exactly, PIML eliminates the need for mesh generation and yields continuous, high-fidelity predictions with reduced computational cost. As a result, it has found success across a broad spectrum of scientific and engineering disciplines, from fluid mechanics [4, 5, 6, 7] and materials science [8, 9] to medicine [10, 11, 12] and chemistry [13, 14].

While a variety of neural architectures have been explored within the PIML paradigm, including convolutional neural networks (CNNs) [15], generative adversarial networks (GANs) [16], and long short-term memory (LSTM) networks [17], the fully connected multilayer perceptron (MLP) is the predominant backbone. When an MLP parametrizes the solution field, the variant is conventionally termed a Physics-Informed Neural Network (PINN), which is also the original formulation of PIML [18]. Despite their widespread use, PINNs exhibit several well-documented shortcomings, including spectral bias toward low-frequency modes [19], restricted interpretability and limited scalability with depth, among other challenges in their training dynamics. To address these issues, numerous mitigation strategies have been proposed,

ranging from architectural modifications [20, 21, 22, 23], to adaptive training techniques [24, 25, 26]. A complementary approach is to forgo the MLP backbone altogether in favor of alternative architectures that mitigate several of these issues by design.

One such emerging alternative is the Kolmogorov–Arnold Network (KAN) [27]. Whereas the expressivity of MLPs is supported by the universal approximation theorem [28], KANs are grounded in the Kolmogorov–Arnold representation theorem [29]. In practice, a KAN layer replaces fixed non-linear activations with a learnable linear combination of basis functions; the original implementation employs B-splines, but other basis functions such as radial basis functions [30], Chebyshev polynomials [31] and Rectified Linear Unit (ReLU)-based functions [32] have also been explored. This design offers several benefits, most notably in terms of enhanced interpretability [33] and robustness against spectral bias [34]. These have motivated the development of Physics-Informed Kolmogorov–Arnold Networks (PIKANs) [35, 36], where the MLP backbone is substituted by a KAN in the PINN framework. Initial studies have demonstrated that PIKANs can attain higher accuracy on benchmark PDEs, or comparable accuracy with considerably smaller network architectures than their MLP-based counterparts [27, 35, 37]. Consequently, they have already seen practical deployment in a variety of scientific and engineering contexts [38, 39, 40].

Despite their promising early results, PIKANs present their own challenges. Computational overhead is the most immediate: evaluating and differentiating the B-spline basis in the original KAN formulation quickly becomes a bottleneck, leading most physics-informed implementations to adopt the more efficient Chebyshev variant (cPIKAN) [35]. Scalability is another concern; empirical studies report training instabilities as the number of network parameters increases beyond a point [41, 42], limiting applicability in deep learning regimes. Similar issues are observed in deep PINNs, although the PirateNet architecture appears to mitigate them [23]. Methodological gaps also persist. Weight initialization schemes are still largely ad hoc: each basis family (B-spline, radial-basis, Chebyshev, etc.) provides its own default, yet no analogue to the well-studied Glorot initialization for PINNs [43] has been empirically or theoretically analyzed. Systematic experimentation with initialization schemes is at a nascent stage and has so far concentrated exclusively on B-spline-based KANs [44]. The picture is similar for adaptive training strategies. While several PINN-oriented techniques have been ported to PIKANs [37, 45, 46], a unified training pipeline, comparable to the

one codified for PINNs [47], has yet to be established.

Motivated by these gaps, we concentrate on cPIKANs, which are a fitting choice for physics-informed applications in terms of their computational efficiency and accuracy. We first observe that the reported depth-related instabilities may be closely tied to weight initialization: an initialization that preserves activation variance can prevent vanishing or exploding gradients, just as the Glorot scheme does in MLPs. Accordingly, we derive a “Glorot-like” initialization for KANs that makes no assumptions about the specific basis and is therefore applicable to any KAN variant. On a series of function-fitting and PDE benchmarks, we show that this initialization improves optimization stability and yields significantly more accurate solutions than the default initialization of Chebyshev-based KANs and cPIKANs.

Building on this foundation, we address the depth-scaling issue. Because each KAN layer carries multiple learnable basis coefficients, a KAN layer of the same width as an MLP layer is substantially more parameter-heavy; real-world tasks that need larger capacity must therefore rely on greater depth, which in turn demands stable training. To this end, we introduce a Residual-Gated Adaptive KAN (RGA KAN) architecture. We analyze its training dynamics through the lens of the Information Bottleneck (IB) framework [48] and empirically demonstrate that RGA KANs remain stable and train effectively at depths where baseline cPIKANs diverge.

Finally, using our proposed initialization scheme and the RGA KAN architecture, we conduct extensive experiments on a suite of forward PDE problems. RGA KANs are compared with parameter-matched PirateNets and baseline cPIKANs under a standardized training pipeline that incorporates adaptive techniques drawn from PINN best practices. Ablation studies are also performed to quantify the influence of each adaptive component of the training pipeline, establishing a first set of depth-scalable benchmarks for cPIKANs and demonstrating that our contributions jointly close much of the performance and stability gap identified in earlier work.

In summary, the key contributions of this work are the following:

- We derive a basis-agnostic, Glorot-like initialization scheme that improves the accuracy of the studied KANs on both function-fitting and PDE-solving tasks.
- We introduce RGA KANs, designed to address the degradation in performance observed during the training of deep cPIKANs. We further analyze their training dynamics through the lens of IB theory.

- We benchmark RGA KANs against baseline cPIKANs and PirateNets on a suite of forward PDE problems, using identical adaptive training techniques across all models.
- Through ablation studies, we quantify the individual contributions of each adaptive technique to the overall performance of our proposed architecture.

The remainder of this paper is structured as follows. Section 2 reviews the theoretical foundations of our study, covering the PIML framework, KANs, and the adaptive training methods applied herein. Section 3 presents the proposed basis-agnostic Glorot-like initialization and demonstrates its clear advantage over the default cPIKAN initialization through small-scale function-fitting and PDE benchmarks. Section 4 addresses the depth-scaling limitations of cPIKANs by introducing the RGA KAN architecture; we analyze its training dynamics via the IB theory and show that it remains stable where cPIKANs diverge. Section 5 delivers a comprehensive empirical comparison among RGA KANs, baseline cPIKANs, and PirateNets on a suite of forward PDE problems, supplemented by ablation studies that isolate the contribution of each adaptive training component. Finally, Section 6 summarizes our principal findings and outlines promising directions for future research.

2. Theoretical Background

2.1. Problem Formulation

Without loss of generality, we consider PDEs of the form

$$\mathcal{F}[u(t, \mathbf{x})] = f(t, \mathbf{x}), \quad t \in [0, T], \quad \mathbf{x} \in \Omega, \quad (1)$$

defined over a bounded d -dimensional spatial domain $\Omega \subset \mathbb{R}^d$ with boundary $\partial\Omega$ and a temporal domain $[0, T]$, and subject to initial and boundary conditions

$$u(0, \mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \Omega, \quad (2)$$

$$\mathcal{R}_{bc}[u(t, \mathbf{x})] = 0, \quad t \in [0, T], \quad \mathbf{x} \in \partial\Omega. \quad (3)$$

In the above expressions, \mathcal{F} corresponds to an abstract differential operator, \mathcal{R}_{bc} is a boundary operator that imposes Dirichlet, Neumann, Robin, or

periodic boundary conditions, while $u(t, \mathbf{x})$ represents the solution of the PDE. Additionally, $f(t, \mathbf{x})$ and $g(\mathbf{x})$ are known functions corresponding to the PDE’s source term and initial condition, respectively.

The core idea behind PIML is to approximate the unknown solution by a neural network $u(t, \mathbf{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ denotes all trainable parameters of the network. To this end, we define the PDE’s residuals as

$$\mathcal{R}_{\text{pde}} [u(t, \mathbf{x}; \boldsymbol{\theta})] = \mathcal{F} [u(t, \mathbf{x}; \boldsymbol{\theta})] - f(t, \mathbf{x}), \quad (4)$$

and the initial condition’s residuals as

$$\mathcal{R}_{\text{ic}} [u(t, \mathbf{x}; \boldsymbol{\theta})] = u(0, \mathbf{x}; \boldsymbol{\theta}) - g(\mathbf{x}). \quad (5)$$

Then, the neural network is trained by minimizing the composite loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_{\text{pde}} \mathcal{L}_{\text{pde}}(\boldsymbol{\theta}) + \lambda_{\text{ic}} \mathcal{L}_{\text{ic}}(\boldsymbol{\theta}) + \lambda_{\text{bc}} \mathcal{L}_{\text{bc}}(\boldsymbol{\theta}), \quad (6)$$

where λ_{pde} , λ_{ic} , λ_{bc} are hyperparameters that allow the assignment of different weights to each individual term of the composite loss function and

$$\mathcal{L}_{\text{pde}}(\boldsymbol{\theta}) = \frac{1}{N_{\text{pde}}} \sum_{i=1}^{N_{\text{pde}}} \|\mathcal{R}_{\text{pde}} [u(t_{\text{pde}}^i, \mathbf{x}_{\text{pde}}^i; \boldsymbol{\theta})]\|_2^2, \quad (7)$$

$$\mathcal{L}_{\text{ic}}(\boldsymbol{\theta}) = \frac{1}{N_{\text{ic}}} \sum_{i=1}^{N_{\text{ic}}} \|\mathcal{R}_{\text{ic}} [u(0, \mathbf{x}_{\text{ic}}^i; \boldsymbol{\theta})]\|_2^2, \quad (8)$$

$$\mathcal{L}_{\text{bc}}(\boldsymbol{\theta}) = \frac{1}{N_{\text{bc}}} \sum_{i=1}^{N_{\text{bc}}} \|\mathcal{R}_{\text{bc}} [u(t_{\text{bc}}^i, \mathbf{x}_{\text{bc}}^i; \boldsymbol{\theta})]\|_2^2, \quad (9)$$

where $\|\cdot\|_2$ denotes the L^2 norm and $\{(t_{\xi}^i, \mathbf{x}_{\xi}^i)\}_{i=1}^{N_{\xi}}$, with ξ being either “pde”, “ic” or “bc”, correspond to collocation points used to calculate the PDE’s, initial condition’s and boundary conditions’ residuals, respectively. We remark that the global λ -weights can either be defined based on domain knowledge (e.g., [35]), or adjusted dynamically during the network’s training [24, 25, 42, 49]. Moreover, we note that the set of collocation points used to calculate the PDE’s residuals can be sampled once from a fixed grid or adaptively re-sampled throughout training [50, 51, 52, 53].

2.2. Kolmogorov–Arnold Networks

Until recently, the vast majority of the neural network architectures which were chosen to approximate the PDE’s solution utilized MLPs as their backbone. In an MLP, the output of the l -th layer is recursively defined in terms of the output of the $(l - 1)$ -th layer as follows:

$$u_j^{(l)}(t, \mathbf{x}; \boldsymbol{\theta}) = \sigma \left(\sum_{i=1}^{d_{l-1}} w_{ji}^{(l)} u_i^{(l-1)}(t, \mathbf{x}; \boldsymbol{\theta}) + b_j^{(l)} \right), \quad (10)$$

where $w_{ji}^{(l)}, b_j^{(l)}$ represent the weights and biases of the l -th layer, d_{l-1} is the output dimension of the $(l - 1)$ -th layer and σ is a non-linear activation function – typically the hyperbolic tangent for PINNs. For the recursion to be consistent, the first layer is assumed to perform an identity operation, i.e.,

$$u_j^{(0)} = \begin{cases} t, & j = 0, \\ x_j, & j \in \{1, \dots, d\}. \end{cases} \quad (11)$$

For an MLP with an input layer of dimension d_I , L hidden layers each of dimension d_H and an output layer of dimension d_O , the cardinality of the set of the network’s parameters is given by

$$|\boldsymbol{\theta}| = d_H [d_I + (L - 1) d_H + L + d_O] + d_O = \mathcal{O}(d_H^2 L). \quad (12)$$

Inspired by the Kolmogorov–Arnold representation theorem [29], the authors of [27] introduced KANs, a novel class of neural networks that have since been adopted as an alternative to MLPs. The formulation corresponding to Eq. (10) for the original implementation of KANs, known as “vanilla” KANs, is given by

$$u_j^{(l)}(t, \mathbf{x}; \boldsymbol{\theta}) = \sum_{i=1}^{d_{l-1}} \left(r_{ji}^{(l)} R \left[u_i^{(l-1)}(t, \mathbf{x}; \boldsymbol{\theta}) \right] + c_{ji}^{(l)} \sum_{m=1}^D w_{jim}^{(l)} B_m^{(l)} \left[u_i^{(l-1)}(t, \mathbf{x}; \boldsymbol{\theta}) \right] \right), \quad (13)$$

where $r_{ji}^{(l)}, c_{ji}^{(l)}$ and $w_{jim}^{(l)}$ are the trainable parameters of the l -th layer,

$$R(x) = \frac{x}{1 + \exp(-x)} \quad (14)$$

is a residual function and $B_m^{(l)}(\cdot)$ are univariate spline basis functions. The superscript (l) reflects the inherent dependency of the spline basis functions on a layer-specific grid, while the subscript m runs from 1 to $D = G + k$, where G is the number of grid intervals and k is the order of the basis functions. Comparing Eq. (13) to Eq. (10), a fundamental distinction arises between MLPs and KANs in terms of their functional representation: in MLPs, nonlinearity is introduced through fixed activation functions, while only the linear transformations between layers are trainable; in contrast, KANs replace these static activation functions with learnable ones, an aspect that highlights their potential to create more expressive architectures [27, 34].

While vanilla KANs have demonstrated promising results in solving forward PDE problems [36, 37], their training is burdened by the expensive computation of spline basis functions. Additionally, their dependency on a grid, although beneficial in certain applications [27, 33, 37], becomes redundant when the grid remains fixed throughout training. To mitigate these inefficiencies, some approaches have introduced optimization strategies [54, 55], while others have explored alternative, more computationally efficient basis functions [30, 31, 56]. In this work, we adopt the latter approach and use Chebyshev-based KANs, due to their proven success in PIML [35, 40, 42, 57]. We therefore modify the expression of Eq. (13) to

$$u_j^{(l)}(t, \mathbf{x}; \boldsymbol{\theta}) = \sum_{i=1}^{d_{l-1}} \sum_{m=1}^D w_{jim}^{(l)} B_m \left[u_i^{(l-1)}(t, \mathbf{x}; \boldsymbol{\theta}) \right] + b_j^{(l)}, \quad (15)$$

where the residual term has now been removed, the c_{ji} weights have been absorbed by w_{jim} , an additional bias term, b_j , has been introduced and B_m are now grid-independent basis functions with $m = 1, \dots, D$. These basis functions are given by

$$B_m(x) = T_m(\tanh(x)), \quad (16)$$

where $T_m(\cdot)$ are Chebyshev polynomials of the first kind. The hyperbolic tangent in the argument of T_m maps x to the $[-1, 1]$ range, where the absolute value of the Chebyshev polynomials is bounded by 1. For the purposes of the present work, the Chebyshev polynomials are explicitly defined as functions up to order D to maximize computational efficiency.

Assuming a KAN architecture equivalent to the previously discussed MLP, with input dimension d_I , L hidden layers of dimension d_H and output dimension d_O , the total number of trainable parameters is given by

$$|\boldsymbol{\theta}| = d_H [d_I D + D (L - 1) d_H + L + d_O D] + d_O = \mathcal{O}(d_H^2 DL). \quad (17)$$

Comparing Eq. (17) to Eq. (12), a fundamental limitation of KANs becomes evident: for architectures with the same depth and width, a KAN contains approximately D times more parameters than its MLP counterpart. Consequently, the primary challenge in training KANs is to achieve equal or superior performance to MLPs while maintaining a comparable total number of parameters.

2.3. Adaptive Training Methods

Regardless of the backbone architecture, the loss function minimized via gradient descent in PIML is generally more complex than in conventional neural networks, as it involves not only the neural network’s output but also its gradients of various orders with respect to different input variables. This leads to a highly intricate loss landscape, where reaching minima requires not only expressive architectures but also effective adaptive training techniques. In this work, we port four such techniques which have been extensively utilized in PINNs: one focuses on the selection of collocation points for loss evaluation, while the other three introduce modifications to the composite loss function itself.

2.3.1. Collocation Points Resampling

The selection of collocation points used to enforce the PDE can be performed in a one-time manner, where a fixed set of training points is used throughout the entire training process, either with or without mini-batching. However, periodically resampling the collocation points during training can serve as a regularization technique, improving the network’s ability to generalize to spatiotemporal regions that were not explicitly sampled. Moreover, if the resampling process is not purely random but instead adaptive – e.g., guided by residuals or other heuristics – it has been shown to significantly enhance the final accuracy of the trained network [50, 52, 53].

For this study, we adopt the Residual-Based Adaptive Distribution (RAD) technique [51]. In particular, an initial dense pool of N_{pool} collocation points,

$\{(t_{\text{pool}}^i, \mathbf{x}_{\text{pool}}^i)\}_{i=1}^{N_{\text{pool}}}$, is generated from a uniform grid, and training is performed on a dynamically selected subset of $N_{\text{pde}} \ll N_{\text{pool}}$ points. These points are periodically resampled from the pool according to the probability density function

$$p(t_{\text{pool}}, \mathbf{x}_{\text{pool}}) = \frac{\|\mathcal{R}_{\text{pde}}[u(t_{\text{pool}}, \mathbf{x}_{\text{pool}}; \boldsymbol{\theta})]\|_2^\delta}{\frac{1}{N_{\text{pool}}} \sum_{i=1}^{N_{\text{pool}}} \|\mathcal{R}_{\text{pde}}[u(t_{\text{pool}}^i, \mathbf{x}_{\text{pool}}^i; \boldsymbol{\theta})]\|_2^\delta} + C, \quad (18)$$

where $\delta \geq 0$, $C \geq 0$ are hyperparameters of the method. This resampling strategy directs the network’s training toward regions where the PDE residuals are larger, which can be particularly beneficial in scenarios involving discontinuities or sharp gradients in the PDE solution.

2.3.2. Global Loss Weighting

When minimizing a loss function composed of multiple terms, a common challenge arises from the fact that different terms may converge at different rates. As a result, selecting appropriate weights for each term is a necessity. For instance, in L2 regularization, the choice of the regularization coefficient significantly impacts training: an excessively large value can impede learning, while a very small value may render the regularization effect negligible. A similar issue occurs in PIML, where there is often a bias toward minimizing certain terms of Eq. (6) while neglecting others.

In this work, we address this issue using the learning-rate annealing algorithm introduced in [20]. During training, the weight adjustment is guided by the computation

$$\hat{\lambda}_\xi = \frac{\|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{pde}}(\boldsymbol{\theta})\|_2 + \|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{ic}}(\boldsymbol{\theta})\|_2 + \|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{bc}}(\boldsymbol{\theta})\|_2}{\|\nabla_{\boldsymbol{\theta}} \mathcal{L}_\xi(\boldsymbol{\theta})\|_2}, \quad (19)$$

ξ represents either “pde”, “ic” or “bc”. The loss weights of Eq. (6) are then updated according to the rule

$$\lambda_\xi^{\text{new}} = a\lambda_\xi^{\text{old}} + (1 - a)\hat{\lambda}_\xi, \quad (20)$$

where a is the method’s hyperparameter. The initial values are set to $\lambda_{\text{pde}} = \lambda_{\text{ic}} = \lambda_{\text{bc}} = 1$ to ensure equal weighting at the start of training, although in cases where domain knowledge is available, more suitable initializations can be selected. The update of Eq. (20) is performed periodically at a predetermined interval.

2.3.3. Causal Training

A common challenge in training neural networks to solve time-dependent PDEs is the violation of causality. Since collocation points are sampled from the entire temporal domain, the network may unintentionally minimize residuals associated with future states before adequately minimizing those corresponding to past states. To mitigate this issue, following [25], we partition the temporal domain into M sequential segments of equal length and introduce temporal weights $\{w_i\}_{i=1}^M$, modifying Eq. (7) as

$$\mathcal{L}_{\text{pde}}(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i=1}^M w_i \mathcal{L}_{\text{pde}}^i(\boldsymbol{\theta}), \quad (21)$$

where $\mathcal{L}_{\text{pde}}^i(\boldsymbol{\theta})$ represents the PDE loss computed over collocation points whose temporal coordinates fall within the i -th segment. The temporal weights are updated at each training iteration (epoch) according to

$$w_i = \exp\left(-\epsilon \sum_{j=1}^{i-1} \mathcal{L}_{\text{pde}}^j(\boldsymbol{\theta})\right), \quad (22)$$

where $\epsilon \geq 0$ is a hyperparameter controlling the influence of the cumulative loss from the first $i - 1$ segments on the weight assigned to the i -th segment. In practice, this enforces a time-ordered minimization of the PDE residuals, ensuring that earlier time steps are prioritized before the network attempts to minimize residuals in future states.

