ScIDL - preliminary results

1st Tomislav Matulić *Dept. of Electronic Systems and Information Processing*
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia
tomislav.matulic@fer.hr

2nd Damir Seršić *Dept. of Electronic Systems and Information Processing*
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia
damir.sersic@fer.hr

Abstract

In this paper, we explore novel approaches to the design of neural network architectures with a minimal number of learnable parameters. Motivated by polynomial regression models, where differential equation solutions can be successfully estimated using only about 10 parameters, we present neural network architectures based on the Horner scheme and Chebyshev polynomials, both consisting of a small parameter set. In the Horner-based network model, we embed the initial conditions directly into the architecture, further reducing the number of learnable parameters while preserving accuracy. These models demonstrate a performance advantage of nearly three orders of magnitude over standard MLP neural networks and networks with periodic activation functions, despite requiring significantly fewer parameters. We extend this idea through a spline-like approach, which reduces the approximation error while introducing only a minimal increase in the number of parameters. This extension ensures continuity and smoothness across subinterval boundaries, making it particularly effective in solving complex problems. Additionally, we apply the Horner-based approach to solve partial differential equations (PDEs), specifically the heat equation. Even with a small number of learnable parameters, our method produces highly accurate solutions, demonstrating its potential for efficient modeling of complex dynamic systems in computational science and engineering. By achieving high accuracy with low computational complexity, our work paves the way for scalable and resource-efficient neural network applications in various domains.

Index Terms

Neural networks, differential equations, Horner scheme

I. Introduction

Neural networks have found widespread applications in many domains, including biology [1], [2], medicine [3]–[5], hyperspectral data analysis [6], and even creative fields such as music, art, and digital media [7]. Their ability to recognize patterns and make predictions from vast and complex datasets has fundamentally transformed how we address many scientific, technological, and creative challenges. Large Language Models [8] (LLMs), such as GPT [9], [10], or DeepSeek [11], have seen unprecedented growth and innovation in recent years. These models, capable of comprehending and generating human-like language with extraordinary accuracy, are opening new frontiers in communication, education, research, and creative writing.

Physics-Informed Neural Networks [12]–[14] (PINNs) represent a powerful class of deep learning models designed to solve problems governed by physical laws, often described by partial differential equations (PDEs) or ordinary differential equations (ODEs). Unlike conventional neural networks, which primarily rely on data-driven learning, PINNs embed known physical principles directly into the training process. This integration ensures that the networks predictions adhere to the underlying laws of physics while simultaneously learning from observed data. Traditional neural networks train by minimizing the difference between predictions and ground truth data, but PINNs take this a step further by incorporating physics-based constraints directly into the loss function, creating more reliable and physically accurate models.

Implicit Neural Representation (INR) is approach that encodes data, such as images, audio signals, or 3D objects, using a neural network. [15]–[18] Here, the network itself acts as a continuous function, mapping input coordinates

to the corresponding data values. Unlike traditional explicit methods that store data as discrete arrays (e.g., pixel grids or voxel grids), INRs provide a compact, flexible, and resolution-independent way to represent and model complex signals. This approach has a wide range of applications, including 3D scene representation, high-resolution image reconstruction, and audio/video encoding. A particularly exciting application of INRs is in solving differential equations by representing their solutions as continuous functions learned by neural networks. One common way to implement this is using a Multilayer Perceptron (MLP), where the network learns the solution through coordinate mappings. Activation functions like ReLU, Tanh, and Softplus are often employed in these architectures to capture the underlying structures of the solutions. In [19], researchers have developed super-resolution frameworks to generate grid-free solutions to PDEs, demonstrating the power of neural networks in avoiding the need for traditional discretized grids. It has been shown that using periodic activation functions (such as sine and cosine) offers significant advantages over traditional activation functions when representing high-frequency signals in applications like audio, video, and 3D object modeling. These periodic functions also excel in solving complex differential equations by accurately capturing oscillatory and periodic behaviors that other activations struggle to represent. [20] Tensor Neural Networks, presented in [21], have also been developed to tackle PDEs, offering powerful architectures capable of solving complex physical models. However, a common characteristic of most neural network-based approaches in this domain is the large number of learnable parameters required for accurate modeling.

