Physics-informed neural networks to solve inverse problems in unbounded domains

G. Pérez-Bernal[1], O. Rincón-Cardeño[1], S. Montoya-Noguera[2], N. Guarín-Zapata[1*]

[1*]Mathematical Applications in Science and Engineering Research Group, School of Applied Sciences and Engineering, Universidad EAFIT, Medellín, Colombia.

[2]Nature and Cities Research Group, School of Applied Sciences and Engineering, Universidad EAFIT, Medellín, Colombia.

*Corresponding author(s). E-mail(s): nguarinz@eafit.edu.co; Contributing authors: gperezb1@eafit.edu.co; orincon@eafit.edu.co; smontoyan@eafit.edu.co;

Abstract

Inverse problems are extensively studied in applied mathematics, with applications ranging from acoustic tomography for medical diagnosis to geophysical exploration. Physics-informed neural networks (PINNs) have emerged as a powerful tool for solving such problems, while Physics-informed Kolmogorov–Arnold networks (PIKANs) represent a recent benchmark that, in certain problems, promises greater interpretability and accuracy compared to PINNs, due to their nature, being constructed as a composition of polynomials. In this work, we develop a methodology for addressing inverse problems in infinite and semi-infinite domains. We introduce a novel sampling strategy for the network's training points, using the negative exponential and normal distributions, alongside a dual-network architecture that is trained to learn the solution and parameters of an equation with the same loss function. This design enables the solution of inverse problems without explicitly imposing boundary conditions, as long as the solutions tend to stabilize when leaving the domain of interest. The proposed architecture is implemented using both PINNs and PIKANs, and their performance is compared in terms of accuracy with respect to a known solution as well as computational time and response to a noisy environment. Our results demonstrate that, in this setting, PINNs provide a more accurate and computationally efficient solution, solving the inverse problem **1, 000** times faster and in the same order of magnitude, yet with a lower relative error than PIKANs.

Keywords: Scientific Machine Learning, Physics Informed Neural Networks, Kolmogorov Arnold Networks, Inverse Problems, Poisson Equation

1 Introduction

Studying differential equations is crucial in physics, as it allows us to model and predict the behavior of dynamic systems in various fields where changes in a function can be modeled more simply than the function itself [1]. A differential equation can be approached in two distinct ways: directly or inversely. The direct approach aims to compute the solution of the equation given its parameters and boundary conditions, whereas the inverse approach focuses on identifying the underlying parameters or causes from a set of observations, as well as the equation itself, that is, determining the values associated with the governing differential equation [2]. For instance, when working with the Poisson equation, the associated parameter that the inverse problem solves can be the thermal conductivity or the electrical permittivity of a system, depending on how it is posed.

From time to time, one encounters an inverse problem that has no information (or limited information) around its boundaries. For instance, in geophysics, full wave inversion can provide information about the materials that are found below the surface [3] without any need for direct excavation. Nevertheless, obtaining data along interior boundaries is not feasible in practice, as it would require intrusive interventions such as drilling or digging. This limitation motivates the study and development of inverse problem methodologies in unbounded domains.

In addition to classical numerical methods, such as the adjoint method [4], a new approach was proposed in 2019 to solve inverse problems: physics-informed neural networks (PINNs) [5]. This framework allows approximating the solution of partial differential equations and inverse problems using a Deep Neural Network (DNN) with a loss function to minimize based on physical laws, as well as observations of solutions of the equation.

In 2024, Kolmogorov–Arnold Networks (KAN) were introduced as a novel alternative to traditional DNNs, offering enhanced interpretability and, in certain cases, improving accuracy in function approximation tasks [6]. Their effectiveness is rooted in the Kolmogorov-Arnold representation theorem, which states that any continuous multivariate function can be expressed as a finite sum of continuous univariate functions. This naturally raises the question: can physics-informed Kolmogorov–Arnold Networks (PIKANs) outperform PINNs in approximating the inverse Poisson problem on an unbounded domain? In this work, we introduce a novel strategy for sampling training points, develop a dual-network framework for addressing the inverse problem, and conduct a comparative analysis of PINNs and PIKANs for the inverse Poisson problem. Our evaluation includes metrics such as solution accuracy and the ability of each method to generalize across the domain.

2 Theoretical framework

2.1 Inverse problems and Poisson equation

Inverse problems consist in determining a system's parameters, given their structure and direct or indirect observations. In the context of differential equations, they arise when the governing physical laws are expressed through operators, but certain coefficients or parameters remain unknown [2]. Given partial observations of a solution, together with knowledge of the underlying equations, the inverse problem seeks to reconstruct the complete set of parameters that characterize the system.

For instance, Poisson's equation provides a highly general mathematical framework for describing a wide range of physical phenomena across multiple areas of physics. Poisson's inverse problem in $\mathbb{R}^3$ then has as a goal of finding the variable parameter $k(x, y)$, as seen on equation 1, given the structure of the differential equation, as well as observations from solutions of the differential equation and/or of the k parameter in the desired domain. Note that, in solving the inverse problem (determining the coefficient k), one simultaneously solves the associated forward problem for the state variable u. Specifically, both quantities are linked through the governing differential equation

$$\nabla \cdot (k\nabla u) = f. \tag{1}$$

In a more practical sense, the value of k can account for electrical permittivity, thermic conductivity, and even Young's modulus, depending on the problem. In some of the areas of interest, one can encounter an infinite domain, a domain that has no bounds, as information regarding boundaries is either non existent or not available.

In the Poisson equation, the source term f denotes the underlying distribution that generates the potential field. It characterizes the spatial localization and intensity of the sources driving the system. For instance, in a electrostatics problem, f corresponds to the charge distribution, describing how electric charges are arranged in space and how they give rise to the associated electric potential.