2.3.4. Residual-Based Attention

The learning-rate annealing algorithm introduced in Section 2.3.2 is essentially a global loss weighting scheme, while causal training corresponds to a batch-wise weighting strategy, where groups of collocation points share the same weights. However, point-wise multipliers have also demonstrated significant success in PIML [58] and have been integral to works achieving state-of-the-art results [42, 59]. In such methods, the individual loss terms in Eqs. (7)-(9) are modified as

$$\mathcal{L}_{\xi}(\boldsymbol{\theta}) = \frac{1}{N_{\xi}} \sum_{i=1}^{N_{\xi}} \|\alpha_{\xi}^i \mathcal{R}_{\xi}[u(t_{\xi}^i, \mathbf{x}_{\xi}^i; \boldsymbol{\theta})]\|_2^2, \quad (23)$$

where ξ represents either ‘‘pde’’, ‘‘ic’’ or ‘‘bc’’ and α_{ξ}^i is the local weight assigned to the i -th collocation point.

For this study, we adopt the Residual-Based Attention (RBA) method introduced in [26], where all local weights are initially set to 1 and updated at each training iteration according to

$$\alpha_{\xi}^{i \text{ (new)}} = \gamma \alpha_{\xi}^{i \text{ (old)}} + \eta \frac{\|\mathcal{R}_{\xi} [u(t_{\xi}^i, \mathbf{x}_{\xi}^i; \boldsymbol{\theta})]\|_2}{\max_j \left(\left\{ \|\mathcal{R}_{\xi} [u(t_{\xi}^j, \mathbf{x}_{\xi}^j; \boldsymbol{\theta})]\|_2 \right\}_{j=1}^{N_{\xi}} \right)} \quad (24)$$

where $\gamma \geq 0$, $\eta \geq 0$ are hyperparameters of the method.

In theory, while RBA is not inherently incompatible with causal training – as the batch-wise weights can be computed using the modified loss of Eq. (23) – the same does not hold for RAD as presented in 2.3.1. Since RAD involves resampling collocation points, it removes the one-to-one correspondence between them and their local weights. In order to apply these methods in conjunction, we adopt the following strategy: instead of assigning local weights solely to the currently selected N_{pde} training points, we define the RBA weights over the entire pool, i.e., $\alpha_{\text{pool}}^i = 1$ for $i = 1, \dots, N_{\text{pool}}$ at initialization. These weights are updated at each training iteration according to Eq. (24), using the residuals evaluated at the corresponding points. Then, during each application of RAD, the probability density function in Eq. (18) is modified by incorporating the RBA weights as multiplicative factors on the residuals:

$$p(t_{\text{pool}}, \mathbf{x}_{\text{pool}}) = \frac{\|\alpha_{\text{pool}} \mathcal{R}_{\text{pde}} [u(t_{\text{pool}}, \mathbf{x}_{\text{pool}}; \boldsymbol{\theta})]\|_2^{\delta}}{\frac{1}{N_{\text{pool}}} \sum_{i=1}^{N_{\text{pool}}} \|\alpha_{\text{pool}}^i \mathcal{R}_{\text{pde}} [u(t_{\text{pool}}^i, \mathbf{x}_{\text{pool}}^i; \boldsymbol{\theta})]\|_2^{\delta}} + C, \quad (25)$$

This modification effectively biases the sampling process toward regions where the weighted residuals are larger, allowing RAD to leverage the localized attention mechanism introduced by RBA. As a result, the method retains the adaptive sampling benefits of RAD while amplifying its focus on regions deemed important by RBA.

3. KAN Initialization

Initialization plays a critical role in the training dynamics of deep neural networks. In the case of MLPs, extensive theoretical and empirical work has led to well-established initialization schemes tailored to different architectures and activation functions. Within the PINN framework, architectures are

most often initialized using the Glorot scheme [47], which aims to preserve the variance of activations and gradients across layers to avoid vanishing or exploding signals, thereby enabling stable deep learning.

For KANs – and, by extension, PIKANs – the choice of weight initialization remains largely ad hoc, with each variant in the literature adopting its own heuristic procedure. In contrast to the extensive body of work on MLP initialization, systematic studies for KANs are scarce and have thus far been limited to the vanilla formulation with B-spline bases [44], where power-law and LeCun-like schemes were proposed. Building on these early efforts, we develop a Glorot-like initialization derived from a variance-preservation analysis and formulated to be agnostic to the choice of basis functions. In the present work, we assess its performance in Chebyshev-based KANs and cPIKANs, demonstrating that it offers improved training stability and accuracy over the current standard initialization scheme.

3.1. Proposed Scheme

Consider a single KAN layer with input $\mathbf{x} \in \mathbb{R}^{d_I}$ and output $\mathbf{y} \in \mathbb{R}^{d_O}$. Based on Eq. (15), the j -th output component is given by

$$y_j = \sum_{i=1}^{d_I} \sum_{m=1}^D w_{jim} B_m(x_i) + b_j, \quad (26)$$

where $B_m(\cdot)$ denotes the m -th basis function. Throughout this derivation we set all biases b_j to zero at initialization and focus on the initialization of the coefficients w_{jim} .

In the MLP setting, weight initialization is typically modeled by assuming i.i.d. Gaussian entries drawn from $\mathcal{N}(0, \sigma^2)$, and selecting σ according to a chosen criterion. Here, the presence of an additional basis index, m , motivates the more general assumption

$$w_{jim} \sim \mathcal{N}(0, \sigma_m^2), \quad (27)$$

where σ_m is a basis-term-dependent standard deviation to be determined. For the inputs x_i , we assume i.i.d. samples with zero mean and unit variance, consistent with common deep learning practices.

A central principle in initialization design, which is also the core idea behind the LeCun initialization [60], is to preserve the variance of the signal during the forward pass. Applying this condition to Eq. (26) gives

$$1 = d_I \sum_{m=1}^D \sigma_m^2 \mu_m^{(0)}, \quad (28)$$

where

$$\mu_m^{(0)} := \mathbb{E} [B_m(x)^2], \quad (29)$$

and the expectation is taken with respect to the distribution of x .

In his original work, Glorot [43] further required variance preservation during the backward pass, so that gradients propagate across layers without amplification or attenuation. Applying the same reasoning to the gradient signal through Eq. (26) leads to

$$1 = d_O \sum_{m=1}^D \sigma_m^2 \mu_m^{(1)}, \quad (30)$$

where

$$\mu_m^{(1)} := \mathbb{E} [B'_m(x)^2], \quad (31)$$

with $B'_m(\cdot)$ denoting the derivative of the m -th basis function. Eqs. (28) and (30) thus impose the forward- and backward-variance constraints. Balancing them in the spirit of Glorot leads to

$$\sigma_m^2 = \frac{1}{D} \cdot \frac{2}{d_I \mu_m^{(0)} + d_O \mu_m^{(1)}}. \quad (32)$$

This expression defines a basis-agnostic Glorot-like initialization rule: $\mu_m^{(0)}$ and $\mu_m^{(1)}$ capture the effect of the chosen basis functions on variance, while d_I and d_O play the same role as in the original Glorot scheme. The additional factor $1/D$ accounts for the contribution of the D basis terms associated with each input dimension. In the special case $\mu_m^{(0)} = \mu_m^{(1)} = 1$ and $D = 1$, corresponding to the MLP setting, which can be interpreted as using a single basis function, Eq. (32) reduces exactly to the standard Glorot initialization. Detailed derivations of Eqs. (28) and (30) are provided in Appendix A.1.

For practical applications, it is often convenient to introduce a multiplicative gain factor, following common practice in initialization utilities provided by deep learning frameworks such as PyTorch [61]. This gain term allows for empirical correction in cases where the input distribution deviates from the

unit-variance assumption. Incorporating this factor, the final initialization rule becomes

$$\sigma_m = \text{gain} \sqrt{\frac{1}{D} \cdot \frac{2}{d_I \mu_m^{(0)} + d_O \mu_m^{(1)}},} \quad (33)$$

which recovers the standard Glorot initialization when $\mu_m^{(0)} = \mu_m^{(1)} = 1$, $D = 1$ and $\text{gain} = 1$.

3.2. Small-Scale Benchmarks

To evaluate the effectiveness of the proposed initialization scheme, we compare it against the prevailing practice for Chebyshev-based KANs. In this default approach, the basis-function coefficients w_{jim} are drawn from a normal distribution with zero mean and variance $[d_I(D + 1)]^{-1}$ [31, 42]. To this end, we conduct experiments on two small-scale benchmarks: (i) function fitting, and (ii) forward PDE problems.

3.2.1. Function Fitting

We first assess initialization performance on five function-fitting tasks of increasing dimensionality: (i) a one-dimensional oscillatory function, $f_1(x)$, (ii) the two-dimensional product function, $f_2(x_1, x_2)$, (iii) a more challenging two-dimensional function borrowed from [44], $f_3(x_1, x_2)$, (iv) the three-dimensional Hartmann function, $f_4(x_1, x_2, x_3)$, which is a common benchmark in function approximation, and (v) the five-dimensional Sobol g-function, $f_5(x_1, \dots, x_5)$, widely used in global sensitivity analysis. The analytic definitions of all functions are provided in Appendix C.

For each function, we generated $4 \cdot 10^3$ input–output samples uniformly over the domain $[-1, 1]^d$, with d being the dimensionality of the function. Chebyshev-based KANs with polynomial order $D = 8$ were trained under both the default initialization and the proposed Glorot-like scheme. In all cases, we modulated the gain factor to account for the standard deviation of inputs sampled from $\mathcal{U}(-1, 1)$, a convention we adopt throughout all subsequent experiments. Networks were trained for $2 \cdot 10^3$ iterations using the Adam [62] optimizer with a constant learning rate of 10^{-3} in full-batch mode. Architectures of varying width (hidden layer dimensions of 2, 4, 8, 16, 32, 64) and depth (2 to 5 hidden layers) were considered. Training minimized the L^2 loss, and final performance was evaluated in terms of the relative L^2 error with respect to the reference solution (see Appendix B for further details).

To ensure statistical significance, each configuration was repeated with five random seeds. For the final relative L^2 error evaluation, we used 1,000 uniformly spaced points in $[-1, 1]$ for the one-dimensional task, a 200×200 grid for the two-dimensional tasks, a 30^3 grid for the three-dimensional Hartmann function, and a 10^5 grid for the five-dimensional Sobol g-function.

The results of these experiments are summarized in Figure 1. Each heatmap corresponds to one benchmark function, with hidden layer dimension on the horizontal axis and number of hidden layers on the vertical axis. The color scale quantifies the relative improvement achieved by the proposed initialization over the default scheme, computed as

$$\frac{\mathcal{E}_{\text{default}} - \mathcal{E}_{\text{proposed}}}{\mathcal{E}_{\text{default}}} \times 100\%,$$

where \mathcal{E} denotes the relative L^2 error. Values are clipped to the range 0–100%, with black cells indicating cases where the default initialization outperforms the proposed method. Evidently, the impact of the proposed initialization is substantial in most benchmarks. For the two-dimensional functions (f_2 and f_3) and the three-dimensional Hartmann function (f_4), improvements

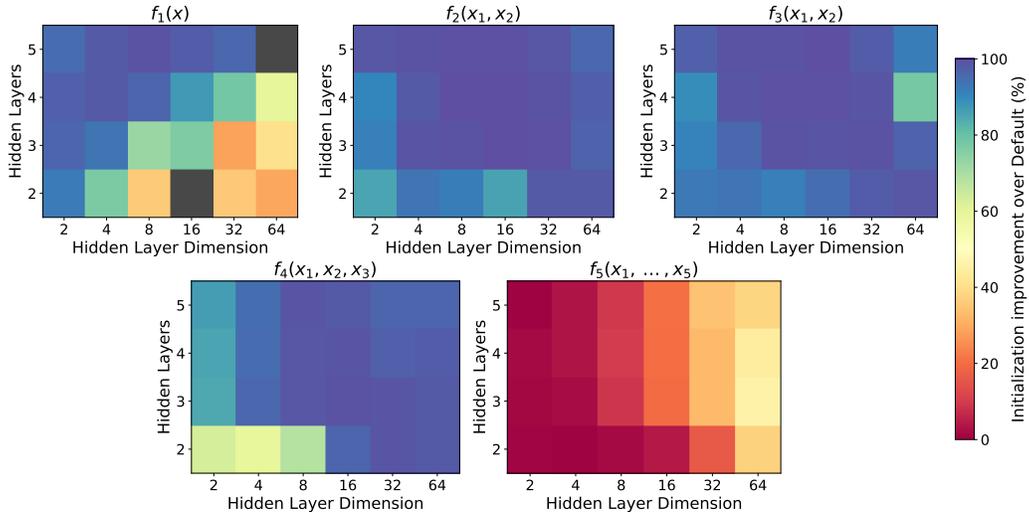


Figure 1: Relative comparison of proposed and default initialization across the five benchmark functions. Each subplot corresponds to one function, with the color scale indicating the percentage improvement of the proposed initialization over the default in terms of the final L^2 error. Black cells denote configurations where the default initialization attains lower error.

approach 100% across nearly all architectures, indicating that the proposed initialization reduces the final relative L^2 error by up to several orders of magnitude compared to the default scheme. The one-dimensional oscillatory function (f_1) also shows clear gains in the majority of cases, with only two isolated configurations where the default (marginally) outperforms. The five-dimensional Sobol g-function (f_5) exhibits improvements as well, though they are less pronounced, typically in the range of 5–50%; in this setting, both initializations yield comparable overall accuracy, and therefore the difference in initialization impact is less striking.

Apart from the final error metrics, it is also informative to examine the training loss evolution, in order to assess whether the proposed initialization leads to a more effective optimization of the loss function. To this end, we considered two representative architectures: a smaller network with width 4 and depth 3, and a larger network with width 16 and depth 5. Figure 2 depicts the training loss curves for each of the five benchmark functions under both initialization schemes. Solid lines indicate the mean loss across five independent runs, while the shaded regions correspond to the standard error. The proposed initialization consistently accelerates convergence and achieves substantially lower training losses. For the oscillatory function (f_1)

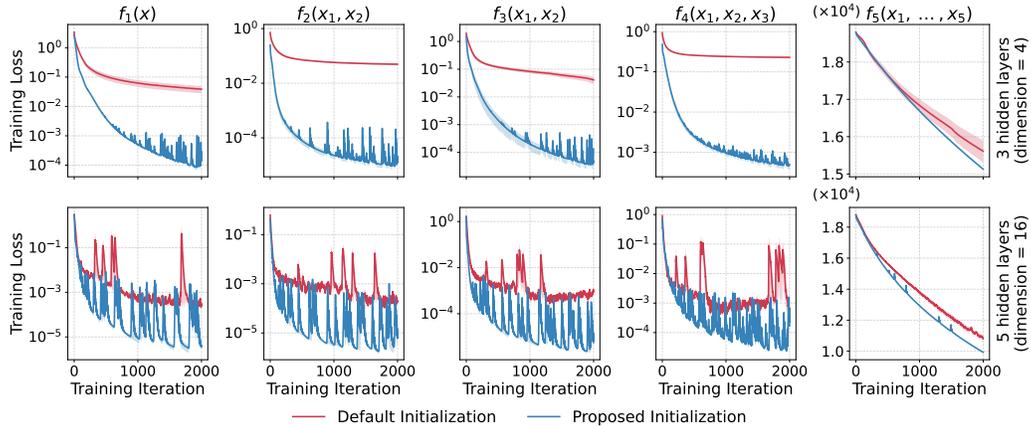


Figure 2: Loss throughout training for two representative architectures (top row: three 4-dimensional hidden layers; bottom row: five 16-dimensional hidden layers) across the five benchmark functions. Each subplot shows the mean training loss over five independent runs (solid lines) together with the standard error (shaded area). The final column, corresponding to f_5 , is shown without logarithmic scaling on the y -axis, since the loss did not exhibit significant improvement during training.

and the two-dimensional cases (f_2 and f_3), the difference spans more than two orders of magnitude, with low variability across seeds. Similar improvements are observed for the Hartmann function (f_4), where the default initialization stalls at higher loss values compared to the proposed scheme. For the Sobol g -function (f_5), both schemes exhibit nearly identical behavior, in line with the earlier observation that this benchmark is less sensitive to initialization. Importantly, these trends are visible in both architectures, demonstrating robustness across different model scales.

3.2.2. Forward PDE Problems

We next assess the proposed initialization on forward PDE benchmarks using cPIKANs. Specifically, we consider Burgers’ equation as well as the Helmholtz equation with $a_1 = 1$, $a_2 = 4$ (see Appendix D for details), following the PIML framework introduced in Section 2, without incorporating any additional adaptive techniques.

For each PDE, $N_{\text{pde}} = 2^{12}$ collocation points are used to enforce the differential operator. In the Burgers’ case, $N_{\text{bc}} = 2^6$ points are used for each of the two boundary conditions, together with $N_{\text{ic}} = 2^6$ points for the initial condition. For Helmholtz, $N_{\text{bc}} = 2^6$ points are used for each of the four boundary conditions. Collocation points are sampled once from a uniform grid and remain fixed throughout training. The trained Chebyshev-based KANs use polynomial order $D = 8$ and are initialized with either the default or the proposed scheme. Training is performed for $5 \cdot 10^3$ iterations using the Adam optimizer with a constant learning rate of 10^{-3} in full-batch mode. Architectures of varying width (2, 4, 8, 16, 32, 64 units per hidden layer) and depth (2 to 5 hidden layers) are considered. Models are evaluated in terms of the relative L^2 error with respect to the reference solution, and each experiment is repeated with five different random seeds.

The results of these experiments are summarized in Figure 3, which combines heatmaps of final relative L^2 error improvements with representative training-loss curves. As in the function-fitting benchmarks, the proposed initialization consistently outperforms the default scheme in terms of final error, with the exception of only two small architectures where the default initialization yields a marginal advantage. The overall gains are somewhat less pronounced than in Section 3.2.1, which can be attributed to the relatively low number of training iterations and the absence of adaptive training techniques. Nevertheless, the training-loss curves reveal a striking effect of initialization. In particular, for the Helmholtz equation, where it is well

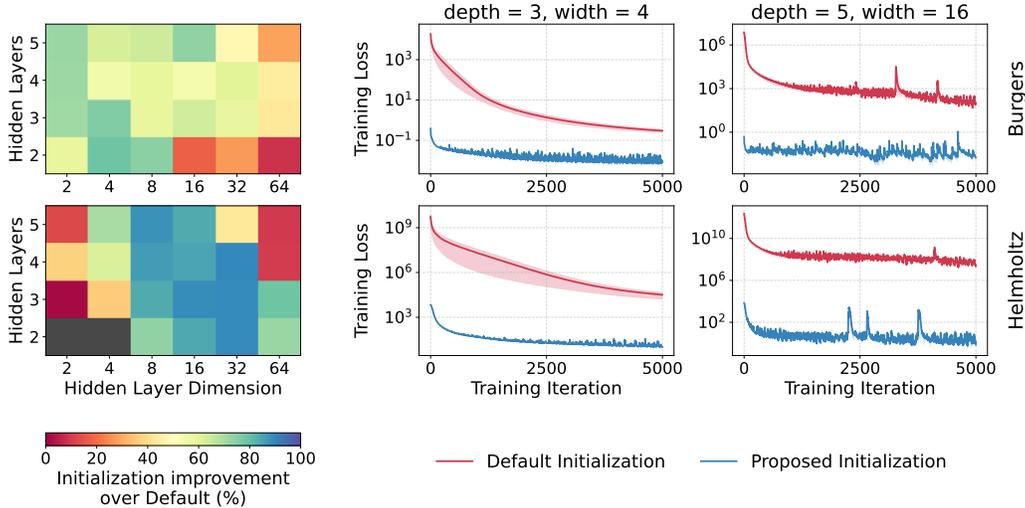


Figure 3: Comparison of default and proposed initialization schemes on Burgers’ (top row) and Helmholtz (bottom row) equations. Left column: heatmaps of relative improvement in final L^2 error. Middle/Right column: training-loss curves per initialization scheme for a representative architecture of depth = 3/5 and width = 4/16, respectively. Shaded regions denote standard error across five runs.

known that fast convergence usually requires adaptive weighting of PDE and boundary condition terms (see, e.g., [35]), the proposed initialization achieves up to eight orders of magnitude lower training loss compared to the default scheme. This difference corresponds not only to better convergence but, in practice, to the distinction between KANs that diverge under the default initialization and KANs that successfully approximate the true solution.

3.3. Training Divergence with Increasing Depth

The benchmarks considered so far involved relatively shallow networks, consistent with most current KAN applications, where architectures are typically limited to a small number of hidden layers. Within this setting, the proposed initialization was shown to improve accuracy and training stability across both function-fitting and forward PDE tasks. A natural question, however, is whether these gains extend to deeper architectures. To this end, we revisit Burgers’ equation and additionally consider the Allen–Cahn equation (see Appendix D for details), training cPIKANs of increasing depth to examine how network depth influences training stability under both initialization schemes.