In this work, we address the challenge of network architecture design by proposing innovative methods to reduce the number of learnable parameters while maintaining high accuracy and precision in solving differential equations. Instead of relying solely on the loss function, we incorporate rules and constraints directly into the structure of the neural network. Section II presents the notation used throughout this paper. Section III demonstrates solving differential equations using standard MLP neural networks, which serve as a baseline for comparison. Section IV introduces the motivation for our neural network architecture and paradigm, based on polynomial linear regression models. Section V presents two novel neural network architectures: one inspired by the Horner scheme and another based on Chebyshev polynomials. Section VI extends these models using a spline-like approach to further enhance their flexibility and accuracy. Finally, Section VII concludes the paper with a summary of our findings and potential future directions.

II. Notations/Setup/Tak ne

The goal of this paper is to develop a novel paradigm in neural networks for solving differential equations. Before proceeding, we describe the notation and assumptions.

We assume that the ordinary differential equation (ODE) of order n can be written as

$$F(t, x(t), x'(t), \dots, x^{(n)}(t)) = G(u(t), u'(t), \dots, u^{(l)}(t))$$

where $x(t)$ is the unknown function and $u(t)$ is a given input function. Since the input function is known, the right-hand side is determined. Therefore, without any loss of generality, we assume that the problem is given by

$$F(t, x(t), x'(t), \dots, x^{(n)}(t)) = u(t). \tag{1}$$

To fully determine $x(t)$, a set of initial conditions is necessary. In the theory of signals and systems, we often assume n initial conditions at $t = 0$

$$x^{(i)}(0) = x_i, \quad i = 0, 1, \dots, n-1.$$

Later in this paper, these initial conditions will be utilized to simplify expressions.

Machine learning methods are inherently data-driven. In this paper, the data consists of a point cloud $(t_k, u(t_k)),\ k = 1, 2, \dots, M$, obtained from the known input signal.

Throughout this paper, we will solve three different differential equations. The first one (referred to as Type **a**) is

$$x'(t) + 2x(t) = 1,$$
$$x(0) = 1,$$

and its solution is $x(t) = \frac{1}{2}\left(1 + e^{-2t}\right)$.

The second ODE (referred to as Type **b**) is

$$x'(t)x(t) = t,$$
$$x(0) = 1,$$

and its solution is $x(t) = \sqrt{t^2 + 1}$.

The last one (referred to as Type **c**) is

$$x''(t) + 4x'(t) + 13x(t) = 2,$$
$$x(0) = 0,$$
$$x'(0) = 1,$$

and its solution is

$$x(t) = \frac{2}{13} + e^{-2t}\left(\frac{3}{13}\sin(3t) - \frac{2}{13}\cos(3t)\right).$$

Note that Types **a** and **c** are linear ODEs, while Type **b** is nonlinear.

III. Solving ODE using MLP nets

We present the results of MLP networks with various activation functions (ReLU, sigmoid, and periodic activation functions (SIREN)). We solve the differential equation of Type **a** using the following loss function

$$\mathcal{L}_{\text{loss}} = \frac{1}{M}\sum_{i=1}^{M}\left(F(t_i, \mathcal{N}(t_i), \frac{d}{dt}\mathcal{N}(t_i), ..., \frac{d^n}{dt^n}\mathcal{N}(t_i)) - u(t_i)\right)^2 + \sum_{j=0}^{n-1}\lambda_j\left|\frac{d^j}{dt^j}\mathcal{N}(0) - x_j\right|$$

where $\mathcal{N}$ denotes the neural network model of the solution and $(t_k, u(t_k))$ represents the data point cloud. The first term represents the MSE loss of the differential equation, while the second term enforces the initial conditions. For the first-order differential equation (Type **a**), the second term simplifies to a single term for $j = 0$, as we have only one initial condition.

For our first example, we employ an MLP network with 4 hidden layers, each containing 256 features, resulting in a total of 263,937 learnable parameters. The activation function used is Leaky ReLU.

In the second example, we again use an MLP network with 4 hidden layers, but each layer has only 5 features, giving a total of 106 learnable parameters. The activation function used in this case is the sigmoid function (logistic function).