2.2 Neural Networks Architectures

2.2.1 Deep Neural Networks

A DNN is a computational model composed of multiple layers of interconnected nodes, or "neurons," that transform input data into output predictions through successive nonlinear operations [7]. Each neuron applies a fixed nonlinear activation function (choices include the rectified linear unit (ReLU), sigmoid, or hyperbolic tangent) to a weighted sum of its inputs. The connections are associated with trainable parameters, typically called weights and biases, which are trained to determine how information flows through the network. The training process consists of optimizing these parameters so that the network can approximate an unknown function. This optimization is usually performed by minimizing a loss function, such as the Mean Squared Error (MSE) in supervised learning tasks, using gradient-based methods like stochastic gradient descent. It is common to denote the network configuration as an ordered pair (*Number of hidden layers*, *Number of neurons per hidden layer*).

2.2.2 Kolmogorov-Arnold Networks

KANs are neural architectures motivated by the Kolmogorov-Arnold representation theorem, which states that any multivariate continuous function can be written as a finite sum (thus product) of univariate functions [8]. In contrast to DNNs where fixed activation functions are assigned to nodes, KANs replace linear weights with *learnable univariate functions* assigned to the edges of the network. These functions are commonly parameterized as splines, enabling each edge to adaptively model nonlinear transformations of its inputs [6].

In practice, various spline families can be used to parameterize these edge functions. However, following [6] and [9], the most widely adopted choice is B-splines with a fixed number of interpolation points. This design not only improves expressive power but also provides a higher degree of interpretability, as the learned transformations are explicitly represented by low-dimensional polynomial bases.

KANs have shown particular promise in domains such as data fitting and the numerical solution of Partial Differential Equations (PDEs), where they have achieved higher accuracy and efficiency compared to DNNs, however, the scientific machine learning community is still divided, as there have also been several applications in solving differential equations where a DNN outperforms a KAN [9, 10].

Formally, the output of a KAN with L layers is given by a composition of univariate spline functions:

$$\mathrm{KAN}(x) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_0)(x),$$

where each Φ_i denotes a collection of spline-based transformations applied to the inputs of layer i. In practice, these spline transformations are defined by two main hyperparameters: the spline *degree* k and the *grid* of knots. The parameter k controls the polynomial order of the B-splines used in the transformation. The grid specifies the placement and number of knots across the input domain. A finer grid allows the KAN to capture more detailed patterns, whereas a coarser grid enforces smoother approximations. Consequently, it has also been noted that KAN need significantly less parameters to converge to a solution, but they take way longer to train [11]. In other words, a KAN is trained to learn nonlinear activation functions, whereas a DNN is trained to learn scalar weights and biases.

Finally, since splines are polynomial pieces, KAN neurons inherit the algebraic closure of polynomials under composition, enhancing their interpretability and mathematical transparency [6].

2.3 Physics informed neural networks (PINNs) for inverse problems

The idea of using neural networks to solve differential equations comes from way back in the 1990's. It was first introduced by [12], proposing to include physical laws into the loss function that is minimized during training of neural networks. With the rise of modern deep learning frameworks such as PyTorch [13] and TensorFlow [14], together with significant advances in neural network architectures and optimization, PINNs were reintroduced and gained momentum around 2019 [5].

In accordance to [5], inverse problems can be solved using neural networks by minimizing a loss function built as a combination of the residual of a differential equation and observed data from either the solution or the parameters.

The residual of a differential equation measures how much a proposed solution fails to satisfy an equation. Specifically, it is the result of substituting the approximate solution into the differential equation and computing the difference between the left-hand side and the right-hand side. A residual of zero indicates that the solution exactly satisfies the equation, while a nonzero residual reflects the error or mismatch. For Poisson's equation, for an approximate solution $\hat{u}$, the residual is defined as:

$$\mathcal{R} = \nabla \cdot (k \nabla \hat{u}) - f. \tag{2}$$

Thus, as shown on equation 3, to approximate an inverse problem, a PINN minimizes the residual of a PDE and the MSE of observed solutions and predicted solutions. This loss function is perfectly applicable to KANs, and PIKANs are described just that way. The training points and the interior data used to minimize this function tend to be sampled using uniform distributions and literature that approaches this issue is limited [15] [16]. Given N_{PDE} training points and N_{obs}, the loss function is defined as:

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \|\mathcal{R}(x_i)\|^2 \; + \frac{1}{N_{obs}} \sum_{i=0}^{N_{obs}} ||\hat{u}(x_i) - u(x_i)||^2. \tag{3}$$

Now, when working in a domain that has information in the boundaries, an additional term can be included. When encountering Dirichlet boundary conditions, an extra term on equation 3 can be added as an MSE to include the extra insights in the boundries, as follows:

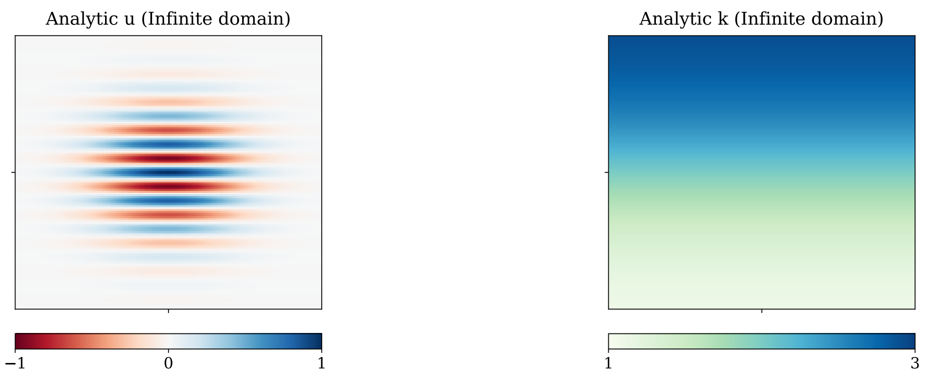
$$\mathcal{L}^*(\theta) = \mathcal{L}(\theta) + \frac{1}{N_{bnd}} \sum_{i=0}^{N_{BND}} ||\hat{u}_{bnd}(x_i) - u_{bnd}(x_i)||^2.$$

A loss function can be minimized using various Algorithms. Adam is an stochastic, adaptive optimization algorithm that adjusts learning rates using estimates of first and second moments of the gradients, making it efficient for training deep networks [17]. L-BFGS is a memory-efficient quasi-Newton method that approximates second-order information to optimize smooth functions with faster convergence than gradient descent [18], however, it has been proved to be significantly slower in runtime [19], and considering that it is not stochastic, does not perform well under noisy environments.