In contrast to the previous benchmarks, here we adopt all of the adaptive training methods described in Section 2.3, since both PDEs are trained for a large number of iterations and the Allen–Cahn equation in particular fails to yield accurate results without adaptive strategies [25, 26]. Throughout training, adaptive collocation-point resampling is performed, with hyperparameters $\delta = 1$ and $C = 1$, which is a standard choice in the literature [51]. The pool of collocation points to enforce the PDE is constructed using $N_{\text{pool}} = 400 \times 400$ uniformly distributed samples over the spatiotemporal domain $[0, 1] \times [-1, 1]$. Every $2 \cdot 10^3$ iterations, $N_{\text{pde}} = 2^{12}$ points are resampled for training. Alongside this, we apply the RBA method with hyperparameters $\gamma = 0.999$ and $\eta = 0.01$ as in [26], so that resampling follows the probability density function defined in Eq. (25). For the initial condition, a fixed set of $N_{\text{ic}} = 2^6$ points is used, while RBA weighting is still applied. No collocation points are used to enforce boundary conditions, which are instead incorporated directly into the network architecture following the exact boundary enforcement strategy of [63].

In addition, every 10^3 iterations we apply learning-rate annealing according to Eqs. (19)-(20), with decay parameter $a = 0.9$. Causal training is also employed by partitioning the temporal domain into $M = 32$ segments and using $\epsilon = 1.0$ [23]. In all experiments, Chebyshev-based KANs with polynomial order $D = 5$ are considered, initialized under both the default and the proposed scheme. Models are trained for 10^5 iterations using the Adam optimizer with an initial warm-up phase of 10^3 iterations, reaching a learning rate of 10^{-3} , followed by exponential decay with a factor of 0.9 every $2 \cdot 10^3$ iterations. Architectures with $\{2, 4, 6, 8, 10, 12\}$ hidden layers (depths) and hidden layer sizes (widths) $\{8, 16, 32\}$ are considered. Performance is evaluated in terms of the relative L^2 error with respect to the reference solution, and each experiment is repeated five times with different random seeds.

The results of these experiments are summarized in Figure 4, which presents the relative L^2 error of each network configuration under both initialization schemes for the two studied PDEs. Each column corresponds to a specific network width, with the horizontal axis of each subplot representing the network depth. Solid lines denote the mean relative L^2 error over the five independent runs, and the shaded regions indicate the corresponding standard error. The results highlight the consistent advantage of the proposed initialization across all architectures and both PDEs, often yielding relative L^2 errors lower by several orders of magnitude compared to the default scheme. For the Burgers’ equation, the effect is particularly strik-

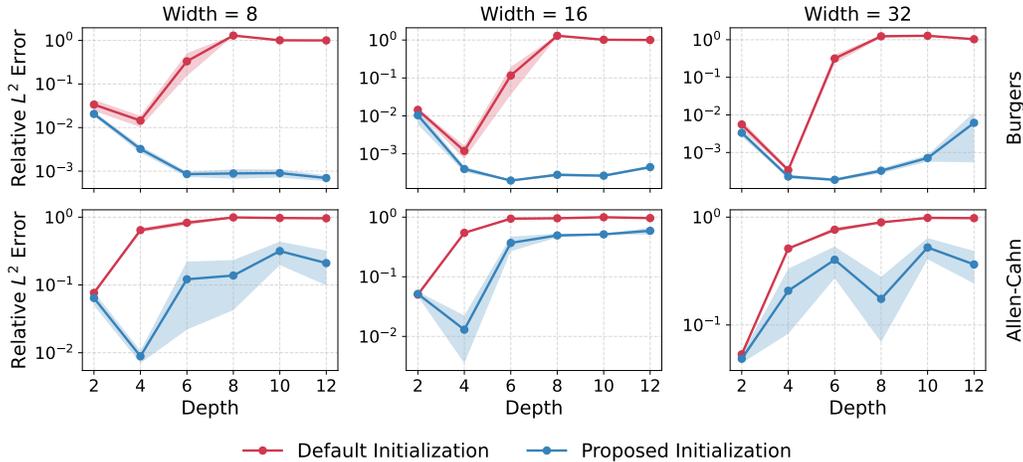


Figure 4: Relative L^2 error across increasing network depths for Burgers’ (top row) and Allen–Cahn (bottom row) equations, under both default and proposed initialization schemes. Each column corresponds to a different network width (8, 16, 32). Solid lines show mean values over five random seeds, while shaded areas represent standard error.

ing: beyond a depth of four hidden layers, models initialized with the default scheme exhibit complete divergence, with relative errors of $\mathcal{O}(1)$, whereas the proposed initialization maintains stable training and achieves errors as low as $\mathcal{O}(10^{-3})$ – $\mathcal{O}(10^{-4})$. Although some degradation in accuracy is observed at larger depths for the widest networks (width 32), no divergence occurs, and the relative errors remain well below those of the default scheme. In contrast, for the Allen–Cahn equation, training instability persists for deeper networks under both initialization schemes. While the proposed initialization consistently improves performance compared to the default, the relative error grows rapidly with depth and reaches $\mathcal{O}(1)$ for the deepest configurations. The lowest errors are typically achieved at shallow depths (i.e., depth 2 for width 32 and depth 4 for the other two cases), after which performance degrades, albeit not monotonically for all cases. Overall, while the proposed scheme substantially improves training stability and accuracy, it does not, by itself, guarantee stable convergence in all cases as network depth increases.

4. Residual-Gated Adaptive KANs

While the proposed Glorot-like initialization improved training stability and accuracy across all benchmarks, its effectiveness in mitigating depth-

related issues proved case-dependent. In particular, for Burgers’ equation, cPIKANs initialized with the proposed scheme remained stable and convergent even at larger depths, whereas for the Allen–Cahn equation similar architectures exhibited divergence beyond a certain number of hidden layers. A similar phenomenon is observed in PINNs: when networks are initialized following the Glorot scheme, training tends to diverge as depth increases, regardless of the specific activation function among those commonly used in practice [23]. However, before attempting to transfer to cPIKANs the remedies that have been proposed for this behavior in PINNs, it is first necessary to determine whether the underlying mechanisms are indeed analogous.

4.1. KAN Derivatives at Initialization

In the case of MLP-based networks – the backbone of PINNs – [23] demonstrated that training divergence with increasing depth originates from the behavior of the network’s derivatives at initialization. To illustrate this, they considered a simplified MLP with scalar input and output, employing the hyperbolic tangent activation $\sigma(x) = \tanh(x)$, and focused on the first-order derivative with respect to the input. At initialization, the network operates in a near-linear regime where $\sigma(x) \approx x$, leading to the following approximation for the first-order derivative of Eq. (10) with respect to the input coordinate x :

$$\frac{\partial u_j^{(l)}(x; \boldsymbol{\theta})}{\partial x} \approx \sum_{i=1}^{d_{l-1}} w_{ji}^{(l)} \frac{\partial u_i^{(l-1)}(x; \boldsymbol{\theta})}{\partial x}. \quad (34)$$

Assuming a network composed of L hidden layers of width d_H , and noting that $\frac{\partial u^{(0)}(x; \boldsymbol{\theta})}{\partial x} = 1$, the derivative of the network output can be recursively expressed as

$$\frac{\partial u(x; \boldsymbol{\theta})}{\partial x} \approx \sum_{i=1}^{d_H} \sum_{n=1}^{d_H} \cdots \sum_{m=1}^{d_H} w_{1i}^{(L+1)} w_{in}^{(L)} \cdots w_{m1}^{(1)}. \quad (35)$$

This result shows that the derivative in Eq. (35) behaves as a deep linear network at initialization and, more importantly, is independent of the input x . This reveals a fundamental limitation in the expressivity of the derivative network and explains why, in the context of PIML – where PDE residuals depend directly on network derivatives – deep MLP-based architectures tend to diverge during training.

Under analogous simplifying assumptions for a KAN-based network, the derivative of Eq. (15) with respect to the input x is given by

$$\frac{\partial u_j^{(l)}(x; \boldsymbol{\theta})}{\partial x} = \sum_{i=1}^{d_{l-1}} \sum_{m=1}^D w_{jim}^{(l)} \frac{\partial B_m}{\partial x} [u_i^{(l-1)}(x; \boldsymbol{\theta})] \frac{\partial u_i^{(l-1)}(x; \boldsymbol{\theta})}{\partial x}. \quad (36)$$

For Chebyshev-based KANs, where the basis functions are given by Eq. (16), it can be shown (see Appendix A.2 for the detailed derivation) that, in the linear regime,

$$\frac{\partial u_j^{(l)}(x; \boldsymbol{\theta})}{\partial x} \approx \sum_{i=1}^{d_{l-1}} \tilde{w}_{ji}^{(l)} \frac{\partial u_i^{(l-1)}(x; \boldsymbol{\theta})}{\partial x}, \quad (37)$$

where

$$\tilde{w}_{ji}^{(l)} = \sum_{m \text{ odd}}^D m w_{jim}^{(l)} (-1)^{\frac{m-1}{2}}. \quad (38)$$

Equation (37) is formally equivalent to Eq. (34), indicating that, within the linear regime, the first-order derivative of a Chebyshev-based KAN behaves analogously to that of an MLP at initialization. This correspondence indicates that the observed training instabilities in deep cPIKANs arise from a similar mechanism identified in deep PINNs.

4.2. Proposed Architecture

To address these instabilities in PINNs, [23] introduced the PirateNet architecture (see Appendix E for details). PirateNets incorporate several architectural components known to improve accuracy, such as random Fourier feature (RFF) embeddings [64] and a physics-informed initialization of the final network layer. However, the key idea for resolving the depth-related issue is the introduction of an adaptive skip connection. This mechanism introduces a learnable gating parameter, α , which dynamically modulates the network’s effective depth during training, thereby stabilizing optimization and enabling convergence for deeper architectures. Inspired by this approach, we introduce *Residual-Gated Adaptive Kolmogorov–Arnold Networks* (RGA KANs), the architecture of which is illustrated in Figure 5.

For a single input sample $\mathbf{x} \in \mathbb{R}^{d_I}$, where d_I denotes the number of coordinates (including a possible temporal coordinate), periodic boundary conditions – when present – are enforced directly through the embedding layer.

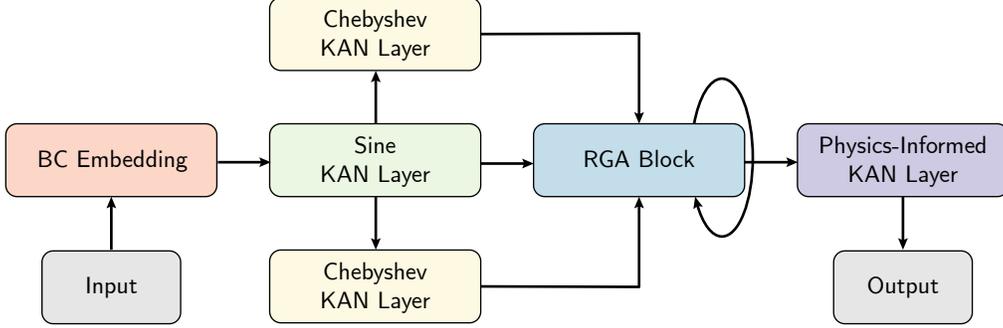


Figure 5: Schematic of the proposed RGA KAN architecture. Periodic boundary conditions, when present, are enforced directly through the BC Embedding layer. The embedded inputs are then passed through a sine-based KAN layer, whose outputs are split into three branches: two feeding Chebyshev-based KAN layers and one entering the first RGA block. Within each RGA block, the three signals are combined through gating operators and routed through adaptive skip connections, which dynamically modulate the effective network depth during training. Multiple RGA blocks can be stacked sequentially. The final output is produced by a physics-informed KAN layer, which incorporates prior information from the initial condition(s) when available.

Specifically, if periodic boundary conditions apply to the i -th coordinate, the embedding is defined as

$$\text{Emb}(x_i) = \begin{bmatrix} \cos(\Omega_i x_i) \\ \sin(\Omega_i x_i) \end{bmatrix} \in \mathbb{R}^2, \quad \Omega_i = \frac{2\pi}{L_i}, \quad (39)$$

where L_i is the length of the i -th coordinate's domain. In most cases considered in this work, where $x_i \in [-1, 1]$, we have $\Omega_i = \pi$. After embedding, the input is mapped to $\tilde{\mathbf{x}} \in \mathbb{R}^{\tilde{d}_1}$, where \tilde{d}_1 is the new number of effective input coordinates.

The embedded input then passes through a sine-based KAN layer, whose output, $\mathbf{s} \in \mathbb{R}^{d_H}$, is computed as

$$s_j = \sum_{i=1}^{\tilde{d}_1} \sum_{m=1}^{D_s} b_{jim}^s B_m^s(\tilde{x}_i) + c_j^s, \quad (40)$$

where c_j^s is a bias term and $B_m^s(\cdot)$, with $m = 1, \dots, D_s$, are sine-based basis functions defined by

$$B_m^s(x) = \frac{\sin(\omega_m x + p_m) - \mu(\omega_m, p_m)}{\sigma(\omega_m, p_m)}. \quad (41)$$

Here, $\mu(\omega_m, p_m)$ and $\sigma(\omega_m, p_m)$ denote the mean and standard deviation of $\sin(\omega_m x + p_m)$, given by

$$\mu(\omega_m, p_m) = \exp\left(-\frac{\omega_m^2}{2}\right) \sin(p_m), \quad (42)$$

$$\sigma(\omega_m, p_m) = \sqrt{\frac{1}{2} - \frac{1}{2} \exp(-2\omega_m^2) \cos(2p_m) - \mu(\omega_m, p_m)^2}. \quad (43)$$

This basis function design is inspired by the ActLayer [56] and plays a similar role to RFF embeddings used in the PirateNet architecture. In preliminary experiments, RFF embeddings were found to degrade performance in our setting, whereas sine-based KAN layers preserved the benefits of trigonometric features, which have been shown to be particularly effective in many PDE problems [56]. In addition to the trainable coefficients b_{jim}^s (initialized using the Glorot-like scheme proposed in this work), we also introduce trainable phase parameters p_m (initialized at zero) and frequency parameters ω_m (initially sampled from a standard normal distribution), as in ActLayers.

At this stage, drawing inspiration from the Modified MLP architecture [20], we define two gates using Chebyshev-based KAN layers:

$$U_j = \sum_{i=1}^{d_H} \sum_{m=1}^D b_{jim}^u B_m(s_i) + c_j^u, \quad V_j = \sum_{i=1}^{d_H} \sum_{m=1}^D b_{jim}^v B_m(s_i) + c_j^v, \quad (44)$$

where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{d_H}$. These gate outputs, together with the outputs of the sine-based KAN layer, form the inputs to the first RGA block. Considering a total of N such RGA blocks and denoting the input to the l -th block by $\mathbf{x}^{(l)}$, with $l = 1, \dots, N$ and $\mathbf{x}^{(1)} = \mathbf{s}$, the forward pass through each block is defined recursively as follows:

$$f_j^{(l)} = \sum_{i=1}^{d_H} \sum_{m=1}^D b_{jim}^{(l)} B_m(x_i^{(l)}) + c_j^{(l)}, \quad (45)$$

$$g_j^{(l)} = f_j^{(l)} U_j + (1 - f_j^{(l)}) V_j, \quad (46)$$

$$z_j^{(l)} = \beta g_j^{(l)} + (1 - \beta) x_j^{(l)}, \quad (47)$$

$$\tilde{f}_j^{(l)} = \sum_{i=1}^{d_H} \sum_{m=1}^D \tilde{b}_{jim}^{(l)} B_m \left(z_i^{(l)} \right) + \tilde{c}_j^{(l)}, \quad (48)$$

$$\tilde{g}_j^{(l)} = \tilde{f}_j^{(l)} U_j + \left(1 - \tilde{f}_j^{(l)} \right) V_j, \quad (49)$$

$$x_j^{(l+1)} = \alpha \tilde{g}_j^{(l)} + (1 - \alpha) x_j^{(l)}, \quad (50)$$

where the dimension of all intermediate and final outputs is d_H . All bias terms appearing in Eqs. (44), (45) and (48) are initialized to zero, while the basis function coefficients are initialized using the proposed Glorot-like scheme of Eq. (33).

The skip connection governed by the parameter α is typically initialized either at zero, effectively suppressing the contribution of each RGA block at initialization, or at unity, enabling the network to start at its full intended depth while still allowing the effect of each block to be adaptively modulated during training. While PirateNets employ a three-layer block with a single adaptive skip connection, we adopt a two-layer design and introduce an additional adaptive parameter, β , after the first layer. When β is initialized at 1, the block behaves analogously to the original PirateNet block at initialization, whereas $\beta = 0$ corresponds to introducing an adaptive skip after each layer. Here, α controls the activation of the entire block, while β affects only the first layer. This design was preferred over a direct three-layer port, as it proved more modular and yielded better results in preliminary tests. In terms of effective depth, an RGA KAN with N blocks is equivalent to a conventional KAN with $L = 2N$ hidden layers.

The output of the final RGA block, $\mathbf{x}^{(N+1)} \in \mathbb{R}^{d_H}$, is mapped to the network output through a final Chebyshev-based KAN layer, defined as

$$u_j = \sum_{i=1}^{d_H} \sum_{m=1}^D b_{jim}^o B_m \left(x_i^{(N+1)} \right), \quad (51)$$

where no bias term is included. In the absence of additional information, the coefficients b_{jim}^o are initialized using the same Glorot-like scheme proposed in Section 3.1. However, if the PDE problem under consideration is equipped with an initial condition, this layer is instead physics-informed initialized. Specifically, the weights are chosen such that the network output approximates the initial condition at $t = 0$ as accurately as possible over the entire temporal domain.

For the purposes of this physics-informed initialization, we re-index the pair (i, m) into a single index $p = 1, \dots, d_{\text{H}}D$, yielding the equivalent form

$$u_j = \sum_{p=1}^{d_{\text{H}}D} \tilde{b}_{jp}^o \tilde{B}_p, \quad (52)$$

where \tilde{B}_p denotes the activation of the m -th basis function evaluated at the i -th component of $\mathbf{x}^{(N+1)}$, with the composite index $p \leftrightarrow (i, m)$. The physics-informed initialization amounts to solving the least-squares problem

$$\tilde{\mathbf{b}}^o = \arg \min_{\mathbf{b}} \left\| \mathbf{y}_0 - \tilde{\mathbf{B}} \mathbf{b} \right\|_2^2, \quad (53)$$

where \mathbf{y}_0 contains the initial-condition target values and $\tilde{\mathbf{B}}$ is the matrix of basis activations evaluated on the outputs of the final RGA block when the original inputs are set to the collocation points enforcing the initial condition. The vector $\tilde{\mathbf{b}}^o$ contains the optimal coefficients obtained by the least-squares fit and can be re-indexed back to the original (i, m) indexing to yield the coefficients b_{jim}^o of Eq. (51). Note that, although here we focus on standard PDE benchmarks with initial conditions, this formulation can in principle incorporate arbitrary external data through \mathbf{y}_0 , such as experimental measurements [23].

If non-periodic boundary conditions are present, they can be directly enforced after this final physics-informed layer by multiplying the network output by suitable boundary-shaping functions, thereby ensuring that the solution satisfies these constraints exactly [63]. Based on the above architectural components, for an RGA KAN model with d_{O} -dimensional output, the total number of trainable parameters can be explicitly determined as follows:

$$\begin{aligned} |\boldsymbol{\theta}| &= \underbrace{d_{\text{H}} \left(\tilde{d}_{\text{I}} D_s + 1 \right) + 2D_s}_{\text{Sine KAN Layer}} + \overbrace{2d_{\text{H}} (d_{\text{H}}D + 1)}^{\text{Chebyshev KAN Gates}} \\ &+ \underbrace{N [2d_{\text{H}} (d_{\text{H}}D + 1) + 2]}_{\text{RGA Blocks}} + \overbrace{d_{\text{O}} d_{\text{H}}D}_{\text{Output Layer}} \\ &= 2d_{\text{H}} (d_{\text{H}}D + 1) (N + 1) + 2N + 2D_s + d_{\text{H}} \left(\tilde{d}_{\text{I}} D_s + d_{\text{O}}D + 1 \right). \end{aligned} \quad (54)$$

Using this architecture, we repeat the experiments of Section 3.3 for the Allen–Cahn equation, where cPIKANs diverged with increasing depth. To ensure a fair comparison with the previous cPIKAN results, we employ identical parameter settings and use the same random seeds. Since the superiority of the proposed Glorot-like initialization scheme has already been established, we focus exclusively on this initialization strategy here. For parameters without a direct analogue in cPIKANs, we set $D_s = 5$ for the sine-based KAN layer and initialize both β and α to zero in each RGA block. The corresponding results are depicted in Figure 6, where the reported number of hidden layers corresponds to twice the number of RGA blocks for the RGA KAN architecture.

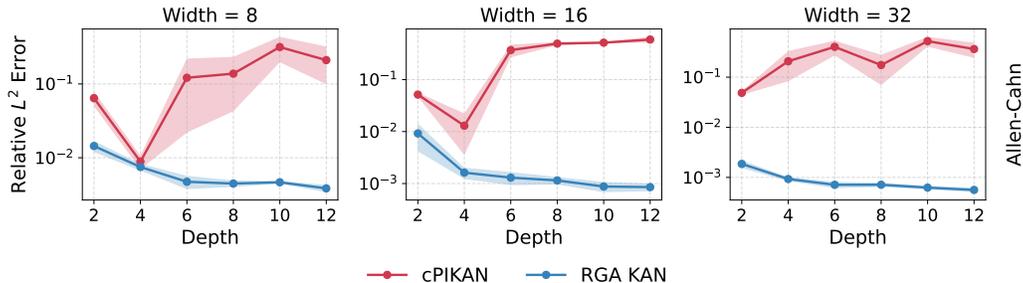


Figure 6: Relative L^2 error across increasing network depths for the Allen–Cahn equation, comparing RGA KANs and cPIKANs. Each column corresponds to a different network width (8, 16, 32) and the number of hidden layers for RGA KANs equals twice the number of RGA blocks. All results are averaged over five random seeds using the proposed Glorot-like initialization scheme. Solid lines indicate mean values and shaded areas denote standard error.