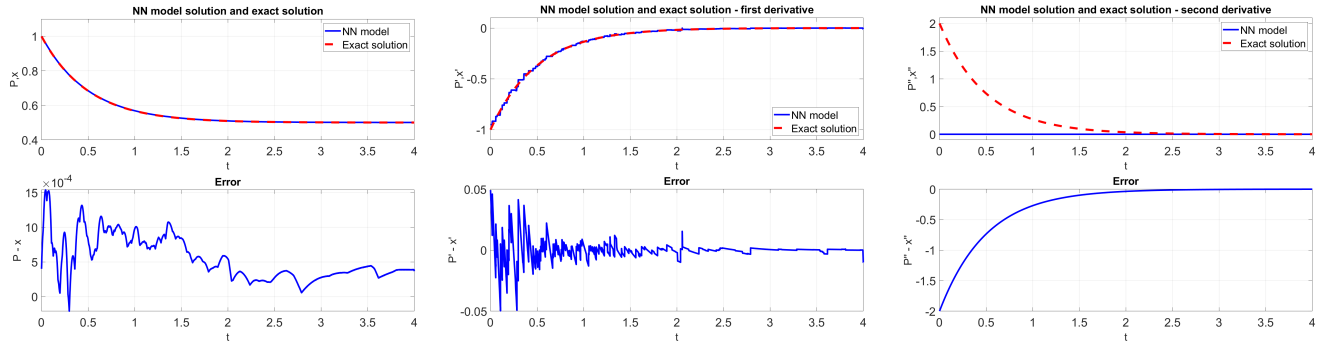
For the final example, we utilize a SIREN network with 4 hidden layers, each containing 5 features, resulting in a total of 106 learnable parameters. The implementation is based on the official repository of "Implicit Neural Representations with Periodic Activation Functions," available on GitHub[1].

To train the neural networks, we employ the Adam optimizer. Starting learning rate is set to 10^{-3}. The hyper-parameter λ_0 is set to 0.1. The networks are trained for 10,000 epochs. The data point cloud consists of a total of $M = 400$ points. For neural networks with periodic activation functions, parameters are set according to the specifications in [20].

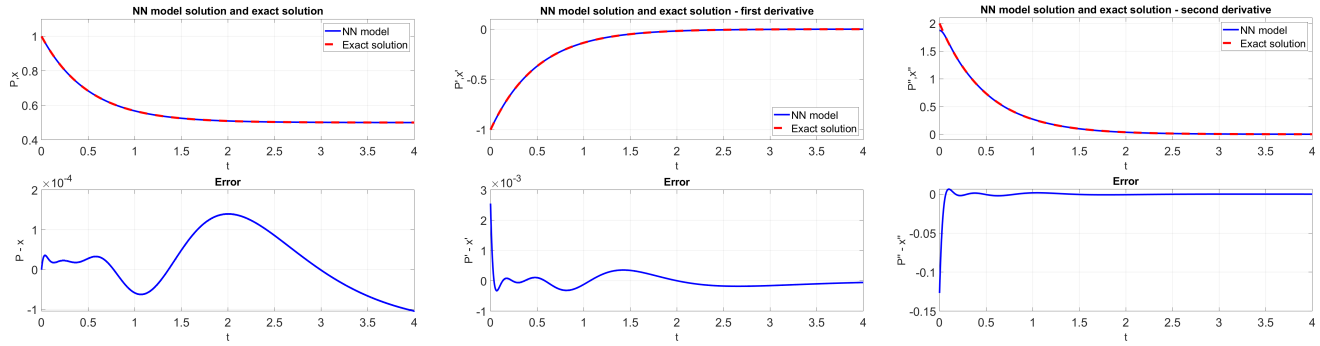
The results from these examples are shown in Fig. 1. We plot the original function and its first two derivatives. We observe that the MLP network with Leaky ReLU activation functions fails to accurately represent the solution. The primary reason for this is that Leaky ReLU is not a differentiable function. Additionally, the second derivative of the Leaky ReLU activation is zero almost everywhere, leading to the second derivative of the implicit representation being a zero function when using this activation.

The results from examples 2 and 3, which utilize networks with 106 learnable parameters, serve as a baseline for comparison with our proposed models. While the parameter count in these networks is relatively low compared to traditional architectures, it is still significantly higher than the dozen or so parameters required by our approach. As will be demonstrated in Section V, our proposed paradigm yields lower error rates despite using fewer parameters, highlighting its efficiency and effectiveness in solving differential equations.