3 Methodology

3.1 Problem selection

Two problems are considered: one in an infinite domain, where there are no boundary conditions and one in a semi infinite domain, where there is one Dirichlet boundary condition in parts of the domain, and other parts remain unbounded.



(a) Solution for u. $\alpha = 0.5$, $\beta = 10$

(b) Solution for k with $\varepsilon = 1$

Fig. 1: Analytic solutions for u and k on an infinite domain

3.1.1 Infinite Domain

Using the method for manufactured solutions [20], we can intentionally construct the equation that we wish to solve, by taking a desired value for k and u and forcing the source term f with those values so that Poisson's equation is fulfilled. We let the analytical solution of the equation and the real value of k to be as follows:

$$u(x,y) = e^{-\alpha(x^2+y^2)}\cos(\beta y), \quad k(y) = -1 + \frac{2}{1+e^{-y/\varepsilon}}$$

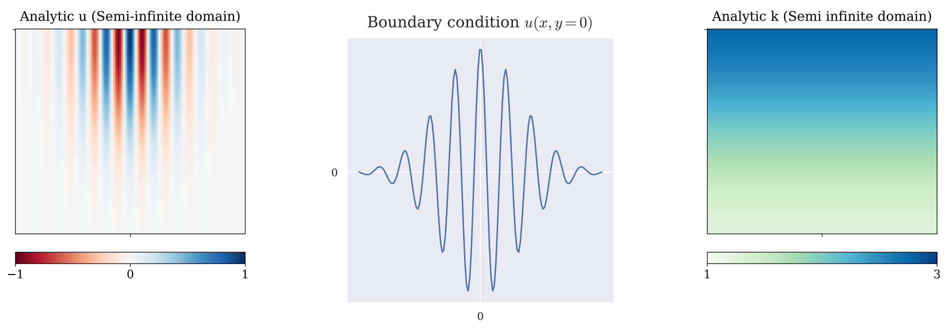
Figures 1a and 1b show the analytic solutions for the u and k proposed. Note that the solutions tend to stabilize when they tend to infinity.

3.1.2 Semi Infinite Domain

Using the same method applied for the infinite domain we can construct the equation that we wish to solve by taking a desired value for k and u and constructing the source term f with those values. To grant a semi infinite domain, we define the boundary Ω_y, along the $y = 0$ axis. We let the analytical solution of the equation and the real value of k to be as follows:

$$\begin{cases} u(x,y) = e^{-\alpha(x^2+y^2)}\cos(\beta x), & (x,y) \in \Omega, \\ u(x,0) = e^{-\alpha x^2}\cos(\beta x), & x \in \partial\Omega_y, \end{cases}$$

$$k(y) = -1 + \frac{2}{1+e^{(-y+1.5)/\varepsilon}}$$



(a) Solution for u with $\alpha = 0.5$, $\beta = 10$

(b) Boundary condition for $u(x, y = 0)$

(c) Solution for k with $\varepsilon = 0.75$

Fig. 2: Analytic solutions for u and k in a semi infinite domain

Figures 2a and 2c show the analytic solutions for the u and k proposed, as well as the boundary contidion set for u. These figures are set to similiar equations as in the infinite domain case, for reproducibility reasons.

3.2 Sampling strategy

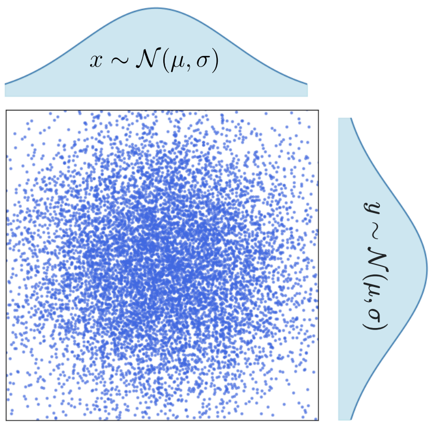
As mentioned, the domain of interest for the inverse problem is unbounded. Therefore, if training points are sampled uniformly, their density will be evenly distributed across the domain. As a result, the gradients may not converge effectively, leading to poor training performance and failure of generalization in areas outside of the training domain.

For an infinite domain, we propose sampling the **training** points using a normal distribution with mean in the zone of interest. That way, for equations where the solution tends to stabilize when going to infinity, there are less training points as one moves from the zone of interest. However, since solutions are assumed to become constant when reaching infinity, the gradients will become zero, and this sampling approach will help the network to generalize outside the training domain. Figure 3a provides a graphic explanation of the sampling strategy used. On the other hand, for the semi infinite case, there is available information on one boundary (in our case $y = 0$), so the sampling for the training points in y is done with an exponential function, as shown on Figure 3b. We propose to solve the problem using $10,000$ training points.

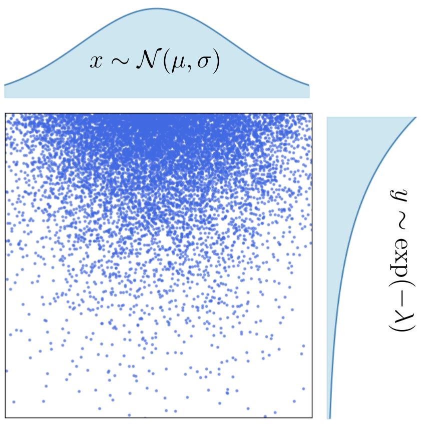
It is important to remark that this strategy is implemented to sample the training points, and not to sample the observed data from u and k that will aid in the solution of the problem. These observations are to be sampled using a uniform distribution.

3.3 Network Architecture

The network designs include two fully connected networks, $\mathcal{NN}^u$, which solves the direct problem and $\mathcal{NN}^k$, trained to solve the inverse problem, inspired by the works of



(a) Sampling strategy for training points on an infinite domain using a normal distribution



(b) Sampling strategy for a semi infinite domain using a normal and an exponential distribution

Fig. 3: Proposed training points sampling

[21]. Both networks are trained in parallel using the same loss function. The reasoning behind creating two networks instead of one fully connected network with two outputs comes from the idea of not having the solution for u bias the solution for k.