The results demonstrate a significant improvement in stability and accuracy compared to the baseline cPIKANs. Across all widths and depths considered, RGA KANs maintain low relative L^2 errors without exhibiting divergence, even for the deepest networks. Moreover, a favorable scaling effect is observed: as the network width increases, the relative error decreases consistently, with the widest configuration (width 32) yielding the best results across depths. More importantly, within each width setting, increasing the number of RGA blocks either preserves performance (plateau behavior) or further reduces the error – a behavior that mirrors the improvements reported for PirateNets over regular MLPs [23].

As a closing remark, we note that we opted for $\alpha = 0$ and $\beta = 0$ at initialization, effectively initializing the network in a state that resembles

a single-layer model and progressively increasing its effective depth during training. This choice follows a similar rationale to that of PirateNets, where gradual deepening contributes to stable optimization. However, in practice, the residual gates \mathbf{U} and \mathbf{V} can already act as stabilizing components, often preventing divergence even when the network is initialized at full depth. To investigate this further, in the next section – where we present more extensive benchmarks across several PDEs – we include ablation studies on the initialization values of α and β to examine their effect on training dynamics and performance for each problem.

4.3. The Lens of Information Bottleneck Theory

To better understand why the RGA KAN architecture not only achieves superior accuracy compared to baseline cPIKANs but also avoids performance degradation with increasing depth, we turn to Information Bottleneck (IB) theory to analyze the training dynamics of both architecture types. In supervised learning, neural networks aim to reproduce target outputs by progressively forming compressed internal representations of the inputs through their layer activations. According to IB theory, an optimally trained model preserves only the information relevant for reproducing the output while discarding irrelevant input details, effectively forming an “information bottleneck” [65]. This learning process typically unfolds in two distinct phases, namely fitting and diffusion, separated by a phase transition [66, 67, 68]; it is during the diffusion phase that the network develops its generalization capabilities. IB theory has also been applied to analyze the training dynamics of neural networks within the PIML framework [48, 26, 35, 42], and has even been extended to incorporate a third phase within this context, termed diffusion equilibrium [48] or total diffusion [35, 42].

To detect phase transitions during training, two key indicators are typically monitored: the relative L^2 error and the batch-wise signal-to-noise ratio (SNR), defined as

$$\text{SNR} = \frac{\|\mathbb{E}[\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{batch}}(\boldsymbol{\theta})]\|_2}{\left\| \sqrt{\mathbb{E}[(\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{batch}}(\boldsymbol{\theta}) - \mathbb{E}[\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{batch}}(\boldsymbol{\theta})])^2]} \right\|_2}, \quad (55)$$

where $\mathcal{L}_{\text{batch}}(\boldsymbol{\theta})$ denotes the loss of Eq. (6) evaluated over a single batch of collocation points, and expectations are taken across all non-overlapping batches. Intuitively, the SNR measures the ratio between the mean gradient

norm (signal) and its standard deviation (noise), reflecting the clarity of the learning signal during optimization.

In addition, recent studies have shown that the geometric complexity of the network can provide further insight into its training dynamics [69, 42]. This metric, defined via the discrete Dirichlet energy, is given by

$$\text{Complexity} = \frac{1}{N} \sum_{i=1}^N \|\nabla_{t,\mathbf{x}} u(t^i, \mathbf{x}^i; \boldsymbol{\theta})\|_F^2, \quad (56)$$

where $\{(t^i, \mathbf{x}^i)\}_{i=1}^N = \{(t_{\text{pde}}^i, \mathbf{x}_{\text{pde}}^i)\}_{i=1}^{N_{\text{pde}}} \cup \{(0, \mathbf{x}_{\text{ic}}^i)\}_{i=1}^{N_{\text{ic}}}$ denotes the complete set of $N = N_{\text{pde}} + N_{\text{ic}}$ collocation points, and $\|\cdot\|_F$ is the Frobenius norm. Note that boundary condition points are not explicitly included here, as they are already enforced through the network architecture; otherwise, they would contribute to this set in the same manner.

To investigate the differences in training dynamics between cPIKANs and RGA KANs, we repeat the experiments for the Allen–Cahn equation using networks with 12 hidden layers (equivalently, 6 RGA blocks), and widths 8,

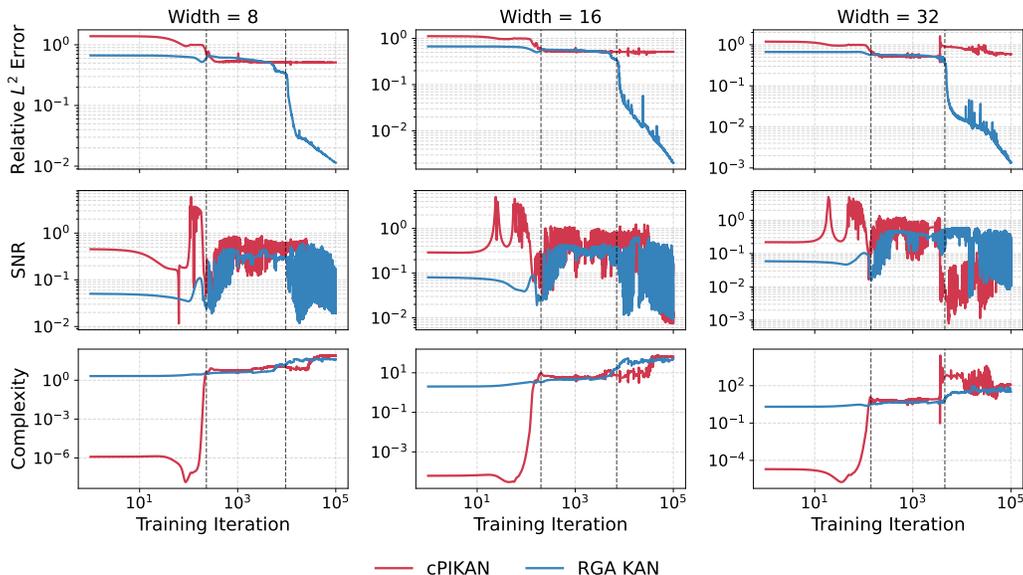


Figure 7: Training dynamics of cPIKAN and RGA KAN architectures of different widths in terms of relative L^2 error (top row), SNR (middle row) and geometric complexity (bottom row). Dashed vertical lines indicate the transitions between training phases (fitting, diffusion and diffusion equilibrium) for the RGA KAN models.

16 and 32. All hyperparameters for RGA KANs are kept identical to those in Section 4.2, while cPIKANs use the same settings as in Section 3.3. During each training iteration, we record the relative L^2 error as well as the SNR and geometric complexity defined in Eqs. (55) and (56), respectively. Among the five independently trained instances per architecture (each initialized with a different random seed), Figure 7 reports the results corresponding, for each width, to the run with the highest final relative L^2 error for the RGA KAN architecture.

The training dynamics of the RGA KAN architecture, analyzed through the lens of IB theory as studied in the context of PIML [48, 35, 42], reveal a clear progression through all three learning phases. During the initial fitting phase, which spans roughly the first 200 training iterations, the relative L^2 error remains nearly constant, while the geometric complexity increases steadily. Simultaneously, the SNR exhibits a brief oscillatory pattern before beginning to decline. In this stage, the network primarily memorizes the training data without significant generalization. A subsequent transition marks the onset of the diffusion phase, characterized by a steadily increasing and fluctuating SNR, a slight decrease in relative L^2 error and a continued increase in geometric complexity. This phase corresponds to an exploratory stage in which the model identifies more effective learning directions and begins to generalize. Finally, the network enters the diffusion equilibrium phase, during which the SNR reaches a stable, though still oscillatory, plateau, the geometric complexity continues to rise and then also plateaus, and the relative L^2 error drops sharply, indicating a rapid improvement in generalization and predictive accuracy. Remarkably, this qualitative behavior is consistent across all three network widths. The only systematic difference is the timing of the phase transitions, which occur earlier as the model width increases. This observation aligns with our previous findings: when combined with the proposed initialization, increasing the capacity of the RGA KAN architecture does not lead to divergence but instead improves model accuracy.

In contrast, the behavior of the cPIKAN models differs substantially. While they exhibit a fitting phase similar to that of the RGA KAN, the increase in geometric complexity is far more abrupt and several orders of magnitude larger – an effect previously reported for cPIKANs in [42]. After transitioning to the diffusion phase, these models never reach the diffusion equilibrium phase. The geometric complexity plateaus prematurely, the SNR exhibits strong oscillations without converging to a stable plateau, and the relative L^2 error stagnates. As a result, the cPIKAN models fail to generalize,

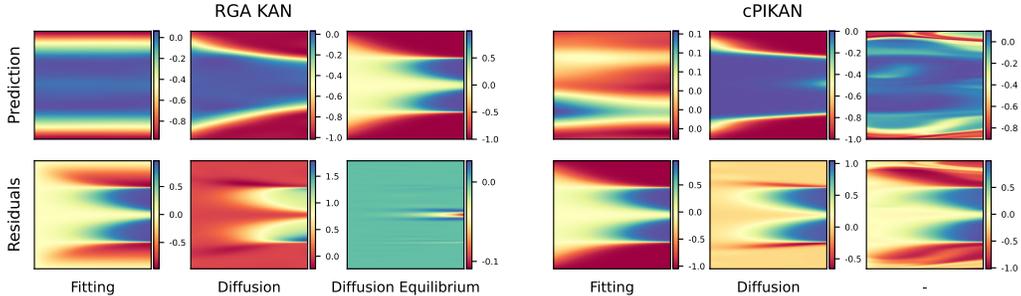


Figure 8: Evolution of the model predictions (top row) and residuals (bottom row) across the three IB training phases for RGA KAN (left) and cPIKAN (right) architectures, using a width-16 configuration. For the RGA KAN model, the predictions become progressively more structured and closely match the reference solution as training proceeds, while the residuals approach noise during the diffusion equilibrium phase. In contrast, the cPIKAN model fails to undergo a clear second transition, resulting in residuals that remain semi-ordered and predictions that deviate significantly from the reference solution.

explaining the poor performance and divergence observed at large depths for the Allen–Cahn equation.

These results provide strong evidence, from the perspective of IB theory, for why the RGA KAN architecture maintains stability and achieves superior accuracy where cPIKANs fail. An additional layer of insight can be gained by examining the evolution of the models’ predictions and residuals (the difference between the reference solution and the model output) across the identified phases. This is illustrated in Figure 8, where we show results for the width 16 configuration for both architectures (RGA KAN on the left, cPIKAN on the right). As expected, during the fitting phase, when the model has not yet learned to generalize, the predictions are overly simplistic and the residuals structured, effectively mirroring the inverse of the reference solution. During the diffusion phase, the predictions gradually become more structured and the residuals more disordered. However, while the RGA KAN undergoes a clear second transition, leading to predictions closely matching the reference solution and residuals steadily approaching noise, the cPIKAN remains stuck in a semi-ordered state, never fully generalizing.

5. Experimental Results

Having established that RGA KANs, when initialized with the proposed Glorot-like scheme, remain stable as depth increases and avoid the divergence

observed in baseline cPIKANs – both empirically (Section 4.2) and through the lens of IB theory (Section 4.3) – we now turn to a series of forward-PDE benchmarks and ablation studies. In all experiments, we use the exact same hyperparameter and optimization settings as those used in Section 3.3 (number of training iterations, number of collocation points, adaptive training hyperparameters, etc.). These settings are applied uniformly across all architectures considered. The goal of this section is not to perform extensive hyperparameter sweeps to obtain state-of-the-art performance for each PDE individually, but rather to demonstrate that RGA KANs already achieve strong results without any task-specific tuning, using a single, generic configuration.

For the main experiments, we use RGA KANs of width 16 and depth 12 (corresponding to $N = 6$ RGA blocks). To provide a fair comparison, we also evaluate baseline cPIKANs initialized with the proposed Glorot-like scheme and PirateNets, which represent the current state of the art MLP-based architecture on several PDE benchmarks [23]. To match parameter counts across architectures at the same depth, we adjust widths accordingly: for cPIKANs we use width 18 and for PirateNets we use width 36. All experiments are repeated with three random seeds for statistical significance. For RGA KANs, we additionally investigate four variants corresponding to different initializations of the adaptive skip parameters, with $(\alpha, \beta) \in \{0, 1\} \times \{0, 1\}$. After reporting the benchmark results for each architecture, we identify the (α, β) configuration that achieves the lowest error and use it as the reference in the subsequent ablation studies.

The ablation experiments aim to quantify the contribution of each adaptive training component to the overall performance of RGA KANs. To this end, we perform the following sequence: (i) train with RBA alone, disabling all other adaptive techniques; (ii) disable RBA while keeping all other techniques enabled; (iii) disable RBA and RAD while keeping causal training and learning-rate annealing; (iv) disable RBA and causal training while keeping RAD and learning-rate annealing; and finally (v) disable RBA and learning-rate annealing while keeping RAD and causal training. Each configuration is again trained with three different random seeds. The following subsections present the results for each PDE benchmark.

5.1. Allen–Cahn Equation

The first benchmark considered is the Allen–Cahn equation (see Appendix D for details), which has already served as the testbed in previous

sections. We begin by examining the effect of different (α, β) initializations on RGA KAN performance. As shown in Table 1, initializing with $\alpha = 1$ and $\beta = 0$ yields the best results, closely followed by $\alpha = 1, \beta = 1$. From the three trained instances under this optimal setting, we retain the one achieving the lowest relative L^2 error (3.96×10^{-4}) for visualization. Figure 9 compares the model prediction against the reference solution and shows the absolute error field, which remains at most $\mathcal{O}(10^{-3})$, demonstrating a good agreement between the two.

We next compare RGA KANs with baseline cPIKAN and PirateNet architectures under similar total parameter numbers. Table 2 summarizes the results. While cPIKAN and PirateNet are more time-efficient per iteration, RGA KAN achieves a substantially lower relative L^2 error, outperforming PirateNet by roughly an order of magnitude. As expected, the cPIKAN

| Configuration | Relative L^2 Error | Final Loss |
|---|--|--|
| $\alpha = 0, \beta = 0$ | $(1.62 \pm 0.20) \times 10^{-3}$ | $(2.99 \pm 0.37) \times 10^{-6}$ |
| $\alpha = 1, \beta = 0$ | $(4.46 \pm 0.25) \times 10^{-4}$ | $(5.27 \pm 0.51) \times 10^{-7}$ |
| $\alpha = 0, \beta = 1$ | $(1.39 \pm 0.43) \times 10^{-3}$ | $(1.89 \pm 0.54) \times 10^{-6}$ |
| $\alpha = 1, \beta = 1$ | $(4.99 \pm 0.84) \times 10^{-4}$ | $(2.78 \pm 0.49) \times 10^{-7}$ |

Table 1: Results for different RGA KAN (α, β) initializations on the Allen–Cahn equation. Reported values are mean \pm standard error over three seeds. The best performing configuration in terms of relative L^2 error is indicated in bold.

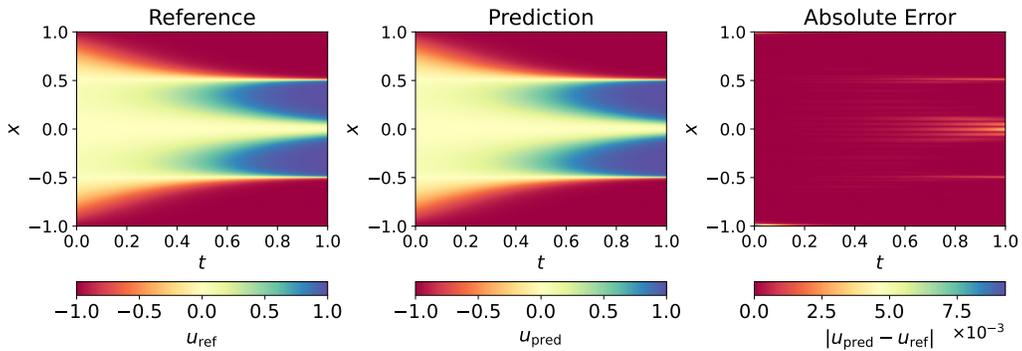


Figure 9: Reference solution (left), RGA KAN prediction (middle) and absolute error (right) for the Allen–Cahn equation, shown for the random seed corresponding to the best-performing model instance.

| Architecture | Parameters | Relative L^2 Error | Time / Iter. |
|----------------|---------------|--|----------------|
| cPIKAN | 18,397 | $(5.21 \pm 0.01) \times 10^{-1}$ | 3.44 ms |
| PirateNet | 19,246 | $(2.46 \pm 1.45) \times 10^{-3}$ | 3.61 ms |
| RGA KAN | 18,502 | $(4.46 \pm 0.25) \times 10^{-4}$ | 5.51 ms |

Table 2: Performance comparison on the Allen–Cahn equation across different architectures. Reported values are mean \pm standard error over three random seeds. The RGA KAN row uses the best (α, β) initialization from Table 1. The best performing architecture in terms of relative L^2 error is indicated in bold.

models diverge, reflecting the absence of a transition to the diffusion equilibrium phase observed in Section 4.3. PirateNets exhibit stable training but are consistently outperformed by RGA KANs in terms of accuracy. Notably, all RGA KAN configurations in Table 1 outperform PirateNet, indicating robustness with respect to the initial choice of (α, β) . Regarding the required training time per iteration, the RGA KAN architecture is slower than the other two, which is consistent across all benchmarks. This is expected, as cPIKANs do not include additional gating operations, while PirateNets rely on MLPs rather than KANs, resulting in faster iterations.

Finally, we quantify the contribution of the adaptive training components through ablation studies (Table 3). Using only RBA leads to the largest errors and high variability, suggesting strong sensitivity to weight initialization. Disabling RBA while retaining the other components degrades performance by less than one order of magnitude, indicating that the remaining adaptive

| RBA | RAD | Causal | LR Annealing | Relative L^2 Error |
|-----|-----|--------|--------------|----------------------------------|
| ✓ | ✓ | ✓ | ✓ | $(4.46 \pm 0.25) \times 10^{-4}$ |
| ✓ | ✗ | ✗ | ✗ | $(2.88 \pm 2.85) \times 10^{-1}$ |
| ✗ | ✓ | ✓ | ✓ | $(8.00 \pm 2.92) \times 10^{-4}$ |
| ✗ | ✗ | ✓ | ✓ | $(1.43 \pm 0.40) \times 10^{-3}$ |
| ✗ | ✓ | ✗ | ✓ | $(3.66 \pm 1.03) \times 10^{-3}$ |
| ✗ | ✓ | ✓ | ✗ | $(9.26 \pm 1.84) \times 10^{-3}$ |

Table 3: Ablation study on adaptive training components for the Allen–Cahn equation using the best (α, β) initialization from Table 1. Each row corresponds to a different combination of enabled (✓) or disabled (✗) components. Reported values are mean \pm standard error over three seeds.

techniques are sufficient to preserve stability. Among them, learning-rate annealing has the largest individual impact, followed by causal training, while RAD plays a less dominant role for this PDE.

5.2. Burgers' Equation

We next consider Burgers' equation (see Appendix D for details) and follow the same experimental procedure. As summarized in Table 4, the best-performing configuration corresponds to $\alpha = 0$ and $\beta = 1$, closely followed by $\alpha = 1, \beta = 1$. Although the difference in mean relative L^2 error between these two configurations is small, the former exhibits a substantially lower standard error, indicating more robustness. We therefore retain $\alpha = 0, \beta = 1$ for the subsequent experiments. Figure 10 shows the predicted solution for the seed with the lowest relative L^2 error (2.46×10^{-4}), together with the

| Configuration | Relative L^2 Error | Final Loss |
|---|--|--|
| $\alpha = 0, \beta = 0$ | $(4.57 \pm 1.25) \times 10^{-4}$ | $(1.13 \pm 0.42) \times 10^{-4}$ |
| $\alpha = 1, \beta = 0$ | $(7.34 \pm 2.07) \times 10^{-4}$ | $(1.69 \pm 0.68) \times 10^{-4}$ |
| $\alpha = 0, \beta = 1$ | $(3.06 \pm 0.31) \times 10^{-4}$ | $(4.14 \pm 0.48) \times 10^{-5}$ |
| $\alpha = 1, \beta = 1$ | $(3.07 \pm 0.84) \times 10^{-4}$ | $(2.87 \pm 1.12) \times 10^{-5}$ |

Table 4: Results for different RGA KAN (α, β) initializations on Burgers' equation. Reported values are mean \pm standard error over three seeds. The best performing configuration in terms of relative L^2 error is indicated in bold.