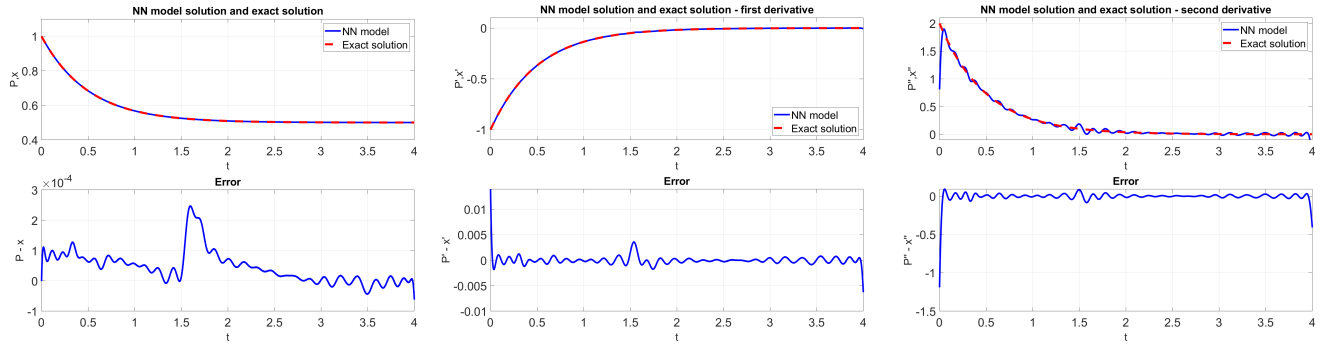
[1]https://github.com/vsitzmann/siren



(a) Type **a** - MLP with Leaky ReLU - solution (b) Type **a** - MLP with Leaky ReLU - first derivative (c) Type **a** - MLP with Leaky ReLU - second derivative



(d) Type **a** - MLP with Sigmoid - solution (e) Type **a** - MLP with Sigmoid - first derivative (f) Type **a** - MLP with Sigmoid - second derivative



(g) Type **a** - SIREN - solution (h) Type **a** - SIREN - first derivative (i) Type 1 - SIREN - second derivative

Fig. 1: Comparison of solutions and their derivatives predicted by the MLP neural networks with the exact solutions.

IV. Motivation - Linear regression

As a motivation for the neural network architecture that we will discuss in Section V, we explore how to construct the solution of a linear ordinary differential equation (ODE) with constant coefficients using a polynomial regression model.

We begin by considering the following differential equation

$$\sum_{i=0}^{n} a_i x^{(i)}(t) = u(t), \tag{2}$$

where $a_i \in \mathbb{R}$. The polynomial $P(t)$, representing the solution model, is given by

$$P(t) = \sum_{j=0}^{m} c_j \frac{t^j}{j!}. \tag{3}$$

Pay attention that it is an approximation of the analytic solution. Note that its derivatives are

$$P^{(l)}(t) = \sum_{j=l}^{m} c_j \frac{t^{j-l}}{(j-l)!}, \quad l = 0, 1, 2, \ldots \tag{4}$$

where c_j are the unknown model parameters to be determined.

Assume that $m \geq n$, meaning that the polynomial order is at least as high as the order of the differential equation. Substituting the polynomial model $P(t)$ into the differential equation yields

$$\begin{aligned} u(t) = \sum_{i=0}^{n} a_i \sum_{j=i}^{m} c_j \frac{t^{j-i}}{(j-i)!} &= \sum_{i=0}^{n} a_i \left(\sum_{j=i}^{n-1} c_j \frac{t^{j-i}}{(j-i)!} + \sum_{j=n}^{m} c_j \frac{t^{j-i}}{(j-i)!} \right) \\ &= \sum_{i=0}^{n} a_i \sum_{j=i}^{n-1} c_j \frac{t^{j-i}}{(j-i)!} + \sum_{i=0}^{n} a_i \sum_{j=n}^{m} c_j \frac{t^{j-i}}{(j-i)!}. \end{aligned} \tag{5}$$

Let's incorporate the initial conditions $x^{(i)}(0) = x_i$. This leads to simplified conditions. From Eq. (4), we observe that:

$$P^{(i)}(0) = c_i = x_i = x^{(i)}(0) \quad \text{for} \quad i = 0, 1, \ldots, n-1.$$

This formulation ensures that the polynomial model satisfies the initial conditions of the differential equation, providing a direct relationship between the coefficients of the polynomial and the initial values of the solution and its derivatives.