Figure 4 shows the structure of the networks. Note that the input of the network is a position vector (x, y), the output of $\mathcal{NN}^k$ gives us the value of k and the output of $\mathcal{NN}^u$ gives us the value of u. The output of each network is derived using automatic differentiation [22]. The loss function to be minimized is then constructed by taking the residual and observed data from both u and k. Both networks are trained at the same time, with the same loss function, in order to influence by some degree the solution.

3.4 Loss function

The loss function to minimize is formed by minimizing the residual of Poisson's equation (see eq. 2) and the MSE for observations of both u and k. Given N_{pde} training points and N_{obs} observations of the solution, the losses that will make up the loss function are presented in table 1. The components are then added and multiplied by a scalar factor λ_i that adds mathematical importance to the different losses. The different values of λ_i are selected by trial and error.

3.5 Two step minimization of the Loss function

The training stage was defined to be carried out in two stages. Initially, the Adam optimizer was employed to rapidly explore the loss landscape and bring the network parameters close to a local minimum. Then, the L-BFGS algorithm, a quasi-Newton method, was applied to refine the solution and achieve higher accuracy. This hybrid strategy leverages the robustness of Adam in navigating complex, high-dimensional

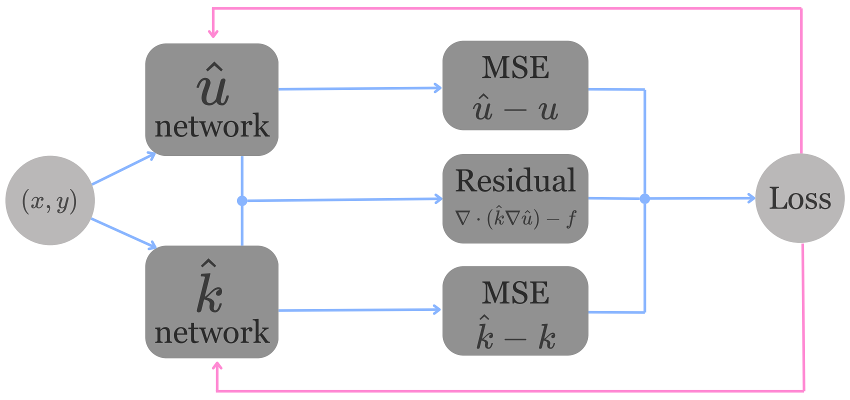


Fig. 4: Dual network architecture: One network learns the value of u whereas one diffrent network learns the value of k. Both networks are then trained using the same loss (residual + data)

Description	**Equation**
PDE residual loss	$\mathcal{L}_{\text{PDE}}(\theta) = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \left\| \mathcal{R}(x_i) \right\|^2$
Data loss for u	$\mathcal{L}_u = \frac{1}{N_{\text{obs}}} \sum_{i=1}^{N_{\text{obs}}} \left\| \hat{u}(x_i) - u_{\text{real}}(x_i) \right\|^2$
Data loss for k	$\mathcal{L}_k = \frac{1}{N_{\text{obs}}} \sum_{i=1}^{N_{\text{obs}}} \left\| \hat{k}(x_i) - k_{real}(x_i) \right\|^2$
Boundary loss for u	$\mathcal{L}_{u_{bnd}} = \frac{1}{N_{\text{bnd}}} \sum_{i=1}^{N_{\text{bnd}}} \left\| \hat{u}(x_i) - u_{real}(x_i) \right\|^2 \quad \forall x \in \partial\Omega$
Combined loss (infinite domain)	$\mathcal{L}_{\text{infinite}} = \lambda_1 \mathcal{L}_{\text{PDE}} + \lambda_2 \mathcal{L}_u + \lambda_3 \mathcal{L}_k$
Combined loss (semi-infinite domain)	$\mathcal{L}_{\text{semi-infinite}} = \lambda_1 \mathcal{L}_{\text{PDE}} + \lambda_2 \mathcal{L}_u + \lambda_3 \mathcal{L}_k + \lambda_4 \mathcal{L}_{u_{bnd}}$

Table 1: Loss function components and combined formulations for the inverse problem.

spaces while being a stochastic method and the precision of L-BFGS in fine-tuning the model parameters, as it is a second order method [23].

3.6 Experiments carried

The following experiments were done to assess and compare the ability of both PINNs and PIKANs to solve Poisson's inverse problem. The training for every network was performed on a single *NVIDIA A10G Tensor Core GPU with 24GB of VRAM*. We used $10,000$ training points and $5,000$ data points. Scenarios training PINNs and

PIKANs where the data observed has 5%, 10% and 15% noise were also executed. Apart from the observations of the inside domain, for semi-infinite problems, 500 boundary observations were also taken.

3.6.1 PINN configurations

We evaluated architectures with varying numbers of hidden layers and neurons per layer, considering the following network setups: $(4, 8)$, $(8, 16)$, $(16, 32)$, $(32, 64)$, and $(64, 128)$. These variations allow us to study the impact of increasing network capacity on the accuracy and stability of the solution. Each PINN was configured with *tanh* activation function, and was trained using ADAM for $15,000$ epochs with a 10^{-4} learning rate, and then for $1,500$ with a 0.5 learning rate using L-FBGS.

3.6.2 PIKAN configurations

We evaluated architectures with varying numbers of hidden layers and neurons per layer, considering the following setups: (2,4),(3,6),(4,8),(5,10), (6,12). These variations allow us to study the impact of increasing network capacity on the accuracy and stability of the solution. Each PIKAN was configured with the parameter *grid = 3* and *k = 3*. Training using ADAM for $1,500$ epochs with a 10^{-4} learning rate, and then for 500 with a 0.5 learning rate using L-FBGS. The PIKAN is trained for less epochs than the PINN because due to the nature of KANs, they take less epochs to converge, however each epoch takes a longer time to be completed [6] [9]. the PIKAN also has less hpyerparameters because KANs have been shown to need less parameters to converge [11]. Thus, the criterion to choose the network configurations for which the simulations are to be done was to choose hyperparameters such that training times are comparable between PINNs and PIKANs.