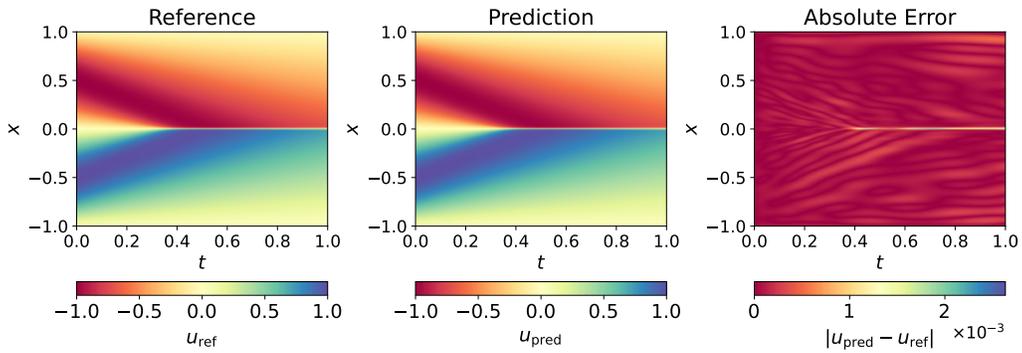


Figure 10: Reference solution (left), RGA KAN prediction (middle) and absolute error (right) for Burgers' equation, shown for the random seed corresponding to the best-performing model instance.

reference solution and the absolute error field, which retains a maximum value of $\mathcal{O}(10^{-3})$.

We then compare RGA KAN with baseline cPIKAN and PirateNet architectures at matched parameter counts. Table 5 reports the corresponding results. Again, the training times per iteration are lower for the baseline architectures, and RGA KAN achieves the lowest relative L^2 error, approximately halving the error of cPIKAN and PirateNet. Interestingly, unlike the Allen–Cahn case, the cPIKAN model initialized using our proposed Glorot-like scheme does not diverge here and even achieves slightly better performance than PirateNet. However, given the overlapping standard errors, the two are essentially comparable. RGA KAN exhibits both the lowest mean error and the smallest variability across seeds.

Finally, we examine the contribution of each adaptive training technique (Table 6). Enabling only RBA leads to a sharp increase in relative error by nearly two orders of magnitude, though training remains stable. Strikingly,

| Architecture | Parameters | Relative L^2 Error | Time / Iter. |
|----------------|---------------|--|----------------|
| cPIKAN | 18,307 | $(5.13 \pm 1.18) \times 10^{-4}$ | 3.52 ms |
| PirateNet | 19,228 | $(5.37 \pm 1.32) \times 10^{-4}$ | 3.88 ms |
| RGA KAN | 18,422 | $(3.06 \pm 0.31) \times 10^{-4}$ | 5.72 ms |

Table 5: Performance comparison on Burgers’ equation across different architectures. Reported values are mean \pm standard error over three random seeds. The RGA KAN row uses the best (α, β) initialization from Table 4. The best performing architecture in terms of relative L^2 error is indicated in bold.

| RBA | RAD | Causal | LR Annealing | Relative L^2 Error |
|-----|-----|--------|--------------|----------------------------------|
| ✓ | ✓ | ✓ | ✓ | $(3.06 \pm 0.31) \times 10^{-4}$ |
| ✓ | ✗ | ✗ | ✗ | $(1.06 \pm 0.39) \times 10^{-2}$ |
| ✗ | ✓ | ✓ | ✓ | $(2.50 \pm 0.62) \times 10^{-4}$ |
| ✗ | ✗ | ✓ | ✓ | $(4.08 \pm 1.95) \times 10^{-3}$ |
| ✗ | ✓ | ✗ | ✓ | $(6.47 \pm 1.16) \times 10^{-4}$ |
| ✗ | ✓ | ✓ | ✗ | $(3.34 \pm 0.64) \times 10^{-4}$ |

Table 6: Ablation study on adaptive training components for Burgers’ equation using the best (α, β) initialization from Table 4. Each row corresponds to a different combination of enabled (✓) or disabled (✗) components. Reported values are mean \pm standard error over three seeds.

when RBA is disabled but the other adaptive techniques are retained, the error actually drops slightly below the fully adaptive configuration, indicating that RBA may act as a mild hindrance in this specific setting. Among the remaining techniques, RAD resampling has the largest individual impact, followed by causal training and learning-rate annealing. This pattern contrasts with the Allen–Cahn case, highlighting that the relative importance of adaptive components can depend strongly on the PDE at hand.

5.3. Korteweg–De Vries Equation

We then turn to the Korteweg–De Vries equation (see Appendix D for details). As summarized in Table 7, the best-performing configuration once again corresponds to $\alpha = 0$, $\beta = 1$. Using the best-performing seed, with a final relative L^2 error of 3.21×10^{-3} , we plot the predicted solution alongside the reference and the absolute error field (Figure 11).

| Configuration | Relative L^2 Error | Final Loss |
|---|--|--|
| $\alpha = 0, \beta = 0$ | $(4.73 \pm 0.61) \times 10^{-3}$ | $(1.58 \pm 0.44) \times 10^{-3}$ |
| $\alpha = 1, \beta = 0$ | $(4.45 \pm 0.45) \times 10^{-3}$ | $(1.40 \pm 0.26) \times 10^{-3}$ |
| $\alpha = 0, \beta = 1$ | $(3.87 \pm 0.52) \times 10^{-3}$ | $(8.31 \pm 0.76) \times 10^{-4}$ |
| $\alpha = 1, \beta = 1$ | $(7.84 \pm 0.68) \times 10^{-3}$ | $(1.42 \pm 0.33) \times 10^{-3}$ |

Table 7: Results for different RGA KAN (α, β) initializations on the Korteweg–De Vries equation. Reported values are mean \pm standard error over three seeds. The best performing configuration in terms of relative L^2 error is indicated in bold.

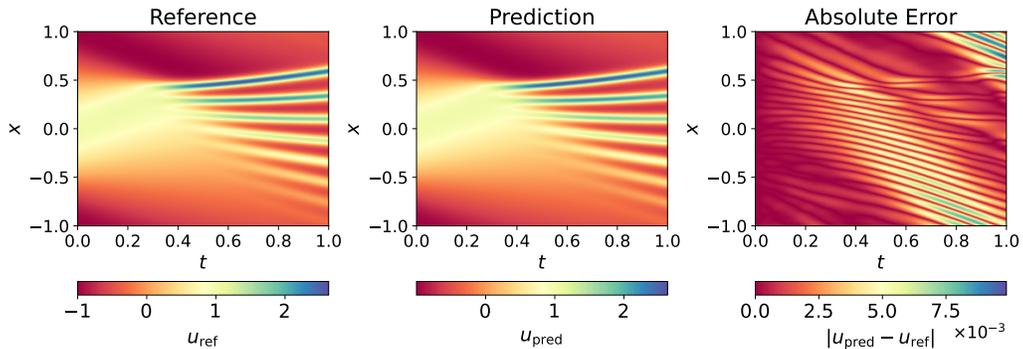


Figure 11: Reference solution (left), RGA KAN prediction (middle) and absolute error (right) for the Korteweg–De Vries equation, shown for the random seed corresponding to the best-performing model instance.

Table 8 presents the comparison between architectures. Interestingly, PirateNet diverges in this setting, with a high relative L^2 error of approximately 7.7×10^{-1} and very low standard error, indicating consistent failure across seeds. This stands in contrast to the results reported in [23], where PirateNet remained stable at similar depths but larger widths, hinting at a sensitivity to model capacity or hyperparameter choices. The cPIKAN model, while also performing poorly, achieves a lower error than PirateNet, suggesting that the proposed Glorot-like initialization once again helps stabilize training. RGA KANs, by comparison, achieve an error nearly two orders of magnitude lower than both baselines.

The ablation results are summarized in Table 9. Retaining only RBA leads to significant performance degradation and divergence, as reflected by an increase of nearly two orders of magnitude in error. Conversely, when RBA is disabled and the other adaptive techniques remain active, the model preserves stable training with only a moderate error increase compared to

| Architecture | Parameters | Relative L^2 Error | Time / Iter. |
|----------------|---------------|--|----------------|
| cPIKAN | 18,397 | $(1.16 \pm 0.03) \times 10^{-1}$ | 4.52 ms |
| PirateNet | 19,246 | $(7.73 \pm 0.10) \times 10^{-1}$ | 4.88 ms |
| RGA KAN | 18,502 | $(3.87 \pm 0.52) \times 10^{-3}$ | 7.44 ms |

Table 8: Performance comparison on the Korteweg–De Vries equation across different architectures. Reported values are mean \pm standard error over three random seeds. The RGA KAN row uses the best (α, β) initialization from Table 7. The best performing architecture in terms of relative L^2 error is indicated in bold.

| RBA | RAD | Causal | LR Annealing | Relative L^2 Error |
|-----|-----|--------|--------------|----------------------------------|
| ✓ | ✓ | ✓ | ✓ | $(3.87 \pm 0.52) \times 10^{-3}$ |
| ✓ | ✗ | ✗ | ✗ | $(5.76 \pm 1.31) \times 10^{-1}$ |
| ✗ | ✓ | ✓ | ✓ | $(5.99 \pm 0.54) \times 10^{-3}$ |
| ✗ | ✗ | ✓ | ✓ | $(7.80 \pm 0.70) \times 10^{-1}$ |
| ✗ | ✓ | ✗ | ✓ | $(4.20 \pm 1.23) \times 10^{-2}$ |
| ✗ | ✓ | ✓ | ✗ | $(2.41 \pm 0.24) \times 10^{-2}$ |

Table 9: Ablation study on adaptive training components for the Korteweg–De Vries equation using the best (α, β) initialization from Table 7. Each row corresponds to a different combination of enabled (✓) or disabled (✗) components. Reported values are mean \pm standard error over three seeds.

the fully adaptive configuration. Among the remaining methods, RAD again emerges as the most critical, as removing it leads to divergence. The removal of causal training or learning-rate annealing results in final errors on the order of 10^{-2} , suggesting both play meaningful, but secondary, roles.

5.4. Sine Gordon Equation

We next consider the Sine Gordon equation (see Appendix D for details). As shown in Table 10, the configuration with $\alpha = 0$ and $\beta = 0$ achieves the lowest mean relative L^2 error, indicating that initializing the network with an effective depth of a single layer is beneficial in this setting. Among the trained instances, the lowest error achieved for this configuration is 2.84×10^{-2} . Figure 12 shows the corresponding prediction, reference solution and absolute error. Notably, the error grows toward the final stages of the temporal do-

| Configuration | Relative L^2 Error | Final Loss |
|---|--|--|
| $\alpha = 0, \beta = 0$ | $(3.26 \pm 0.21) \times 10^{-2}$ | $(1.24 \pm 0.17) \times 10^{-6}$ |
| $\alpha = 1, \beta = 0$ | $(4.15 \pm 0.51) \times 10^{-2}$ | $(4.15 \pm 1.48) \times 10^{-7}$ |
| $\alpha = 0, \beta = 1$ | $(7.64 \pm 3.61) \times 10^{-2}$ | $(4.28 \pm 0.87) \times 10^{-7}$ |
| $\alpha = 1, \beta = 1$ | $(5.61 \pm 0.84) \times 10^{-2}$ | $(3.28 \pm 3.02) \times 10^{-6}$ |

Table 10: Results for different RGA KAN (α, β) initializations on the Sine Gordon equation. Reported values are mean \pm standard error over three seeds. The best performing configuration in terms of relative L^2 error is indicated in bold.

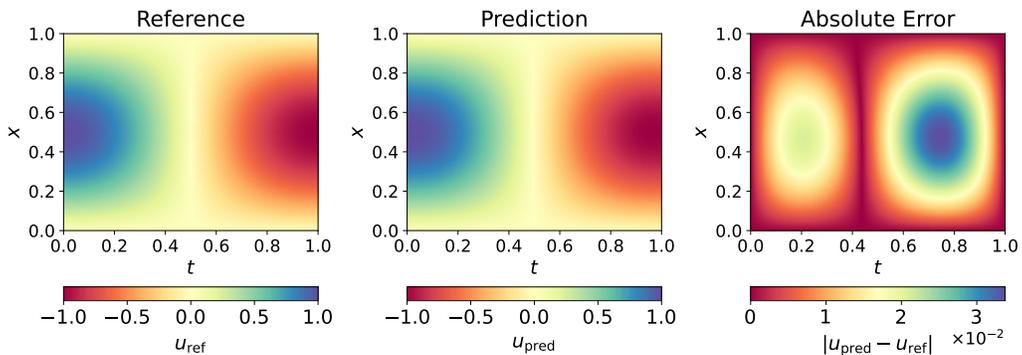


Figure 12: Reference solution (left), RGA KAN prediction (middle) and absolute error (right) for the Sine Gordon equation, shown for the random seed corresponding to the best-performing model instance.

main, suggesting that a more strict tolerance for causal training or additional training iterations could further improve performance.

The comparative results across architectures are presented in Table 11. Here, the cPIKAN model diverges with a mean relative L^2 error around 50%. PirateNet performs substantially better but still lags behind RGA KANs, which achieve a lower error compared to PirateNets across all (α, β) configurations tested. Moreover, the variability across seeds is noticeably lower for RGA KANs, indicating more consistent performance.

Finally, the ablation results are summarized in Table 12. In this case, all ablations lead to relative L^2 errors of the same order of magnitude as the fully adaptive configuration, with the notable exception of removing learning-rate annealing. In that scenario, the error increases to an average of 1.8×10^{-1} , highlighting the key role of learning-rate annealing for this benchmark.

| Architecture | Parameters | Relative L^2 Error | Time / Iter. |
|----------------|---------------|--|----------------|
| cPIKAN | 18,307 | $(5.01 \pm 0.62) \times 10^{-1}$ | 3.93 ms |
| PirateNet | 19,228 | $(8.02 \pm 1.70) \times 10^{-2}$ | 3.68 ms |
| RGA KAN | 18,422 | $(3.26 \pm 0.21) \times 10^{-2}$ | 6.30 ms |

Table 11: Performance comparison on the Sine Gordon equation across different architectures. Reported values are mean \pm standard error over three random seeds. The RGA KAN row uses the best (α, β) initialization from Table 10. The best performing architecture in terms of relative L^2 error is indicated in bold.

| RBA | RAD | Causal | LR Annealing | Relative L^2 Error |
|-----|-----|--------|--------------|----------------------------------|
| ✓ | ✓ | ✓ | ✓ | $(3.26 \pm 0.21) \times 10^{-2}$ |
| ✓ | ✗ | ✗ | ✗ | $(8.85 \pm 0.73) \times 10^{-2}$ |
| ✗ | ✓ | ✓ | ✓ | $(4.74 \pm 0.96) \times 10^{-2}$ |
| ✗ | ✗ | ✓ | ✓ | $(4.74 \pm 1.16) \times 10^{-2}$ |
| ✗ | ✓ | ✗ | ✓ | $(3.46 \pm 0.43) \times 10^{-2}$ |
| ✗ | ✓ | ✓ | ✗ | $(1.80 \pm 0.20) \times 10^{-1}$ |

Table 12: Ablation study on adaptive training components for the Sine Gordon equation using the best (α, β) initialization from Table 10. Each row corresponds to a different combination of enabled (✓) or disabled (✗) components. Reported values are mean \pm standard error over three seeds.

5.5. Advection Equation

The advection equation (see Appendix D for details), especially at high transport velocities (the constant multiplying the spatial derivative of the solution field in Eq. (D.14)), is a challenging benchmark that typically requires specialized training strategies (e.g., learnable spatial periodical embeddings as in [23]) to obtain accurate solutions. In this work, however, we intentionally refrain from introducing any problem-specific modifications to maintain a unified training pipeline across all PDEs. To this end, we set the advection velocity to $c = 20$, which remains a nontrivial setting. Table 13 shows that initializing RGA KANs with $\alpha = 1$ and $\beta = 1$ yields the best performance, with both the lowest mean error and the smallest variability across seeds. Among these runs, the best-performing model achieves a final relative L^2 error of 1.81×10^{-4} , and its prediction is depicted in Figure 13 alongside the

| Configuration | Relative L^2 Error | Final Loss |
|---|--|--|
| $\alpha = 0, \beta = 0$ | $(6.29 \pm 1.02) \times 10^{-4}$ | $(2.46 \pm 0.54) \times 10^{-4}$ |
| $\alpha = 1, \beta = 0$ | $(4.78 \pm 1.18) \times 10^{-4}$ | $(2.40 \pm 0.96) \times 10^{-4}$ |
| $\alpha = 0, \beta = 1$ | $(3.08 \pm 1.98) \times 10^{-3}$ | $(1.74 \pm 1.39) \times 10^{-3}$ |
| $\alpha = 1, \beta = 1$ | $(2.41 \pm 0.39) \times 10^{-4}$ | $(5.89 \pm 3.04) \times 10^{-5}$ |

Table 13: Results for different RGA KAN (α, β) initializations on the advection equation. Reported values are mean \pm standard error over three seeds. The best performing configuration in terms of relative L^2 error is indicated in bold.

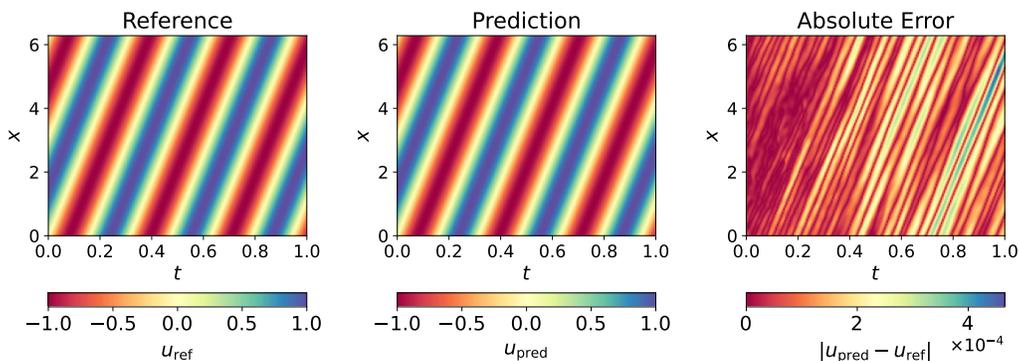


Figure 13: Reference solution (left), RGA KAN prediction (middle) and absolute error (right) for the advection equation, shown for the random seed corresponding to the best-performing model instance.

| Architecture | Parameters | Relative L^2 Error | Time / Iter. |
|----------------|---------------|--|----------------|
| cPIKAN | 18,397 | $(4.11 \pm 4.01) \times 10^{-1}$ | 2.19 ms |
| PirateNet | 19,246 | $(1.13 \pm 0.13) \times 10^0$ | 2.67 ms |
| RGA KAN | 18,502 | $(2.41 \pm 0.39) \times 10^{-4}$ | 3.98 ms |

Table 14: Performance comparison on the advection equation across different architectures. Reported values are mean \pm standard error over three random seeds. The RGA KAN row uses the best (α, β) initialization from Table 13. The best performing architecture in terms of relative L^2 error is indicated in bold.

reference solution and absolute error.

The comparison with the other architectures (Table 14) reveals a stark contrast. Both cPIKAN and PirateNet exhibit poor performance, with large errors and clear indications of instability. While two out of three cPIKAN runs produce moderate errors, the third diverges, resulting in a large mean error and high standard deviation. PirateNets fail consistently across all seeds. In contrast, RGA KANs outperform both baselines by several orders of magnitude, maintaining low errors and small variance, once again showcasing their stability and reliability.

The ablation results in Table 15 highlight the importance of adaptive training for this PDE. Training with only RBA leads to divergence, as does removing both RBA and causal training (indicated by the dash in the table), or RBA and learning-rate annealing. Notably, RAD is the only adaptive strategy whose removal alongside RBA does not cause divergence, although

| RBA | RAD | Causal | LR Annealing | Relative L^2 Error |
|-----|-----|--------|--------------|----------------------------------|
| ✓ | ✓ | ✓ | ✓ | $(2.41 \pm 0.39) \times 10^{-4}$ |
| ✓ | ✗ | ✗ | ✗ | $(1.01 \pm 0.02) \times 10^0$ |
| ✗ | ✓ | ✓ | ✓ | $(9.22 \pm 4.90) \times 10^{-4}$ |
| ✗ | ✗ | ✓ | ✓ | $(6.96 \pm 5.31) \times 10^{-3}$ |
| ✗ | ✓ | ✗ | ✓ | - |
| ✗ | ✓ | ✓ | ✗ | $(7.29 \pm 3.72) \times 10^{-1}$ |

Table 15: Ablation study on adaptive training components for the advection equation using the best (α, β) initialization from Table 13. Each row corresponds to a different combination of enabled (✓) or disabled (✗) components. Reported values are mean \pm standard error over three seeds.

performance is still noticeably degraded. This stands in contrast to previous benchmarks, where RAD often had the most pronounced effect, emphasizing that the relative contribution of each adaptive technique is highly problem dependent and should be evaluated individually for each PDE, as done in this study.