Thus, the first sum in Equation (5) is fully determined. We are left with finding the coefficients c_j for $j = n, n+1, \ldots, m$. The expression in Equation (5) can now be rewritten as:

$$u_1(t) = u(t) - \sum_{i=0}^{n} a_i \sum_{j=i}^{n-1} c_j \frac{t^{j-i}}{(j-i)!} = \sum_{i=0}^{n} a_i \sum_{j=n}^{m} c_j \frac{t^{j-i}}{(j-i)!} = \sum_{j=n}^{m} \sum_{i=0}^{n} a_i \frac{t^{j-i}}{(j-i)!} c_j. \tag{6}$$

We recall that the input dataset consists of a point cloud $(t_k, u(t_k))$ for $k = 1, 2, \ldots, M$, where $M > m - n + 1$. By substituting these points into Eq. (6), we obtain an overdetermined system of equations

$$u_1(t_k) = u(t_k) - \sum_{i=0}^{n} \sum_{j=i}^{n-1} a_i c_j \frac{t_k^{j-i}}{(j-i)!} = \sum_{j=n}^{m} \sum_{i=0}^{n} a_i \frac{t_k^{j-i}}{(j-i)!} c_j. \tag{7}$$

This system can be expressed in matrix form as

$$\hat{\boldsymbol{u}} = \boldsymbol{A}\hat{\boldsymbol{c}},$$

where

$$[\hat{\boldsymbol{u}}]_k = u(t_k) - \sum_{i=0}^{n} \sum_{j=i}^{n-1} a_i c_j \frac{t_k^{j-i}}{(j-i)!}, \quad [\boldsymbol{A}]_{k,j-n+1} = \sum_{i=0}^{n} a_i \frac{t_k^{j-i}}{(j-i)!}, \quad [\hat{\boldsymbol{c}}]_{j-n+1} = c_j,$$

for $k = 1, 2, \ldots, M$ and $j = n, n+1, \ldots, m$.

To determine the unknown coefficients, we solve the system using the least squares method. In matrix form, this is expressed as

$$\hat{\boldsymbol{c}} = (\boldsymbol{A}^T \boldsymbol{A})^{-1} \boldsymbol{A}^T \hat{\boldsymbol{u}}.$$

With this, all the unknown coefficients c_j are obtained, and the polynomial

$$P(t) = \sum_{j=0}^{m} c_j \frac{t^j}{j!}$$

becomes the model solution of Eq. 2.

A. *Example 1*

We consider Type **a** differential equation. The differential equation is solved using a polynomial regression model. We employ a polynomial of degree $m = 15$ and $M = 10000$ ordered pairs $(t_k, u(t_k))$, where the samples t_k are taken from a uniform distribution. The interval of interest is the segment $[0, 4]$.

Figure 2a illustrates the solution obtained using the polynomial regression model, the exact solution, and their difference (the error). In Figure 2b, the derivatives and their difference (error) is shown.

We observe that the model, with only 16 parameters, provides an excellent approximation of the exact solution.

B. *Example 2*

Similar to the previous example, we consider the differential equation

$$x'(t) + 2x(t) = e^{-2t},$$
$$x(0) = 0.$$

This time, the input function matches the natural frequency of the homogeneous solution. The same number of parameters is used as in the previous example. The solution is given by