4 Results

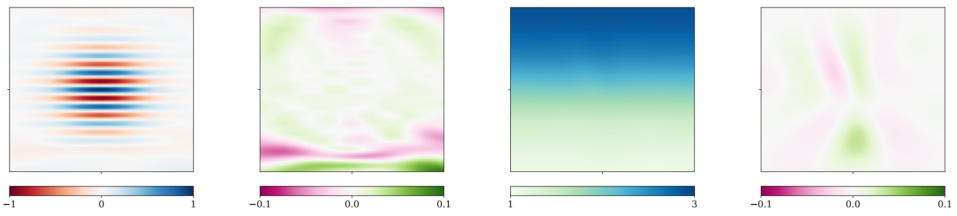
The following section presents the results obtained for simulations of the Poisson equation on both infinite and semi-infinite domains. All approximated values are non-dimensional. The simulations for the infinite domain were carried out on the square $[-3, 3] \times [-3, 3]$, and those for the semi-infinite domain on $[-3, 3] \times [0, 3]$. To evaluate the approximate solutions against the analytic ones, we used a variation of the relative error in which the absolute value is omitted. This allows us to capture the "sign" of the error, which is particularly relevant in wave-related problems.

4.1 Infinite domain

Figure 5 shows the approximation of u and k achieved by the PINN. Note that the error does not necesarily increase when approaching the boundary of observed data. In fact, as shown in figure 7, the PINN proves to generalize in u and specially in k, showing that it can learn outside the training region. In figure 7, the the domain for which there are training points available is the area enclosed by the purple squares. Figure 6 shows the approximations obtained using the PIKAN that performed best. By comparing the error profiles for both u and k obtained with PINN and PIKAN

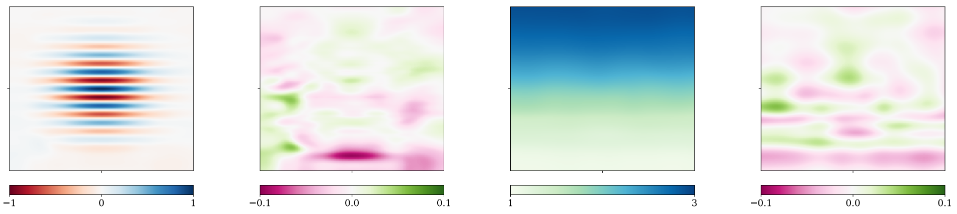
(Figures 5b, 5d, 6b, 6d), we observe that both methods yield errors of the same order of magnitude. However, sprecially in the case of the k approximation, the PIKAN model exhibits noticeably larger and more spatially dispersed errors across the domain.

When experimenting with both PINNs and PIKANs, it was found that for the proposed configurations, not only were the PINNs trained in a considerably shorter time, but the relative error achieved for the approximation of k was one order of magnitude lower than with the PIKAN, as shown in figure 8. By analyzing figure 8, it can be noticed that when increasing the PINNs parameters to the $(64, 128)$ configuration, the relative error increases to up to 10^2, due to overfitting, when a model learns irrelevant patterns from the training data instead of generalizing, which increases relative error because its predictions perform poorly on unseen data. This phenomenon can happen when there is an excess of parameters or when the number of epochs defined is not enough for the network to learn the function.



(a) Approx. for u (b) u relative error (c) Approx. for k (d) k relative error

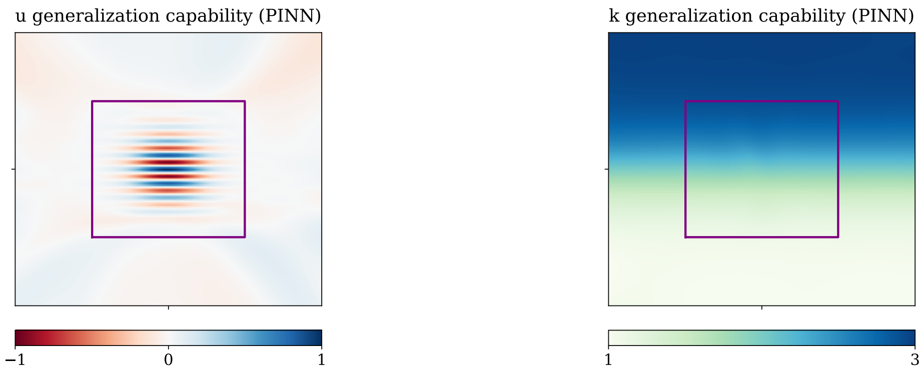
Fig. 5: PINN Predictions for u and k using a $(16, 32)$ network configuration



(a) Approx. for u (b) u relative error (c) Approx. for k (d) k relative error

Fig. 6: PIKAN Predictions for u and k using a $(3, 6)$ KAN network configuration with grid=3 and $k = 3$

Figure 9 shows an experiment in which the number of training epochs, the network configuration, and the number of training points were kept fixed, while only the number



(a) u capacity of generalizing (b) k capacity of generalizing

Fig. 7: Network generalization capacity on an infinite domain

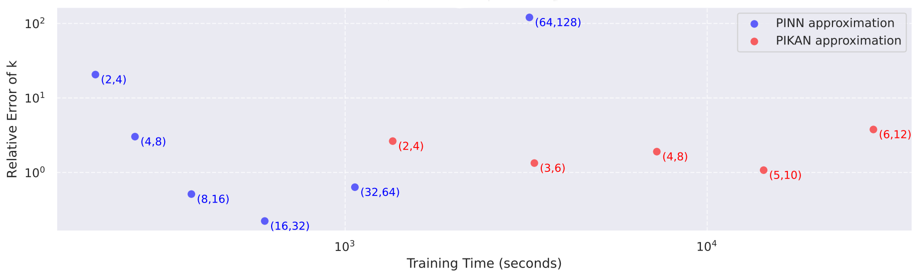


Fig. 8: log log plot of different configurations of PINNs and PIKANs approximation on an infinite domain. PIKANs tend to take much more to approximate a worse solution

of available observations for u and k varied. As expected, the error decreases as the number of training points increases, up to a point where it begins to level off. Notably, increasing the number of available points does not significantly raise the computation time, since the time complexity in neural networks depends more on the network architecture itself than on the size of the dataset [24].