5.6. Helmholtz Equation

We next consider the Helmholtz equation (see Appendix D for details), defined solely on a spatial domain and thus involving no initial condition. Consequently, this benchmark excludes both learning-rate annealing (as the loss contains only one PDE residual term) and causal training (due to the absence of temporal dependencies). Only RBA and RAD are employed as adaptive training strategies. Table 16 shows that initializing RGA KANs with $\alpha = 1$ and $\beta = 1$ yields the best performance, achieving both the lowest

| Configuration | Relative L^2 Error | Final Loss |
|---|--|--|
| $\alpha = 0, \beta = 0$ | $(9.91 \pm 2.28) \times 10^{-5}$ | $(2.63 \pm 0.25) \times 10^{-3}$ |
| $\alpha = 1, \beta = 0$ | $(1.08 \pm 0.36) \times 10^{-4}$ | $(1.47 \pm 0.31) \times 10^{-3}$ |
| $\alpha = 0, \beta = 1$ | $(8.60 \pm 1.78) \times 10^{-5}$ | $(2.20 \pm 0.24) \times 10^{-3}$ |
| $\alpha = 1, \beta = 1$ | $(2.44 \pm 0.34) \times 10^{-5}$ | $(2.84 \pm 1.24) \times 10^{-4}$ |

Table 16: Results for different RGA KAN (α, β) initializations on the Helmholtz equation. Reported values are mean \pm standard error over three seeds. The best performing configuration in terms of relative L^2 error is indicated in bold.

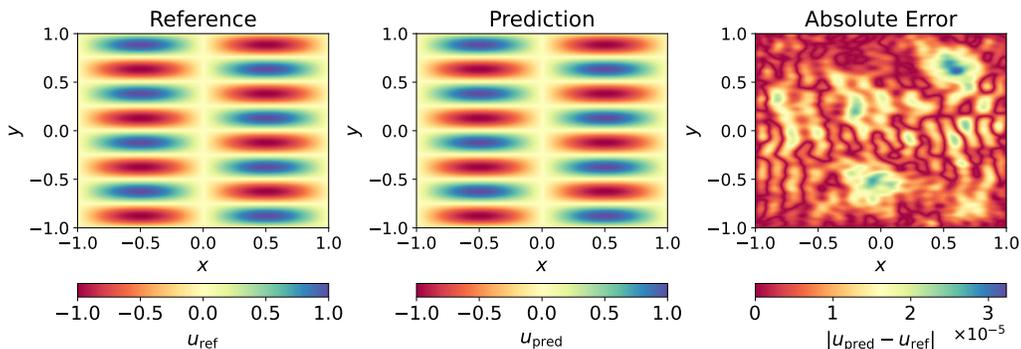


Figure 14: Reference solution (left), RGA KAN prediction (middle) and absolute error (right) for the Helmholtz equation, shown for the random seed corresponding to the best-performing model instance.

mean relative L^2 error and the smallest variance across seeds. Notably, this configuration also attains the smallest final loss of order $\mathcal{O}(10^{-4})$, which is particularly meaningful for this benchmark: the Helmholtz equation is well known to exhibit a high and challenging loss landscape at initialization [35]. The best-performing model for this configuration achieves a relative L^2 error of 1.76×10^{-5} , and its prediction is shown in Figure 14.

In the cross-architecture comparison (Table 17), all three architectures converge, but with clearly different accuracies. cPIKANs achieve a relative L^2 error of order $\mathcal{O}(10^{-3})$, PirateNets improve by one order of magnitude, and RGA KANs outperform both by another order of magnitude, reaching the $\mathcal{O}(10^{-5})$ range. The variance across seeds remains low for all three architectures.

The ablation study (Table 18) provides interesting insights for the training dynamics of the models trained for this PDE. Unlike in the case of the Korteweg–De Vries or the advection equation, the RGA KAN architecture converges even without any adaptive training. In fact, the performance in this setting is comparable to that of PirateNets with both adaptive strate-

| Architecture | Parameters | Relative L^2 Error | Time / Iter. |
|----------------|---------------|--|----------------|
| cPIKAN | 18,307 | $(1.03 \pm 0.21) \times 10^{-3}$ | 3.25 ms |
| PirateNet | 19,230 | $(1.89 \pm 0.25) \times 10^{-4}$ | 3.05 ms |
| RGA KAN | 18,423 | $(2.44 \pm 0.34) \times 10^{-5}$ | 4.30 ms |

Table 17: Performance comparison on the Helmholtz equation across different architectures. Reported values are mean \pm standard error over three random seeds. The RGA KAN row uses the best (α, β) initialization from Table 16. The best performing architecture in terms of relative L^2 error is indicated in bold.

| RBA | RAD | Relative L^2 Error |
|-----|-----|----------------------------------|
| ✓ | ✓ | $(2.44 \pm 0.34) \times 10^{-5}$ |
| ✓ | ✗ | $(1.61 \pm 1.05) \times 10^{-3}$ |
| ✗ | ✓ | $(6.81 \pm 1.82) \times 10^{-5}$ |
| ✗ | ✗ | $(1.95 \pm 0.54) \times 10^{-4}$ |

Table 18: Ablation study on adaptive training components for the Helmholtz equation using the best (α, β) initialization from Table 16. Each row corresponds to a different combination of enabled (✓) or disabled (✗) components. Reported values are mean \pm standard error over three seeds.

gies active. Another notable observation is that disabling only RAD leads to higher errors than disabling both RBA and RAD. This suggests that the interaction between adaptive components can be nontrivial: while RBA combined with RAD yields the best performance, using RBA alone may actually be suboptimal for this PDE under the chosen hyperparameter configuration.

5.7. Poisson Equation

The final benchmark is the Poisson equation (see Appendix D for details), which – similarly to the Helmholtz equation – is defined on a purely spatial domain and therefore involves no initial condition. As a result, only RBA and RAD are employed as adaptive training strategies. The source term is chosen such that the analytical solution is $u(x, y) = \sin(\pi\omega x)\sin(\pi\omega y)$, allowing us to systematically investigate how performance degrades as the frequency parameter ω increases. We consider $\omega \in \{1, 2, 4\}$ and Table 19 presents the results of the (α, β) initialization study for each ω . Evidently, different initialization configurations are optimal for different frequencies, and performance deteriorates with increasing ω . Moreover, the standard error of the mean for the optimal configuration grows alongside ω : for $\omega = 1$, it is roughly an order of magnitude lower than the mean error, whereas for $\omega = 4$ it exceeds half of it, indicating increased sensitivity to initialization and optimization. For each ω value, we select the best-performing seed of the optimal configuration, achieving relative L^2 errors of 7.33×10^{-7} , 2.74×10^{-5} , and 3.34×10^{-3} for $\omega = 1, 2$, and 4 , respectively. Figure 15 shows the reference solution, prediction and absolute error for these runs.

| (α, β) | Metric | $\omega = 1$ | $\omega = 2$ | $\omega = 4$ |
|-------------------|------------------|--|--|--|
| (0, 0) | Rel. L^2 Error | $(8.95 \pm 1.97) \times 10^{-6}$ | $(2.92 \pm 2.20) \times 10^{-4}$ | $(2.15 \pm 1.09) \times 10^{-2}$ |
| | Final Loss | $(1.98 \pm 0.08) \times 10^{-6}$ | $(3.35 \pm 0.69) \times 10^{-4}$ | $(1.30 \pm 0.07) \times 10^{-1}$ |
| (1, 0) | Rel. L^2 Error | $(2.03 \pm 0.70) \times 10^{-6}$ | $(4.53 \pm 0.90) \times 10^{-5}$ | $(1.93 \pm 0.83) \times 10^{-2}$ |
| | Final Loss | $(7.08 \pm 0.31) \times 10^{-7}$ | $(9.30 \pm 1.29) \times 10^{-5}$ | $(1.44 \pm 0.59) \times 10^{-1}$ |
| (0, 1) | Rel. L^2 Error | $(3.33 \pm 1.58) \times 10^{-6}$ | $(6.42 \pm 0.34) \times 10^{-5}$ | $(9.67 \pm 5.80) \times 10^{-3}$ |
| | Final Loss | $(7.35 \pm 0.34) \times 10^{-7}$ | $(2.69 \pm 0.58) \times 10^{-4}$ | $(9.40 \pm 3.03) \times 10^{-2}$ |
| (1, 1) | Rel. L^2 Error | $(1.10 \pm 0.26) \times 10^{-6}$ | $(9.24 \pm 2.05) \times 10^{-5}$ | $(2.12 \pm 0.94) \times 10^{-2}$ |
| | Final Loss | $(5.66 \pm 0.18) \times 10^{-7}$ | $(2.00 \pm 0.05) \times 10^{-5}$ | $(2.48 \pm 0.80) \times 10^{-2}$ |

Table 19: Results for different RGA KAN (α, β) initializations on the Poisson equation for $\omega \in \{1, 2, 4\}$. Reported values are mean \pm standard error over three seeds. The best performing configuration in terms of relative L^2 error per column is indicated in bold.

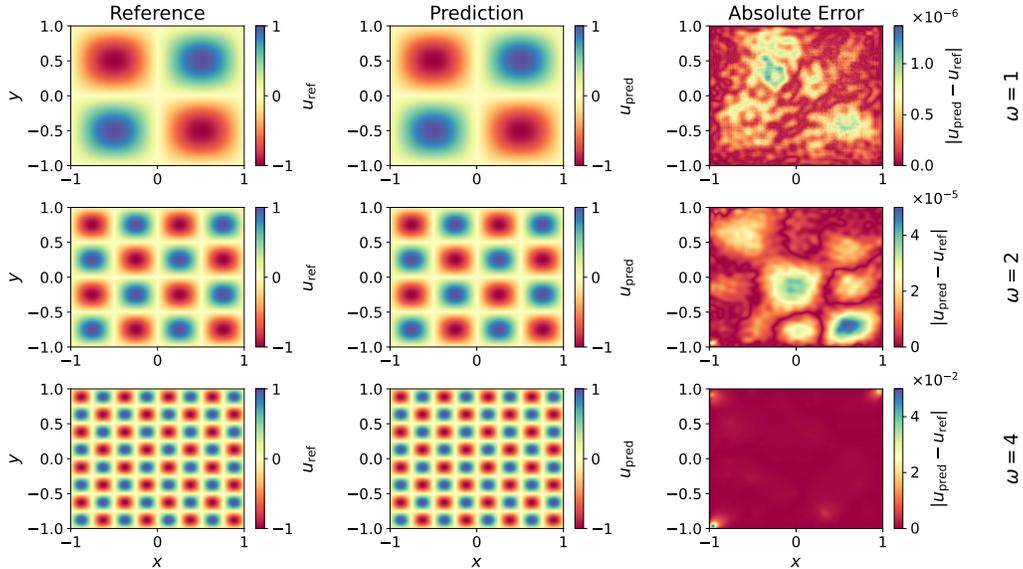


Figure 15: Reference solution (left column), RGA KAN prediction (middle column) and absolute error (right column) for the Poisson equation, shown for the random seed corresponding to the best-performing model instance per value of $\omega \in \{1, 2, 4\}$.

We next compare architectures using the best (α, β) initialization for each ω . The parameter counts and iteration times are shown in Table 20, while the accuracy results in terms of relative L^2 error are summarized in Table 21. For $\omega = 1$ and $\omega = 2$, all architectures converge, with RGA KANs achieving the lowest error by a wide margin. However, for $\omega = 4$ the situation changes: PirateNets diverge, while, interestingly, for the first time across all benchmarks, cPIKANs outperform RGA KANs under the default training settings, although both achieve errors of the same order of magnitude.

The ablation study (Table 22) provides additional insight. For $\omega = 1$

| Architecture | Parameters | Time / Iteration (ms) |
|--------------|------------|-----------------------|
| cPIKAN | 18,307 | 2.93 ms |
| PirateNet | 19,230 | 2.94 ms |
| RGA KAN | 18,423 | 3.81 ms |

Table 20: Number of parameters and average time per training iteration (milliseconds) for each architecture on the Poisson equation.

| Architecture | $\omega = 1$ | $\omega = 2$ | $\omega = 4$ |
|--------------|--|--|--|
| cPIKAN | $(1.56 \pm 0.48) \times 10^{-5}$ | $(2.15 \pm 0.99) \times 10^{-3}$ | $(5.08 \pm 1.41) \times 10^{-3}$ |
| PirateNet | $(7.66 \pm 2.72) \times 10^{-6}$ | $(1.58 \pm 0.47) \times 10^{-4}$ | $(2.57 \pm 1.86) \times 10^0$ |
| RGA KAN | $(1.10 \pm 0.26) \times 10^{-6}$ | $(4.53 \pm 0.90) \times 10^{-5}$ | $(9.67 \pm 5.80) \times 10^{-3}$ |

Table 21: Performance comparison on the Poisson equation across different architectures. Reported values are mean \pm standard error over three random seeds for $\omega \in \{1, 2, 4\}$. The RGA KAN row uses the best (α, β) initialization from Table 19. The best performing architecture in terms of relative L^2 error per column is indicated in bold.

and $\omega = 2$, RGA KANs converge even without adaptive training, albeit with somewhat higher errors. For $\omega = 4$, however, performance collapses without adaptive methods. Remarkably, when RBA is disabled but RAD is retained, the error decreases by nearly an order of magnitude compared to the non-adaptive case, with very low variance across runs. In fact, this configuration outperforms the average error achieved by cPIKANs at $\omega = 4$, effectively restoring the advantage of RGA KANs for this challenging regime. This observation is consistent with the findings for the Helmholtz equation, which is structurally similar to the Poisson equation, where RAD also emerged as the dominant adaptive component.

| RBA | RAD | $\omega = 1$ | $\omega = 2$ | $\omega = 4$ |
|-----|-----|----------------------------------|----------------------------------|----------------------------------|
| ✓ | ✓ | $(1.10 \pm 0.26) \times 10^{-6}$ | $(4.53 \pm 0.90) \times 10^{-5}$ | $(9.67 \pm 5.80) \times 10^{-3}$ |
| ✓ | ✗ | $(1.95 \pm 0.12) \times 10^{-6}$ | $(7.97 \pm 1.29) \times 10^{-5}$ | $(8.01 \pm 3.72) \times 10^{-2}$ |
| ✗ | ✓ | $(1.04 \pm 0.73) \times 10^{-5}$ | $(5.01 \pm 0.27) \times 10^{-5}$ | $(1.19 \pm 0.25) \times 10^{-3}$ |
| ✗ | ✗ | $(2.54 \pm 1.31) \times 10^{-5}$ | $(5.09 \pm 0.62) \times 10^{-5}$ | $(1.23 \pm 0.79) \times 10^{-1}$ |

Table 22: Ablation study on adaptive training components for the Poisson equation using the best (α, β) initialization from Table 19. Each row corresponds to a different combination of enabled (✓) or disabled (✗) components. Reported values are mean \pm standard error over three seeds for $\omega \in \{1, 2, 4\}$.

6. Conclusion and Outlook

In this work, we studied the deep training of Chebyshev-based KANs with the goal of improving their stability and accuracy in PDE benchmarks under a uniform training setup. We began by examining their initialization

properties, since initialization has historically been the starting point for enabling depth scaling in neural architectures. To this end, we proposed a Glorot-like initialization scheme that is basis-agnostic and, as demonstrated by its application to sine-based KANs within the RGA KAN architecture, not tied to a specific basis family. Preliminary results on function fitting and PDE tasks showed that this initialization alone significantly improved training outcomes compared to the default initialization, in some cases by several orders of magnitude. For certain benchmarks such as Burgers’ equation, the proposed initialization was sufficient to train deeper models successfully, whereas for others, such as Allen–Cahn, it fell short. This observation made it clear that initialization alone was not enough to fully address the depth-scaling issue.

Motivated by this, and inspired by PirateNets, we analyzed the properties of Chebyshev KANs at initialization and observed strong parallels to standard MLPs. This insight led to the design of the RGA KAN architecture. This architecture proved effective in overcoming divergence for deeper networks, with increased depth and parameter count leading to improved accuracy rather than degradation. Through the IB perspective, we linked this desirable behavior to the model’s ability to traverse all three characteristic phases of training – fitting, diffusion, and diffusion equilibrium – unlike baseline cPIKANs that tend to stall prematurely. Equipped with the proposed initialization and architecture, we then established a fixed training pipeline with adaptive components, namely RBA, RAD, causal training and learning-rate annealing. We compared RGA KANs against parameter-matched cPIKANs (also using the new initialization) and PirateNets, which are widely considered state of the art for many PDE benchmarks. Importantly, our aim was not to finely tune hyperparameters on a per-task basis but to test the performance of the proposed design in a uniform setting. Across all seven PDE benchmarks, RGA KANs outperformed both baselines, often by large margins, and remained stable in cases where the others diverged. The ablation studies further clarified the relative contribution of each adaptive method, but also revealed that in several cases the combination of the proposed initialization and architecture alone was sufficient to achieve good accuracy without divergence.

Naturally, this study also has limitations. To conserve computational resources, we deliberately avoided per-task hyperparameter tuning and considered networks of medium width at 16 neurons for the final PDE benchmarks. Our focus was not to set state-of-the-art results through exhaustive tuning,

but to show that the proposed initialization and architecture provide a strong and robust foundation that already outperforms both baseline cPIKANs and PirateNets under default settings. A more extensive hyperparameter search would likely further improve performance. Another limitation is that all experiments relied on first-order optimization, specifically Adam. While this is standard in the PIML literature, recent work has shown that higher-order optimizers can yield remarkable improvements [70, 71], and it would be valuable to explore the proposed architecture under such optimization regimes.

The initialization itself also opens several avenues for future work. Its basis-agnostic nature suggests that it can be applied to domains beyond PDE solving, as well as to KAN variants using other bases, which may be better suited to different tasks. Similarly, while the RGA KAN architecture was designed with PDE applications and depth-scaling in mind, its structure bears similarities to transformer architectures, hinting at its potential relevance to other application domains. Moreover, it would be interesting to test the architecture in conjunction with alternative representations of the Kolmogorov–Arnold representation theorem, such as ActNets [56] or KKANs [42], to assess whether the benefits of residual gating and adaptive training extend beyond standard KAN formulations.

Appendix A. Detailed Derivations

Appendix A.1. Proposed Initialization Scheme

Consider a single KAN layer with outputs given by Eq. (26):

$$y_j = \sum_{i=1}^{d_1} \sum_{m=1}^D Z_{im}^{(j)}, \quad (\text{A.1})$$

where all biases have been set to zero at initialization and we have defined

$$Z_{im}^{(j)} = w_{jim} B_m(x_i). \quad (\text{A.2})$$

Since weights are independent of inputs, expectations factor as products. Therefore

$$\mathbb{E} \left[Z_{im}^{(j)} \right] = \mathbb{E} [w_{jim} B_m(x_i)] = \mathbb{E} [w_{jim}] \mathbb{E} [B_m(x_i)] = 0 \quad (\text{A.3})$$

holds. In addition, considering two pairs of indices $(i, m) \neq (i', m')$, we find

$$\begin{aligned} \mathbb{E} \left[Z_{im}^{(j)} Z_{i'm'}^{(j)} \right] &= \mathbb{E} [w_{jim} w_{ji'm'} B_m(x_i) B_{m'}(x_{i'})] \\ &= \mathbb{E} [w_{jim} w_{ji'm'}] \mathbb{E} [B_m(x_i) B_{m'}(x_{i'})] \\ &= \mathbb{E} [w_{jim}] \mathbb{E} [w_{ji'm'}] \mathbb{E} [B_m(x_i) B_{m'}(x_{i'})] \\ &= 0, \end{aligned} \quad (\text{A.4})$$

where we have taken into consideration that distinct weights are independent and zero-mean. Therefore, for $(i, m) \neq (i', m')$, we arrive at

$$\begin{aligned} \text{Cov} \left(Z_{im}^{(j)}, Z_{i'm'}^{(j)} \right) &= \mathbb{E} \left[Z_{im}^{(j)} Z_{i'm'}^{(j)} \right] - \mathbb{E} \left[Z_{im}^{(j)} \right] \mathbb{E} \left[Z_{i'm'}^{(j)} \right] \\ &= 0 - 0 \cdot 0 = 0. \end{aligned} \quad (\text{A.5})$$

Using this result, the variance of the sum in Eq. (A.1) reduces to a sum of individual variances,

$$\text{Var}(y_j) = \sum_{i=1}^{d_1} \sum_{m=1}^D \text{Var} \left(Z_{im}^{(j)} \right) = \sum_{i=1}^{d_1} \sum_{m=1}^D \text{Var} (w_{jim} B_m(x_i)). \quad (\text{A.6})$$

For a single term, the independence of w_{jim} and x_i together with $\mathbb{E}[w_{jim}] = 0$ leads to

$$\begin{aligned}\text{Var}[w_{jim} B_m(x_i)] &= \mathbb{E}^2[B_m(x_i)] \text{Var}(w_{jim}) + \text{Var}(w_{jim}) \text{Var}[B_m(x_i)] \\ &= \sigma_m^2 \mathbb{E}[B_m(x_i)^2].\end{aligned}\quad (\text{A.7})$$

Therefore,

$$\text{Var}(y_j) = d_I \sum_{m=1}^D \sigma_m^2 \mu_m^{(0)}, \quad (\text{A.8})$$

and enforcing $\text{Var}(y_j) = \text{Var}(x_i) = 1$ leads directly to Eq. (28) of the main text.