$$x(t) = te^{-2t}.$$

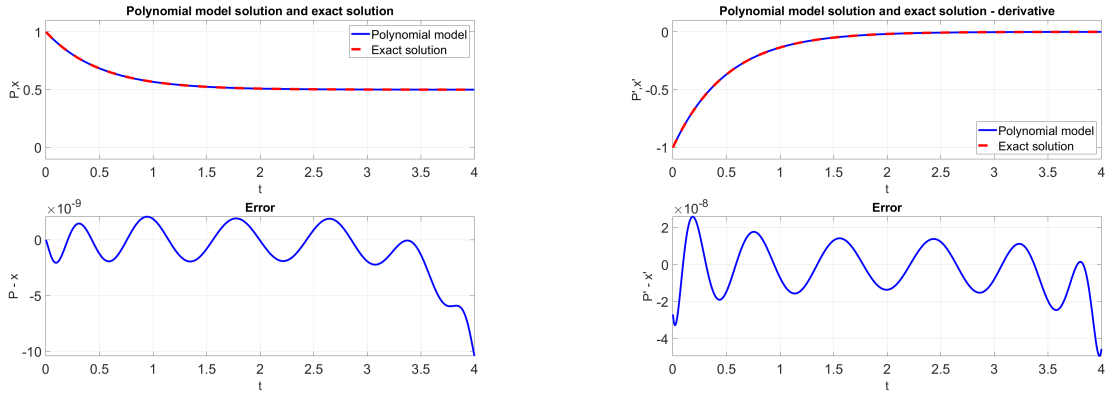
In Figure 3, we observe both the polynomial regression and exact solutions, as well as their derivatives.
As in the previous case, we see that the model, with only 16 parameters, provides a good approximation of the exact solution.

C. *Example 3*

In the final motivational example, we analyze Type **c** differential. This equation represents a system with damping and periodic behavior due to the presence of oscillatory components. We maintain the same model parameters as in the previous examples, where the polynomial regression approach is utilized to approximate the solution.

In Figure 4, we compare the numerical solution obtained through the polynomial regression model with the exact analytical solution. The figure also includes comparisons of the first and second derivatives, which help illustrate how well the model captures both the dynamic behavior and the rate of change within the system.

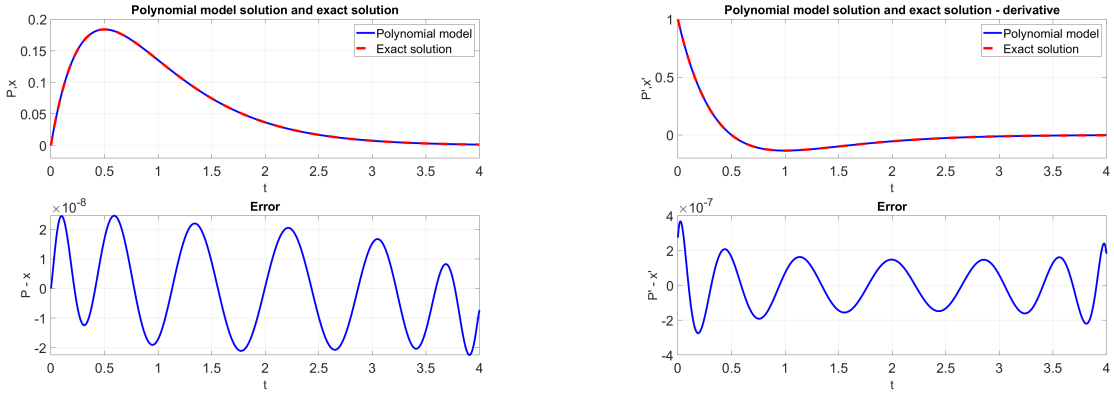
As with the previous examples, we observe an excellent match between the polynomial regression model and the exact solution, even with only 16 parameters in the model. The differences between the polynomial approximation and the exact solution, as well as their derivatives, are minimal, indicating that the regression model effectively captures both the steady-state and transient behaviors of the system.



(a) Polynomial regression and exact solution

(b) Derivatives and their difference

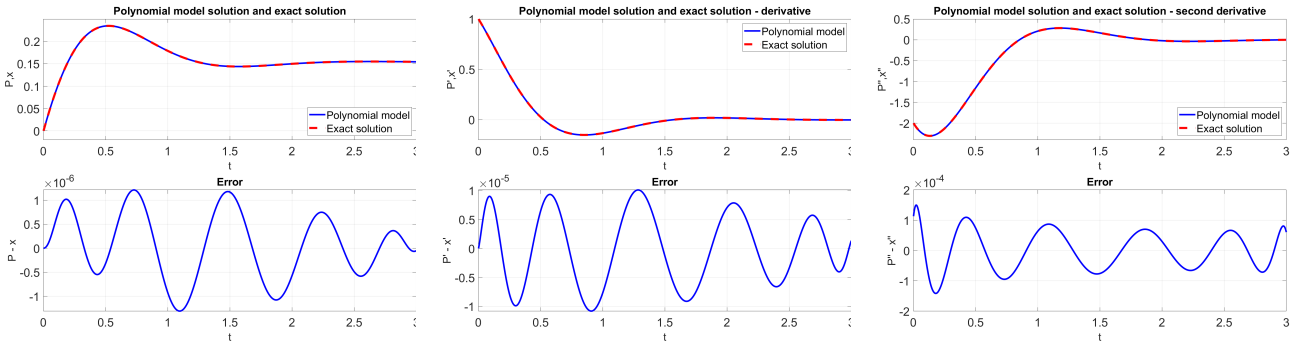
Fig. 2: Polynomial regression model vs. exact solution and their errors