4.2 Semi-infinite domain

Figure 10 shows the approximation of u and k achieved by the PINN on a semi-infinite domain, whereas 11 shows the approximation and error obtained by the best performing KAN. It is interesting to see that the PIKAN proves to have way higher errors when approaching the end of the computational domain created, as one can see in figures 11b and 11d. As shown in figure 12, the PINN proves to somewhat generalize

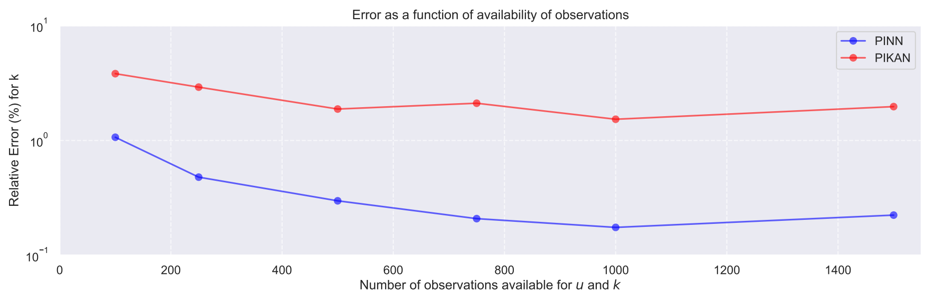
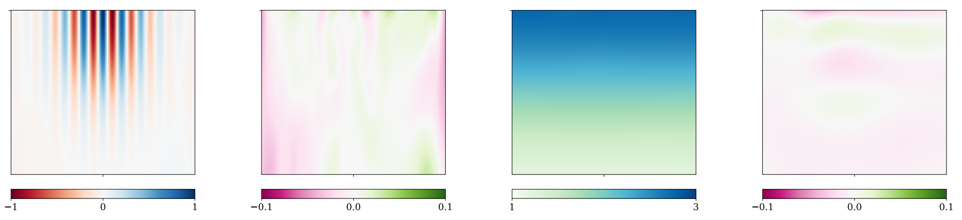


Fig. 9: log plot of error obtained by PINNs and PIKANs approximating k on an infinite domain, while varying the number of available information.

in u and specially generalize in k, proving that it can solve the inverse problem outside the training region. Interestingly, even though there is information available in one of the boundaries, the errors for both u and k seem to be lower on the infinite case.

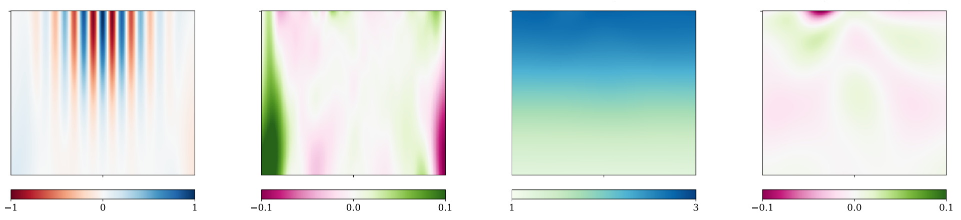
When experimenting with both PINNs and PIKANs, it was found that for the proposed configurations, not only were the PINNs trained in a considerably shorter time, but the relative error achieved for the approximation of k was one order of magnitude lower than with the PIKAN, as shown in figure 13.



(a) Approx. for u (b) u relative error (c) Approx. for k (d) k relative error

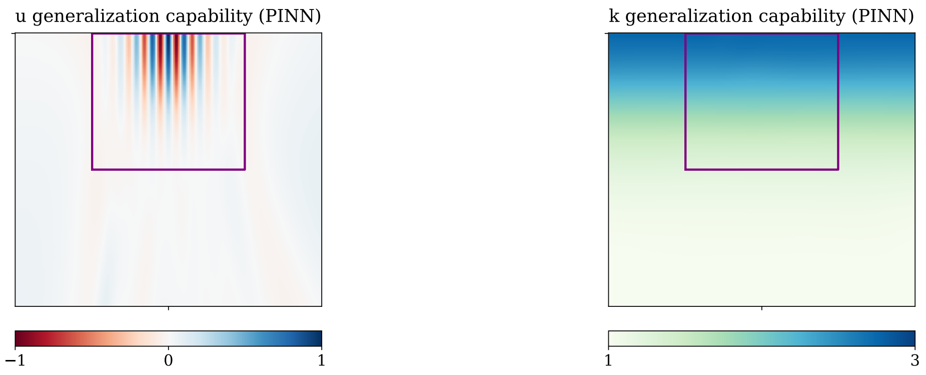
Fig. 10: PINN Predictions for u and k using a $(16, 32)$ network configuration in a semi infinite domain

Figure 14 shows the same experiment as in the infinite domain: the number of training epochs, the network configuration, and the number of training points were kept fixed, while only the number of available observations for u and k varied. For the boundary conditions, an additional set of points equal to 10% of the total interior observations was included. For example, when using 100 interior points, 10 extra boundary points were added. As expected, the error decreases as the number of training points increases, up to a point where it begins to level off.