As far as the backward pass is concerned, differentiating Eq. (A.1) with respect to x_i yields

$$\frac{\partial y_j}{\partial x_i} = \sum_{m=1}^D w_{jim} B'_m(x_i). \quad (\text{A.9})$$

The loss gradient with respect to x_i then becomes

$$\delta x_i = \sum_{j=1}^{d_O} \frac{\partial y_j}{\partial x_i} \delta y_j = \sum_{j=1}^{d_O} \sum_{m=1}^D w_{jim} B'_m(x_i) \delta y_j. \quad (\text{A.10})$$

Following the same reasoning as for the forward pass, distinct (j, m) pairs are uncorrelated, so

$$\text{Var}(\delta x_i) = \sum_{j=1}^{d_O} \sum_{m=1}^D \text{Var}(w_{jim} B'_m(x_i) \delta y_j). \quad (\text{A.11})$$

Each summand can be evaluated as

$$\text{Var}(w_{jim} B'_m(x_i) \delta y_j) = \sigma_m^2 \mathbb{E}[B'_m(x_i)^2] \mathbb{E}[\delta y_j^2] = \sigma_m^2 \mu_m^{(1)} \text{Var}(\delta y_j). \quad (\text{A.12})$$

Thus,

$$\text{Var}(\delta x_i) = d_O \text{Var}(\delta y_j) \sum_{m=1}^D \sigma_m^2 \mu_m^{(1)}, \quad (\text{A.13})$$

and imposing $\text{Var}(\delta x_i) = \text{Var}(\delta y_j)$ yields Eq. (30).

Appendix A.2. Chebyshev-based KAN Derivative

Following [23], we consider small activations at initialization and adopt the linear-regime approximation, where

$$\tanh x \approx x, \quad \frac{d}{dx} \tanh x \approx 1. \quad (\text{A.14})$$

In this regime, and recalling from Eq. (16) that $B_m(x) = T_m(\tanh x)$, we expand the Chebyshev polynomial $T_m(x)$ around zero and retain only the linear term $\mathcal{O}(x)$. The basis functions then simplify to

$$B_m(x) \approx T_m(0) + T'_m(0)x + \mathcal{O}(x^2). \quad (\text{A.15})$$

Differentiating Eq. (A.15) yields

$$B'_m(x) \approx T'_m(0) = m U_{m-1}(0), \quad (\text{A.16})$$

where $U_n(\cdot)$ denotes the n -th Chebyshev polynomial of the second kind and we have used the identity $T'_m(z) = m U_{m-1}(z)$. Since

$$U_n(0) = \sin\left(\frac{(n+1)\pi}{2}\right), \quad (\text{A.17})$$

it follows that

$$U_{m-1}(0) = \sin\left(\frac{m\pi}{2}\right) = \begin{cases} 0, & m \text{ even,} \\ (-1)^{\frac{m-1}{2}}, & m \text{ odd.} \end{cases} \quad (\text{A.18})$$

Substituting Eq. (A.18) into Eq. (A.16) gives

$$B'_m(x) \approx \begin{cases} 0, & m \text{ even,} \\ m (-1)^{\frac{m-1}{2}}, & m \text{ odd.} \end{cases} \quad (\text{A.19})$$

Therefore, substituting Eq. (A.19) into Eq. (36) from the main text, and using Eq. (38), we find

$$\begin{aligned} \frac{\partial u_j^{(l)}(x; \boldsymbol{\theta})}{\partial x} &\approx \sum_{i=1}^{d_{l-1}} \frac{\partial u_i^{(l-1)}(x; \boldsymbol{\theta})}{\partial x} \sum_{m \text{ odd}}^D m (-1)^{\frac{m-1}{2}} w_{jim}^{(l)} \\ &= \sum_{i=1}^{d_{l-1}} \tilde{w}_{ji}^{(l)} \frac{\partial u_i^{(l-1)}(x; \boldsymbol{\theta})}{\partial x}. \end{aligned} \quad (\text{A.20})$$

Appendix B. Implementation Details

All neural network architectures utilized in this study are implemented in JAX [72] using the jaxKAN framework [54] and trained at the highest precision settings on an NVIDIA GeForce RTX 4090 GPU. Their performance is assessed in terms of the L^2 error of the predicted solution, \mathbf{u}_{pred} , relative to a reference solution, \mathbf{u}_{ref} , i.e.,

$$\mathcal{E} = \frac{\|\mathbf{u}_{\text{pred}} - \mathbf{u}_{\text{ref}}\|_2}{\|\mathbf{u}_{\text{ref}}\|_2}, \quad (\text{B.1})$$

where \mathcal{E} and $\|\cdot\|_2$ denote the relative L^2 error and L^2 norm, respectively.

Appendix C. Function Fitting Benchmarks

In this appendix we provide the analytic definitions of the benchmark functions used in the function-fitting experiments of Section 3.2.1. Each function is defined on the hypercube $[-1, 1]^d$, where d denotes the input dimensionality.

One-dimensional oscillatory function.

$$f_1(x) = \sin(2\pi x) + 3x. \quad (\text{C.1})$$

Two-dimensional product function.

$$f_2(x_1, x_2) = x_1 x_2. \quad (\text{C.2})$$

Two-dimensional Bessel-based function.

$$f_3(x_1, x_2) = I_1(x_1) + \exp(I_1^{(e)}(x_2)) + \sin(x_1 x_2), \quad (\text{C.3})$$

where $I_1(\cdot)$ denotes the modified Bessel function of the first kind of order 1, and $I_1^{(e)}(\cdot)$ its exponentially scaled version.

Three-dimensional Hartmann function.

$$f_4(x_1, x_2, x_3) = -\sum_{k=1}^4 \alpha_k \exp\left(-\sum_{j=1}^3 A_{kj} (x_j - P_{kj})^2\right), \quad (\text{C.4})$$

where

$$\alpha = (1.0, 1.2, 3.0, 3.2),$$

and

$$A = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix}, \quad P = 10^{-4} \begin{bmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{bmatrix}.$$

Five-dimensional Sobol g-function.

$$f_5(x_1, \dots, x_5) = \prod_{j=1}^5 \frac{|4x_j - 2| + a_j}{1 + a_j}, \quad (\text{C.5})$$

where $a_j = \frac{j-2}{2}$ for $j = 1, \dots, 5$.

Appendix D. Studied Partial Differential Equations

In this appendix, we present the PDEs studied throughout this work, including their governing equations, boundary and/or initial conditions, and corresponding reference solutions. The equations are presented with the specific parameter values used for this study and are listed in the same order in which they appear in Section 5 of the main text.

Allen–Cahn Equation. The Allen–Cahn equation on the spatiotemporal domain $t \in [0, 1]$, $x \in [-1, 1]$ is given by

$$\frac{\partial u}{\partial t} - 10^{-4} \frac{\partial^2 u}{\partial x^2} = 5(u - u^3). \quad (\text{D.1})$$

It is considered with initial condition

$$u(0, x) = x^2 \cos(\pi x), \quad (\text{D.2})$$

and periodic boundary conditions

$$u(t, -1) = u(t, 1), \quad \frac{\partial u}{\partial x}(t, -1) = \frac{\partial u}{\partial x}(t, 1). \quad (\text{D.3})$$

The reference solution shown in Figure D.16 corresponds to the data used in [23] and accessed from the paper’s accompanying GitHub repository [73].

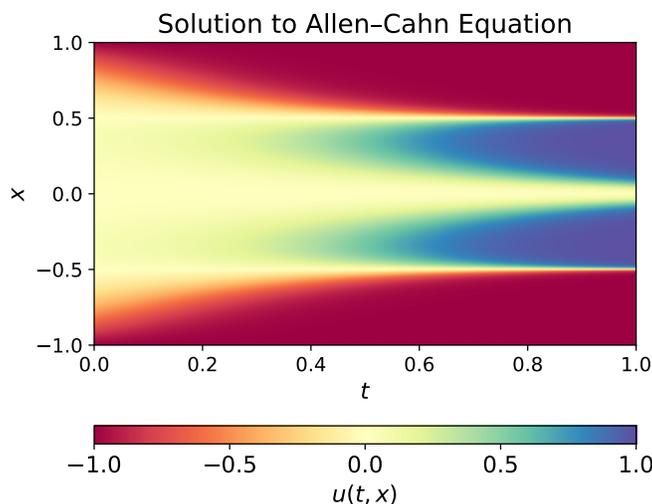


Figure D.16: Reference solution of the Allen–Cahn equation on $t \in [0, 1]$, $x \in [-1, 1]$.

Burgers' Equation. The viscous Burgers' equation on the spatiotemporal domain $t \in [0, 1]$, $x \in [-1, 1]$ is given by

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \frac{1}{100\pi} \frac{\partial^2 u}{\partial x^2}. \quad (\text{D.4})$$

It is considered with initial condition

$$u(0, x) = -\sin(\pi x), \quad (\text{D.5})$$

and homogeneous Dirichlet boundary conditions

$$u(t, -1) = u(t, 1) = 0. \quad (\text{D.6})$$

The reference solution shown in Figure D.17 corresponds to the data used in [23] and accessed from the paper's accompanying GitHub repository [73].

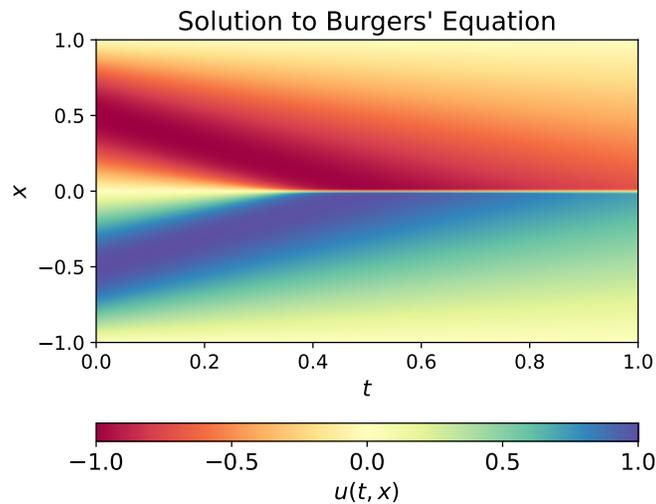


Figure D.17: Reference solution of Burgers' equation on $t \in [0, 1]$, $x \in [-1, 1]$.

Korteweg–De Vries Equation. The Korteweg–De Vries equation on the spatiotemporal domain $t \in [0, 1]$, $x \in [-1, 1]$ is given by

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + 0.022^2 \frac{\partial^3 u}{\partial x^3} = 0. \quad (\text{D.7})$$

It is considered with initial condition

$$u(0, x) = \cos(\pi x), \quad (\text{D.8})$$

and periodic boundary conditions

$$u(t, -1) = u(t, 1). \quad (\text{D.9})$$

The reference solution shown in Figure D.18 corresponds to the data used in [23] and accessed from the paper’s accompanying GitHub repository [73].

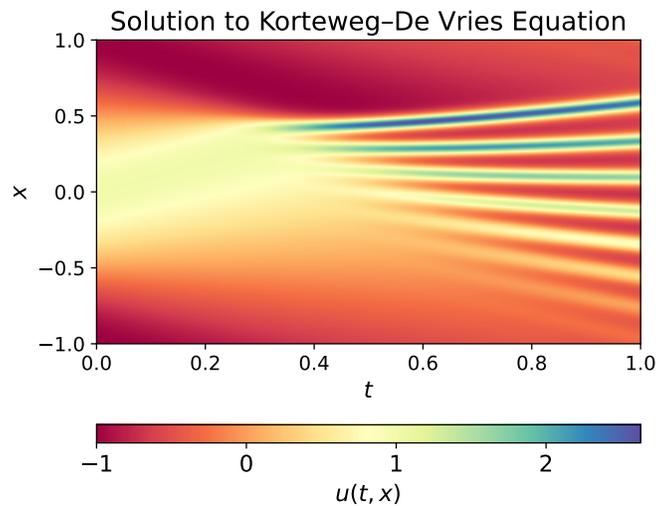


Figure D.18: Reference solution of the Korteweg–De Vries equation on $t \in [0, 1]$, $x \in [-1, 1]$.

Sine Gordon Equation. The Sine Gordon equation on the spatiotemporal domain $t \in [0, 1]$, $x \in [0, 1]$ is given by

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} + \sin u = 0. \quad (\text{D.10})$$

It is considered with initial condition

$$u(0, x) = \sin(\pi x), \quad (\text{D.11})$$

and homogeneous Dirichlet boundary conditions

$$u(t, 0) = u(t, 1) = 0. \quad (\text{D.12})$$

The analytical solution of this equation is known and given by

$$u(t, x) = \frac{1}{2} [\sin(\pi(x+t)) + \sin(\pi(x-t))], \quad (\text{D.13})$$

and is depicted in Figure D.19.

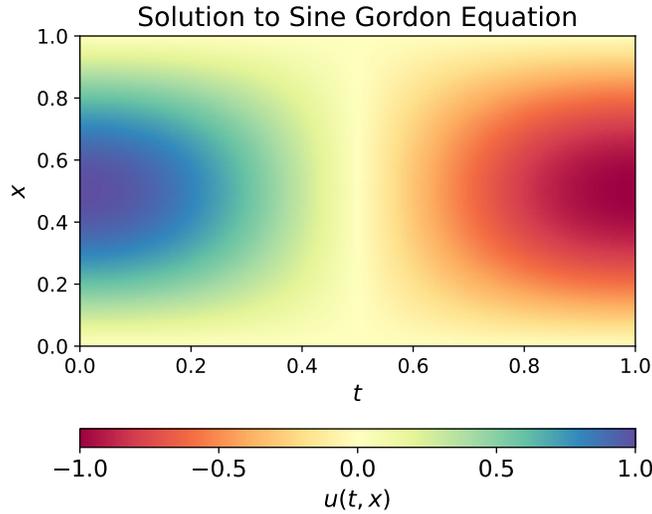


Figure D.19: Reference solution of the Sine Gordon equation on $t \in [0, 1]$, $x \in [0, 1]$.

Advection Equation. The advection equation on the spatiotemporal domain $t \in [0, 1]$, $x \in [0, 2\pi]$ is given by

$$\frac{\partial u}{\partial t} + 20 \frac{\partial u}{\partial x} = 0. \quad (\text{D.14})$$

It is considered with initial condition

$$u(0, x) = \sin x, \quad (\text{D.15})$$

and periodic boundary conditions

$$u(t, x) = u(t, x + 2\pi). \quad (\text{D.16})$$

The analytical solution of this equation is known and given by

$$u(t, x) = \sin(\text{mod}(x - 20t, 2\pi)), \quad (\text{D.17})$$

and is depicted in Figure D.20.

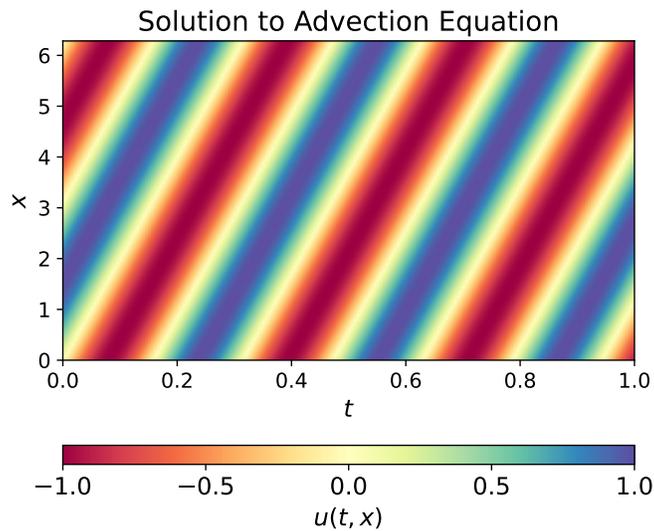


Figure D.20: Reference solution of the advection equation on $t \in [0, 1]$, $x \in [0, 2\pi]$.

Helmholtz Equation. The 2-dimensional Helmholtz equation on the spatial domain $x \in [-1, 1]$, $y \in [-1, 1]$ is given by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + u = [1 - \pi^2 (a_1^2 + a_2^2)] \sin(a_1 \pi x) \sin(a_2 \pi y). \quad (\text{D.18})$$

It is considered with homogeneous Dirichlet boundary conditions

$$u(-1, y) = u(1, y) = u(x, -1) = u(x, 1) = 0. \quad (\text{D.19})$$

The analytical solution of this equation is known and given by

$$u(x, y) = \sin(a_1 \pi x) \sin(a_2 \pi y). \quad (\text{D.20})$$

In Figure D.21 we depict this solution for $a_1 = 1$ and $a_2 = 4$.

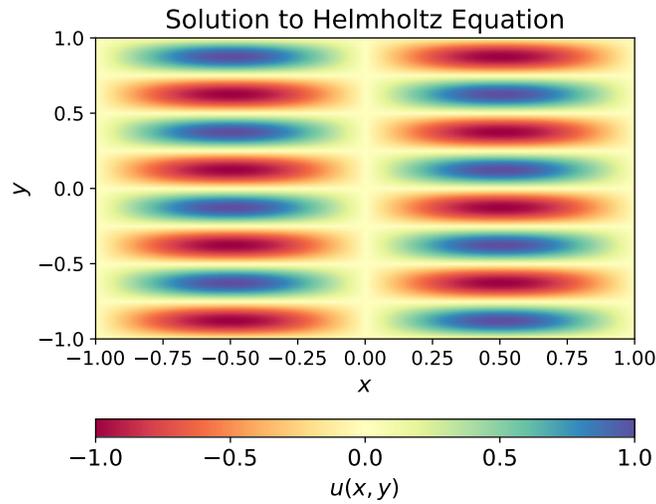


Figure D.21: Analytical solution of the Helmholtz equation on $[-1, 1]^2$ with $a_1 = 1$, $a_2 = 4$.

Poisson Equation. The 2-dimensional Poisson equation on the spatial domain $x \in [-1, 1]$, $y \in [-1, 1]$ is given by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2\pi^2 \omega^2 \sin(\omega\pi x) \sin(\omega\pi y). \quad (\text{D.21})$$

It is considered with homogeneous Dirichlet boundary conditions

$$u(-1, y) = u(1, y) = u(x, -1) = u(x, 1) = 0. \quad (\text{D.22})$$

The analytical solution of this equation is known and given by

$$u(x, y) = \sin(\omega\pi x) \sin(\omega\pi y). \quad (\text{D.23})$$

In Figure D.22 we depict this solution for $\omega \in \{1, 2, 4\}$.

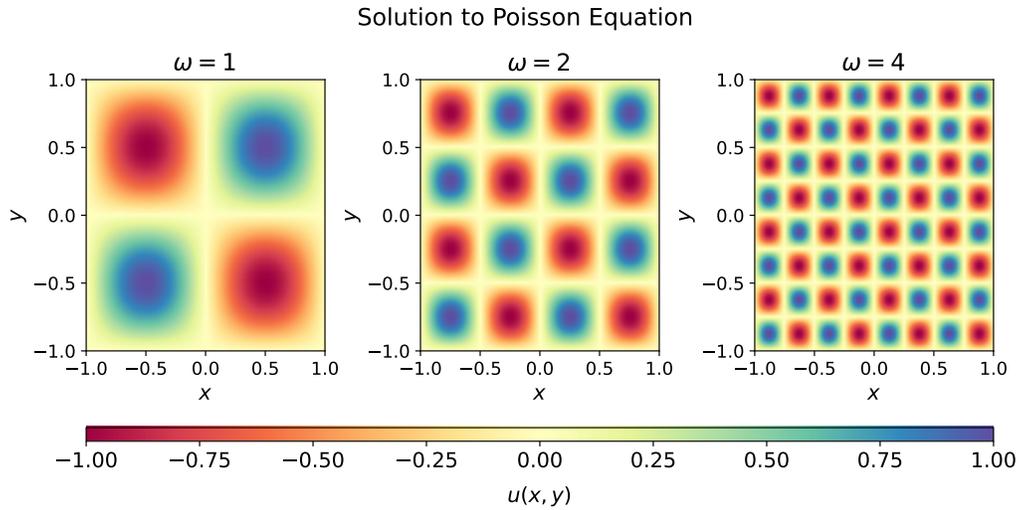


Figure D.22: Analytical solution of the Poisson equation on $[-1, 1]^2$ with $\omega = 1$ (left), $\omega = 2$ (middle) and $\omega = 4$ (right).

Appendix E. PirateNet Architecture

In this appendix, we provide a detailed description of the PirateNet architecture [23] employed in the benchmarks of Section 5, together with an explicit parameter count. As in Section 4.2, we consider a single input sample $\mathbf{x} \in \mathbb{R}^{d_I}$, where d_I is the number of input coordinates. If periodic boundary conditions are present, they are first enforced through the embedding of Eq. (39) in the main text, resulting in the transformed input $\tilde{\mathbf{x}} \in \mathbb{R}^{\tilde{d}_I}$.