(a) Approx. for u (b) u relative error (c) Approx. for k (d) k relative error

Fig. 11: PIKAN Predictions for u and k using a $(5, 10)$ network configuration with grid=3 and $k = 3$ in a semi infinite domain



(a) u capacity of generalizing (b) k capacity of generalizing

Fig. 12: Network generalization capacity on a semi-infinite domain domain

4.3 Response to noise on an infinite domain

Experiments with 5%, 10% and 15% white noise added in the observed information were carried, as shown on table 2. The best performing PINN $[(16, 32)]$ configuration, and the best performing PIKAN $[(5, 10)]$ configuration, were retrained for the same number of epochs as described in the methodology, however, 5%, 10% and 15% of white noise was added to the interior data observations (both u and k) used to solve the inverse problems. Figure 15 shows the relative error for k as a function of the noise percentage on observed data. As expected, the error is lower on PINNs, however, it is very interesting to observe that the relation between error and noise percentage appears to be linear in the studied range. Moreover, the growth rate of the error

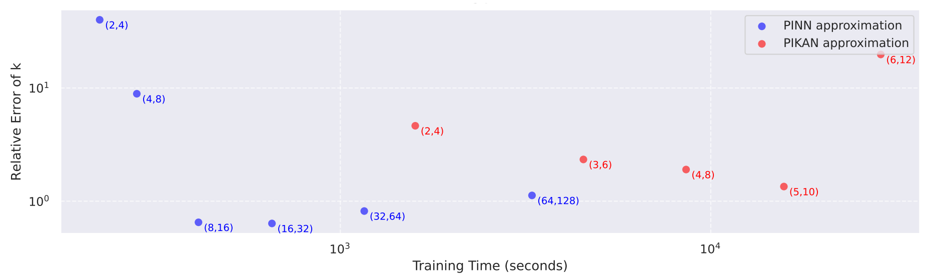


Fig. 13: log log plot of PINNs and PIKANs approximation on a semi-infinite domain. PIKANs tend to take much more to approximate a worse solution

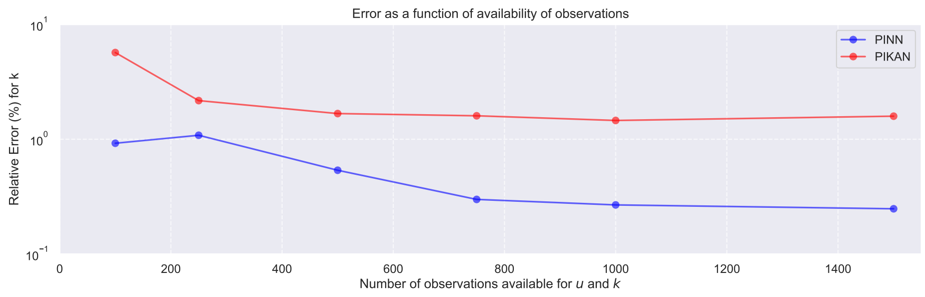


Fig. 14: log plot of error obtained by PINNs and PIKANs approximating k on a semi-infinite domain, while varying the number of available information

for both PINNs and PIKANs is extremely similar, which shows that these networks behave alike under noise in this particular problem.

Noise	Relative error PINN	Relative error PIKAN
No noise for u and k	0.097%	1.086%
5% noise for u and k	0.473%	1.722%
10% noise for u and k	0.931%	1.926%
15% noise for u and k	1.283%	2.447%

Table 2: Relative errors for k, when attempting to solve the inverse problem using PINNs and PIKANs, whith 5%, 10% and 15% noise

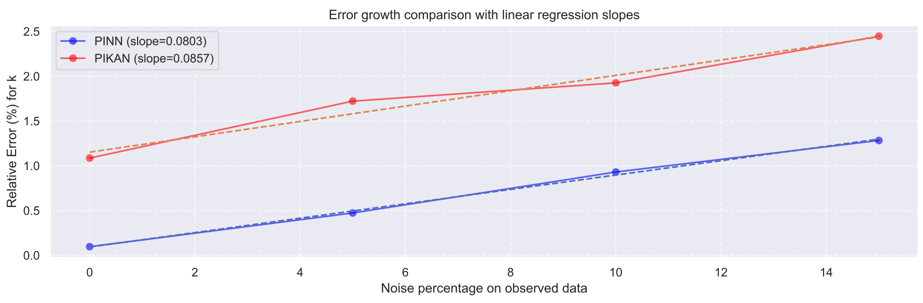


Fig. 15: Error growth for PINNs and PIKANs under different percentages of white noise. Experiments performed on an infinite domain

4.4 A word about the interpretability of Kolmogorov–Arnold Networks

Although Kolmogorov-Arnold Networks ultimately represent solutions as compositions of piecewise polynomials, this structure does not automatically guarantee interpretability in the traditional sense. The intuition behind KANs is that each spline-parameterized univariate function offers a transparent, local view of how inputs are transformed, in contrast to the opaque weight matrices of deep neural networks. In practice, this means that one can visualize or analyze each learned spline to understand the nonlinear behaviors the model captures.

Nonetheless, extracting actionable insight from these spline compositions is not as straight forward as portrayed. As networks grow deeper, the interpretability gained at the level of individual basis functions becomes increasingly entangled through successive compositions. While some structure given by the polynomials is indeed more accessible, for example, identifying dominant regions or transitions or drawing mathematical conclusions from a trained KAN is still challenging. Thus, the appeal of interpretability in KANs should be considered incremental rather than transformative.

Whether this interpretability gain justifies the additional training time depends heavily on the application. In settings where understanding localized nonlinear relations is valuable, the spline-level transparency may beat the computational overhead. However, for purely performance-driven tasks, or when models must scale to high dimensionality, the interpretability advantage may not be sufficient to offset the increased complexity and training cost.

5 Conclusions

In this work, we addressed the inverse Poisson problem in infinite and semi-infinite domains using PINNs and PIKANs. From the experiments carried out, several conclusions can be drawn. Overall, PINNs consistently outperformed PIKANs in terms of both accuracy and computational efficiency. The relative errors obtained with PINNs were lower by approximately one order of magnitude, and the associated training times were significantly reduced. These results indicate that PINNs provide a more reliable and efficient framework for solving inverse problems posed on unbounded domains.

In contrast, PIKANs exhibited limitations that are closely related to their polynomial-based structure, which relies on compositions of polynomials. As the domain approaches infinity, this structure can lead to numerical instability and error growth, negatively impacting performance. Although the interpretability of PIKANs remains a potential advantage, particularly from a theoretical standpoint, this observed instability restricts their applicability when dealing with infinite or semi-infinite domains.