The embedded coordinates are then passed through a RFF embedding layer. A trainable kernel $\mathbf{B} \in \mathbb{R}^{\tilde{d}_I \times 0.5d_H}$ is initialized from a Gaussian distribution $\mathcal{N}(0, s^2)$ with $s > 0$ ($s = 1$ is used for the benchmarks presented in this work), and the embedding is defined as

$$\Phi_j = \begin{bmatrix} \cos \left(\sum_{i=1}^{\tilde{d}_I} B_{ji} \tilde{x}_i \right) \\ \sin \left(\sum_{i=1}^{\tilde{d}_I} B_{ji} \tilde{x}_i \right) \end{bmatrix}, \quad (\text{E.1})$$

where $\Phi \in \mathbb{R}^{d_H}$. The resulting features are then processed through two MLP gates that generate the vectors $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{d_H}$:

$$U_j = \tanh \left(\sum_{i=1}^{d_H} w_{ji}^u \phi_i + b_j^u \right), \quad V_j = \tanh \left(\sum_{i=1}^{d_H} w_{ji}^v \phi_i + b_j^v \right). \quad (\text{E.2})$$

All MLP layers, including the two of Eq. (E.2), follow the Random Weight Factorization (RWF) formulation [22], with weights initialized using the standard Glorot scheme [43] and biases initialized at zero.

The adaptive skip connection is introduced through N identical blocks, each consisting of three MLP layers and a single gating parameter α (initialized at zero). Denoting the input to the l -th block by $\mathbf{x}^{(l)}$, with $\mathbf{x}^{(1)} = \Phi$, the forward pass is given by

$$f_j^{(l)} = \tanh \left(\sum_{i=1}^{d_H} w_{1,ji}^{(l)} x_i^{(l)} + b_{1,j}^{(l)} \right), \quad (\text{E.3})$$

$$z_{1,j}^{(l)} = f_j^{(l)} U_j + (1 - f_j^{(l)}) V_j, \quad (\text{E.4})$$

$$g_j^{(l)} = \tanh \left(\sum_{i=1}^{d_H} w_{2,ji}^{(l)} z_{1,i}^{(l)} + b_{2,j}^{(l)} \right), \quad (\text{E.5})$$

$$z_{2,j}^{(l)} = g_j^{(l)} U_j + (1 - g_j^{(l)}) V_j, \quad (\text{E.6})$$

$$h_j^{(l)} = \tanh \left(\sum_{i=1}^{d_{\text{H}}} w_{3,ji}^{(l)} z_{2,i}^{(l)} + b_{3,j}^{(l)} \right), \quad (\text{E.7})$$

$$x_j^{(l+1)} = \alpha h_j^{(l)} + (1 - \alpha) x_j^{(l)}. \quad (\text{E.8})$$

The output of the final PirateNet block, $\mathbf{x}^{(N+1)} \in \mathbb{R}^{d_{\text{H}}}$, is mapped to the network output through a linear layer

$$u_j = \sum_{i=1}^{d_{\text{H}}} w_{ji}^o x_i^{(N+1)}, \quad (\text{E.9})$$

where $\mathbf{u} \in \mathbb{R}^{d_{\text{O}}}$. This final layer is initialized using the same physics-informed least-squares procedure described in Section 4.2, but since this is a standard linear transformation, no re-indexing is required. If non-periodic boundary conditions are present, they are directly enforced at this stage by multiplying the network output with suitable boundary-shaping functions.

The total number of trainable parameters of the above architecture is

$$\begin{aligned} |\boldsymbol{\theta}| &= \underbrace{0.5 d_{\text{H}} \tilde{d}_{\text{I}}}_{\text{RFF Embeddings}} + \underbrace{2d_{\text{H}}(d_{\text{H}} + 2)}_{\text{MLP Gates}} + \underbrace{N[3d_{\text{H}}(d_{\text{H}} + 2) + 1]}_{\text{PirateNet Blocks}} + \underbrace{d_{\text{O}} d_{\text{H}}}_{\text{Output Layer}} \\ &= d_{\text{H}} \left[0.5 \tilde{d}_{\text{I}} + d_{\text{O}} + (d_{\text{H}} + 2)(3N + 2) \right] + N. \end{aligned} \quad (\text{E.10})$$

Note that, due to the RWF formulation of the layers, each MLP block with input dimension n_{in} and output dimension n_{out} contains $n_{\text{out}}(n_{\text{in}} + 2)$ trainable parameters, rather than $n_{\text{out}}(n_{\text{in}} + 1)$ as in standard MLPs.

CRediT authorship contribution statement

Spyros Rigas: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft. **Fotios Anagnostopoulos:** Data curation, Visualization, Writing – original draft. **Michalis Papachristou:** Data curation, Visualization, Writing – original draft. **Georgios Alexandridis:** Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

All data and source code used to produce the experimental results reported in this work are openly accessible at <https://github.com/srigas/RGA-KANs>.

References

- [1] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, *Nat. Rev. Phys.* 3 (2021) 422–440. doi:10.1038/s42254-021-00314-5.
- [2] J. D. Toscano, V. Oommen, A. J. Varghese, Z. Zou, N. A. Daryakenari, C. Wu, G. E. Karniadakis, From pinns to pikans: recent advances in physics-informed machine learning, *Mach. Learn. Comput. Sci. Eng* 1 (15) (2025). doi:10.1007/s44379-025-00015-1.
- [3] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation in machine learning: a survey, *J. Mach. Learn. Res.* 18 (153) (2018) 1–43.
URL <http://jmlr.org/papers/v18/17-468.html>
- [4] M. Raissi, Z. Wang, M. S. Triantafyllou, G. E. Karniadakis, Deep learning of vortex-induced vibrations, *J. Fluid Mech.* 861 (2019) 119–137. doi:10.1017/jfm.2018.872.

- [5] B. Reyes, A. A. Howard, P. Perdikaris, A. M. Tartakovsky, Learning unknown physics of non-newtonian fluids, *Phys. Rev. Fluids* 6 (7) (2021) 073301. doi:10.1103/PhysRevFluids.6.073301.
- [6] S. Cai, C. Gray, G. E. Karniadakis, Physics-informed neural networks enhanced particle tracking velocimetry: An example for turbulent jet flow, *IEEE Trans. Instrum. Meas.* 73 (2024) 1–9. doi:10.1109/TIM.2024.3398068.
- [7] S. Wang, S. Sankaran, P. Stinis, P. Perdikaris, Simulating three-dimensional turbulence with physics-informed neural networks, *arXiv preprint* (2025). doi:10.48550/arXiv.2507.08972.
- [8] K. Shukla, A. D. Jagtap, J. L. Blackshire, D. Sparkman, G. E. Karniadakis, A physics-informed neural network for quantifying the microstructural properties of polycrystalline nickel using ultrasound data: A promising approach for solving inverse problems, *IEEE Signal Process. Mag.* 39 (1) (2022) 68–77. doi:10.1109/MSP.2021.3118904.
- [9] Y. Diao, J. Yang, Y. Zhang, D. Zhang, Y. Du, Solving multi-material problems in solid mechanics using physics-informed neural networks based on domain decomposition technology, *Comput. Methods Appl. Mech. Eng.* 413 (2023) 116120. doi:10.1016/j.cma.2023.116120.
- [10] F. S. Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado, E. Kuhl, Physics-informed neural networks for cardiac activation mapping, *Front. Phys.* 8 (2020). doi:10.3389/fphy.2020.00042.
- [11] M. Yin, X. Zheng, J. D. Humphrey, G. E. Karniadakis, Non-invasive inference of thrombus material properties with physics-informed neural networks, *Comput. Methods Appl. Mech. Eng.* 375 (2021) 113603. doi:10.1016/j.cma.2020.113603.
- [12] Q. Chen, Q. Ye, W. Zhang, H. Li, X. Zheng, Tgm-nets: A deep learning framework for enhanced forecasting of tumor growth by integrating imaging and modeling, *Eng. Appl. Artif. Intell.* 126 (2023) 106867. doi:10.1016/j.engappai.2023.106867.
- [13] Y. Weng, D. Zhou, Multiscale physics-informed neural networks for stiff chemical kinetics, *J. Phys. Chem. A* 126 (45) (2022). doi:10.1021/acs.jpca.2c06513.

- [14] Y.-T. Liu, C.-Y. Wu, T. Chen, Y. Yao, Multi-fidelity surrogate modeling for chemical processes with physics-informed neural networks, in: A. C. Kokossis, M. C. Georgiadis, E. Pistikopoulos (Eds.), 33rd European Symposium on Computer Aided Process Engineering, Vol. 52, Elsevier, 2023, pp. 57–63. doi:10.1016/B978-0-443-15274-0.50010-X.
- [15] H. Gao, L. Sun, J. Wang, Super-resolution and denoising of fluid flow using physics-informed convolutional neural networks without high-resolution labels, *Phys. Fluids*. 33 (7) (2021). doi:10.1063/5.0054312.
- [16] L. Yang, D. Zhang, G. E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, *SIAM J. Sci. Comput.* 42 (1) (2020) A292–A317. doi:10.1137/18M1225409.
- [17] G. Cho, D. Zhu, J. J. Campbell, M. Wang, An lstm-pinn hybrid method to estimate lithium-ion battery pack temperature, *IEEE Access* 10 (2022) 100594–100604. doi:10.1109/ACCESS.2022.3208103.
- [18] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.* 378 (2019) 686–707. doi:10.1016/j.jcp.2018.10.045.
- [19] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, A. Courville, On the spectral bias of neural networks, in: *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97, 2019, pp. 5301–5310.
- [20] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, *SIAM J. Sci. Comput.* 43 (5) (2021) A3055–A3081. doi:10.1137/20M1318043.
- [21] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks, *Comput. Methods Appl. Mech. Eng.* 384 (2021) 113938. doi:10.1016/j.cma.2021.113938.
- [22] S. Wang, H. Wang, J. H. Seidman, P. Perdikaris, Random weight factorization improves the training of continuous neural representations, *arXiv preprint* (2022). doi:10.48550/arXiv.2210.01274.

- [23] S. Wang, B. Li, Y. Chen, P. Perdikaris, Piratenets: Physics-informed deep learning with residual adaptive networks, *J. Mach. Learn. Res.* 25 (402) (2024) 1–51.
URL <http://jmlr.org/papers/v25/24-0313.html>
- [24] S. Wang, X. Yu, P. Perdikaris, When and why pinns fail to train: A neural tangent kernel perspective, *J. Comput. Phys.* 449 (2022) 110768.
doi:10.1016/j.jcp.2021.110768.
- [25] S. Wang, S. Sankaran, P. Perdikaris, Respecting causality for training physics-informed neural networks, *Comput. Methods Appl. Mech. Eng.* 421 (2024) 116813. doi:10.1016/j.cma.2024.116813.
- [26] S. J. Anagnostopoulos, J. D. Toscano, N. Stergiopoulos, G. E. Karniadakis, Residual-based attention in physics-informed neural networks, *Comput. Methods Appl. Mech. Eng.* 421 (2024) 116805.
doi:10.1016/j.cma.2024.116805.
- [27] Z. Liu, Y. Wang, S. Vaidya, F. Ruelle, J. Halverson, M. Soljacic, T. Y. Hou, M. Tegmark, KAN: Kolmogorov–arnold networks, in: *The Thirteenth International Conference on Learning Representations*, 2025.
URL <https://openreview.net/forum?id=0zo7qJ5vZi>
- [28] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* 2 (5) (1989) 359–366.
doi:10.1016/0893-6080(89)90020-8.
- [29] A. K. Kolmogorov, On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition, *Doklady Akademii Nauk SSSR* 114 (1957) 369–373.
- [30] Z. Li, Kolmogorov-arnold networks are radial basis function networks, *arXiv preprint* (2024). doi:10.48550/arXiv.2405.06721.
- [31] S. S. Sidharth, A. R. Keerthana, R. Gokul, K. P. Anas, Chebyshev polynomial-based kolmogorov-arnold networks: An efficient architecture for nonlinear function approximation, *arXiv preprint* (2024). doi:10.48550/arXiv.2405.07200.

- [32] Q. Qiu, T. Zhu, H. Gong, L. Chen, H. Ning, Relu-kan: New kolmogorov-arnold networks that only need matrix addition, dot multiplication, and relu, arXiv preprint (2024). doi:10.48550/arXiv.2406.02075.
- [33] Z. Liu, P. Ma, Y. Wang, W. Matusik, M. Tegmark, Kan 2.0: Kolmogorov-arnold networks meet science, arXiv preprint (2024). doi:10.48550/arXiv.2408.10205.
- [34] Y. Wang, J. W. Siegel, Z. Liu, T. Y. Hou, On the expressiveness and spectral bias of KANs, in: The Thirteenth International Conference on Learning Representations, 2025.
- [35] K. Shukla, J. D. Toscano, Z. Wang, Z. Zou, G. E. Karniadakis, A comprehensive and fair comparison between mlp and kan representations for differential equations and operator networks, *Comput. Methods Appl. Mech. Eng.* 431 (2024) 117290. doi:10.1016/j.cma.2024.117290.
- [36] Y. Wang, J. Sun, J. Bai, C. Anitescu, M. S. Eshaghi, X. Zhuang, T. Rabczuk, Y. Liu, Kolmogorov–arnold-informed neural network: A physics-informed deep learning framework for solving forward and inverse problems based on kolmogorov–arnold networks, *Comput. Methods Appl. Mech. Eng.* 433 (2025) 117518. doi:10.1016/j.cma.2024.117518.
- [37] S. Rigas, M. Papachristou, T. Papadopoulos, F. Anagnostopoulos, G. Alexandridis, Adaptive training of grid-dependent physics-informed kolmogorov-arnold networks, *IEEE Access* 12 (2024) 176982–176998. doi:10.1109/ACCESS.2024.3504962.
- [38] N. A. Daryakenari, K. Shukla, G. E. Karniadakis, Representation meets optimization: Training pinns and pikans for gray-box discovery in systems pharmacology, arXiv preprint (2024). doi:10.48550/arXiv.2504.07379.
- [39] A. Kashefi, Kolmogorov–arnold pointnet: Deep learning for prediction of fluid fields on irregular geometries, *Comput. Methods Appl. Mech. Eng.* 439 (2025) 117888. doi:10.1016/j.cma.2025.117888.
- [40] J. D. Toscano, T. Käufer, Z. Wang, M. Maxey, C. Cierpka, G. E. Karniadakis, Aivt: Inference of turbulent thermal convection from measured

- 3d velocity data by physics-informed kolmogorov-arnold networks, *Sci. Adv.* 11 (19) (2025) eads5236. doi:10.1126/sciadv.ads5236.
- [41] A. Pal, D. Das, Understanding the limitations of b-spline KANs: Convergence dynamics and computational efficiency, in: *NeurIPS 2024 Workshop on Scientific Methods for Understanding Deep Learning*, 2024.
URL <https://openreview.net/forum?id=yPE7S57uei>
- [42] J. D. Toscano, L.-L. Wang, G. E. Karniadakis, Kkans: Kurkova-kolmogorov-arnold networks and their learning dynamics, *Neural Networks* 191 (2025) 107831. doi:10.1016/j.neunet.2025.107831.
- [43] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feed-forward neural networks, in: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Vol. 9, 2010, pp. 249–256.
URL <https://proceedings.mlr.press/v9/glorot10a.html>
- [44] S. Rigas, D. Verma, G. Alexandridis, Y. Wang, An empirical investigation of initialization strategies for kolmogorov-arnold networks, in: *ICML 2025 Workshop on Methods and Opportunities at Small Scale*, 2025.
URL <https://openreview.net/forum?id=eC285SNCiW>
- [45] A. A. Howard, B. Jacob, S. H. Murphy, A. Heinlein, P. Stinis, Finite basis kolmogorov-arnold networks: domain decomposition for data-driven and physics-informed problems, *arXiv preprint* (2024). doi:10.48550/arXiv.2406.19662.
- [46] A. A. Howard, B. Jacob, P. Stinis, Multifidelity kolmogorov-arnold networks, *arXiv preprint* (2024). doi:10.48550/arXiv.2410.14764.
- [47] S. Wang, S. Sankaran, H. Wang, P. Perdikaris, An expert’s guide to training physics-informed neural networks, *arXiv preprint* (2023). doi:10.48550/arXiv.2308.08468.
- [48] S. J. Anagnostopoulos, J. D. Toscano, N. Stergiopoulos, G. E. Karniadakis, Learning in pinns: Phase transition, diffusion equilibrium, and generalization, *Neural Networks* 193 (2026) 107983. doi:<https://doi.org/10.1016/j.neunet.2025.107983>.

- [49] A. A. Howard, S. Qadeer, A. W. Engel, A. Tsou, M. Vargas, T. Chiang, P. Stinis, The conjugate kernel for efficient training of physics-informed deep operator networks, in: ICLR 2024 Workshop on AI4DifferentialEquations In Science, 2024.
- [50] M. A. Nabian, R. J. Gladstone, H. Meidani, Efficient training of physics-informed neural networks via importance sampling, *Comput.-Aided Civ. Infrastruct. Eng.* 36 (8) (2021) 962–977. doi:10.1111/mice.12685.
- [51] C. Wu, M. Zhu, Q. Tan, Y. Kartha, L. Lu, A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks, *Comput. Methods Appl. Mech. Eng.* 403 (2023) 115671. doi:10.1016/j.cma.2022.115671.
- [52] G. K. R. Lau, A. Hemachandra, S.-K. Ng, B. K. H. Low, PINNACLE: PINN adaptive collocation and experimental points selection, in: The Twelfth International Conference on Learning Representations, 2024.
- [53] Z. Zhang, J. Li, B. Liu, Annealed adaptive importance sampling method in pinns for solving high dimensional partial differential equations, *J. Comput. Phys.* 521 (2025) 113561. doi:10.1016/j.jcp.2024.113561.
- [54] S. Rigas, M. Papachristou, jaxkan: A unified jax framework for kolmogorov-arnold networks, *Journal of Open Source Software* 10 (108) (2025) 7830. doi:10.21105/joss.07830.
- [55] B. Jacob, A. A. Howard, P. Stinis, Spikans: Separable physics-informed kolmogorov-arnold networks, *arXiv preprint* (2024). doi:10.48550/arXiv.2411.06286.
- [56] L. F. Guilhoto, P. Perdikaris, Deep learning alternatives of the kolmogorov superposition theorem, in: The Thirteenth International Conference on Learning Representations, 2025.
- [57] Z. Gao, G. E. Karniadakis, Scalable bayesian physics-informed kolmogorov-arnold networks, *arXiv preprint* (2025). doi:10.48550/arXiv.2501.08501.
- [58] L. D. McClenny, U. M. Braga-Neto, Self-adaptive physics-informed neural networks, *J. Comput. Phys.* 474 (2023) 111722. doi:10.1016/j.jcp.2022.111722.

- [59] W. Chen, A. A. Howard, P. Stinis, Self-adaptive weights based on balanced residual decay rate for physics-informed neural networks and deep operator networks, *J. Comput. Phys.* (2025) 114226doi:10.1016/j.jcp.2025.114226.
- [60] Y. LeCun, L. Bottou, G. B. Orr, K.-R. Müller, Efficient backprop, in: *Neural Networks: Tricks of the Trade*, Springer Berlin Heidelberg, 1998, pp. 9–50. doi:10.1007/3-540-49430-8_2.
- [61] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, *PyTorch: an imperative style, high-performance deep learning library*, Curran Associates Inc., 2019.
- [62] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: *International Conference on Learning Representations (ICLR)*, 2015.
- [63] N. Sukumar, A. Srivastava, Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks, *Computer Methods in Applied Mechanics and Engineering* 389 (2022) 114333. doi:10.1016/j.cma.2021.114333.
- [64] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, R. Ng, Fourier features let networks learn high frequency functions in low dimensional domains, in: *Advances in Neural Information Processing Systems*, Vol. 33, 2020, pp. 7537–7547.
- [65] N. Tishby, N. Zaslavsky, Deep learning and the information bottleneck principle, in: *2015 IEEE Information Theory Workshop (ITW)*, 2015, pp. 1–5. doi:10.1109/ITW.2015.7133169.
- [66] R. Shwartz-Ziv, N. Tishby, Opening the black box of deep neural networks via information, *arXiv preprint* (2017). doi:10.48550/arXiv.1703.00810.
- [67] Z. Goldfeld, Y. Polyanskiy, The information bottleneck problem and its applications in machine learning, *IEEE Journal on Selected Areas in Information Theory* 1 (1) (2020) 19–38. doi:10.1109/JSAIT.2020.2991561.

- [68] R. Shwartz-Ziv, Information flow in deep neural networks, arXiv preprint (2022). doi:10.48550/arXiv.2202.06749.
- [69] B. Dherin, M. Munn, M. Rosca, D. G. Barrett, Why neural networks find simple solutions: The many regularizers of geometric complexity, in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022.
- [70] E. Kiyani, K. Shukla, J. F. Urbán, J. Darbon, G. E. Karniadakis, Optimizing the optimizer for physics-informed neural networks and kolmogorov-arnold networks, Computer Methods in Applied Mechanics and Engineering 446 (2025) 118308. doi:<https://doi.org/10.1016/j.cma.2025.118308>.
- [71] S. Wang, A. K. Bhartari, B. Li, P. Perdikaris, Gradient alignment in physics-informed neural networks: A second-order optimization perspective, arXiv preprint (2025). doi:10.48550/arXiv.2502.00604.
- [72] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang, JAX: composable transformations of Python+NumPy programs (2018).
URL <http://github.com/jax-ml/jax>
- [73] S. Wang, P. Perdikaris, Predictive intelligence lab: Jax-pi, <https://github.com/PredictiveIntelligenceLab/jaxpi>, accessed: 2025-07-23 (2023).