The sampling strategies proposed in this study, based on normal and exponential distributions for the infinite and semi-infinite cases respectively, proved effective in enabling both architectures to generalize beyond the region where training data were concentrated, even in the absence of explicit boundary conditions. It is important to emphasize, however, that this strategy is particularly suitable for problems whose solutions stabilize as we leave the zone of interest. In such cases, the reduced density of training points at large distances and the decay of gradients toward zero contribute to stable learning behavior.

Furthermore, the adoption of a Dual-Network architecture, in which one network is dedicated to learning the state variable u and a second network to learning the coefficient k, was shown to reduce bias between the two approximations. This separation led to improved numerical stability and better performance in the inverse formulation for both PINNs and PIKANs.

Finally, both approaches exhibited a nearly linear increase in error as white noise was added to the observed data. Despite PINNs achieving lower absolute error values across all noise levels considered in this study, the sensitivity to noise was remarkably similar for both PINNs and PIKANs, suggesting comparable robustness trends under noisy conditions.

Appendix A Implementation for reproducibility

For reproducibility purposes and following the Guidelines for Open Science Policies [25], the entirety of code used during the development of this project, as well as the models created by the authors is reported in the following repository: `github.com/gperezb12/PINN-PIKAN-Poisson`.

References

[1] Evans, G., Blackledge, J., Yardley, P.: Numerical Methods for Partial Differential Equations. Springer Undergraduate Mathematics Series. Springer, London, UK (1999). https://doi.org/10.1007/978-1-4471-0377-6

[2] Tarantola, A.: Inverse Problem Theory and Methods for Model Parameter Estimation. Society for Industrial and Applied Mathematics, Philadelphia, PA (2005). https://doi.org/10.1137/1.9780898717921 . https://epubs.siam.org/doi/book/10.1137/1.9780898717921

[3] Burger, H.R., Sheehan, A.F., Jones, C.H.: Introduction to Applied Geophysics: Exploring the Shallow Subsurface. W.W. Norton, New York (2006). Google-Books-ID: pl2qQgAACAAJ

[4] Givoli, D.: A tutorial on the adjoint method for inverse problems. Computer Methods in Applied Mechanics and Engineering **380**, 113810 (2021) https://doi.org/10.1016/j.cma.2021.113810

[5] Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics **378**, 686–707 (2018) https://doi.org/10.1016/j.jcp.2018.10.045

[6] Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T.Y., Tegmark, M.: Kolmogorov-arnold networks (kans). arXiv preprint (2024). Preprint, under review

[7] Bishop, C.M., Bishop, H.: Deep Learning: Foundations and Concepts. Springer, Cham (2023)

[8] Schmidt-Hieber, J.: The kolmogorov–arnold representation theorem revisited. CoRR **abs/2007.15884** (2020) arXiv:2007.15884 [cs.LG]

[9] Yu, R., Yu, W., Wang, X.: Kan or mlp: A fairer comparison. arXiv preprint (2024). Preprint, under review

[10] Shukla, K., Toscano, J.D., Wang, Z., Zou, Z., Karniadakis, G.E.: A comprehensive and fair comparison between MLP and KAN representations for differential equations and operator networks. Computer Methods in Applied Mechanics and Engineering (2024) https://doi.org/10.2139/ssrn.4858126 . Preprint available on SSRN

[11] Shuai, H., Li, F.: Physics-informed kolmogorov–arnold networks for power system dynamics. CoRR **abs/2408.06650** (2024) arXiv:2408.06650

[12] M.Dissanayake, Phan-Thien, N.: Neural-network-based approximations for solving partial differential equations. Communications in Numerical Methods in Engineering **10**(3), 195–201 (1994) https://doi.org/10.1002/cnm.1640100303

[13] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 32 (2019)

[14] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore,

S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 265–283 (2016)

[15] Kochliaridis, V., Dilmperis, I., Palaskos, A., Vlahavas, I.: Generative point sampling strategies for physics-informed neural networks. Engineering with Computers **41**, 3219–3239 (2025) https://doi.org/10.1007/s00366-025-02158-4

[16] Wu, C., Zhu, M., Tan, Q., Kartha, Y., Lu, L.: A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. arXiv preprint arXiv:2207.10289 (2022)

[17] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[18] Mustajab, A.H., Lyu, H., Rizvi, Z., Wuttke, F.: Physics-informed neural networks for high-frequency and multi-scale problems using transfer learning. Applied Sciences **14**(8), 3204 (2024) https://doi.org/10.3390/app14083204

[19] Taylor, J., Wang, W., Bala, B., Bednarz, T.: Optimizing the optimizer for data driven deep neural networks and physics informed neural networks. arXiv preprint arXiv:2205.07430 (2022)

[20] Shunn, L., Ham, F.: Method of manufactured solutions applied to variable-density flow solvers. In: Center for Turbulence Research Annual Research Briefs, pp. 155–168. Stanford University, Stanford, CA (2007)

[21] Zhang, Y., Zhu, X., Gao, J.: Seismic inversion based on acoustic wave equations using physics-informed neural network. IEEE Transactions on Geoscience and Remote Sensing **61**(99), 1–1 (2023) https://doi.org/10.1109/TGRS.2023.3236973

[22] Hare, J.: An Introduction to Automatic Differentiation. Lecture notes, Vision, Learning and Control Group, University of Southampton. Available at: https://rufflewind.com/2016-12-30/reverse-mode-automatic-differentiation (2023)

[23] Rathore, P., Lei, W., Frangella, Z., Lu, L., Udell, M.: Challenges in training pinns: A loss landscape perspective. arXiv preprint arXiv:2402.01868 (2024)

[24] Rincon-Cardeno, O., Perez-Bernal, G., Montoya Noguera, S., Guarin-Zapata, N.: Comparative Analysis of Physics-Informed Neural Networks and the Boundary Element Method for Solving the Helmholtz Equation (Under review) (2025)

[25] UNESCO: UNESCO Open Science Toolkit. UNESCO, Paris (2023). https://unesdoc.unesco.org/ark:/48223/pf0000387983