(a) Polynomial regression and exact solution

(b) Derivatives and their difference

Fig. 3: Polynomial regression model vs. exact solution and their errors



(a) Polynomial regression and exact solution

(b) First derivatives and their difference

(c) Second derivatives and their difference

Fig. 4: Comparison of the polynomial regression model with the exact solution and their respective errors.

Motivated by the idea that a small number of parameters can be effectively utilized to solve differential equations through a simple regression model, we extend our research toward the modification and implementation of neural network designs. Our goal is to develop networks that, with a small number of parameters, can efficiently and accurately describe natural phenomena governed by differential equations. By leveraging this approach, we aim to achieve high accuracy while maintaining computational efficiency, providing a robust framework for modeling complex dynamic systems across various fields of science and engineering.

V. Neural Nets

In this section, we present and test neural network architectures that consist of a small number of learnable parameters while effectively solving differential equations. Furthermore, we compare these architectures with the MLP network and the SIREN paradigm, which utilizes periodic activation functions.

A. Horner Scheme

As demonstrated earlier, a simple polynomial regression model can effectively describe solutions to differential equations. Motivated by this concept, we further build upon the idea of representing a polynomial using Horners scheme, which allows for efficient polynomial evaluation with minimal computational overhead

$$P_h(t) = a_0 + t\left(a_1 + t\left(a_2 + \cdots + t\left(a_{m-1} + a_m t\right)\cdots\right)\right). \tag{8}$$

Figure 5a illustrates the Basic Horner Block (BHB), consisting of a single learnable parameter a_i. The entire Horner Network (HN) architecture is depicted in Figure 5b. This architecture serves as the core of our neural

network design. For solving an n-th order differential equation, we assume that the initial conditions are of the same form as those in the motivational examples. The parameters $a_0, a_1, \ldots, a_{n-1}$ are derived from terms involving higher-order polynomial terms. By embedding these initial conditions directly into the model, we reduce the number of learnable parameters, thereby improving learning efficiency and convergence speed.

The input data is given as a set of points $(t_k, u(t_k))$ with $M = 200$. We test the network on three types of differential equations. To train the neural networks, we employ the Adam optimizer. The starting learning rate is set to 10^{-3}, and the networks are trained for 10,000 epochs.

The loss function we minimize is defined as

$$\mathcal{L}_{\text{loss}} = \frac{1}{M}\sum_{i=1}^{M}\left(F(t_i, \mathcal{N}(t_i), \frac{d}{dt}\mathcal{N}(t_i), ..., \frac{d^n}{dt^n}\mathcal{N}(t_i)) - u(t_i)\right)^2,$$

where $\mathcal{N}$ denotes the neural network model.

The initial conditions are embedded in the model through the parameters a_0 for first-order differential equations (Types **a** and **b**) and a_0 and a_1 for the second-order equation (Type **c**).

$$a_0 = \mathcal{N}(0) = x(0) = x_0$$
$$a_1 = \frac{d}{dt}\mathcal{N}(0) = x'(0) = x_1$$

This embedding reduces the number of learnable parameters while ensuring that the initial conditions are met exactly.

For each differential equation, we present the solution given by the neural network, along with its first and second derivatives, which are also computed using the network representation. We use a neural network with 10 learnable parameters for differential equations of Types **1** and **2**, and 13 learnable parameters for Type **3**. Figure 6 shows the network-predicted solutions compared to the exact solutions for each differential equation, along with comparisons of their first and second derivatives.

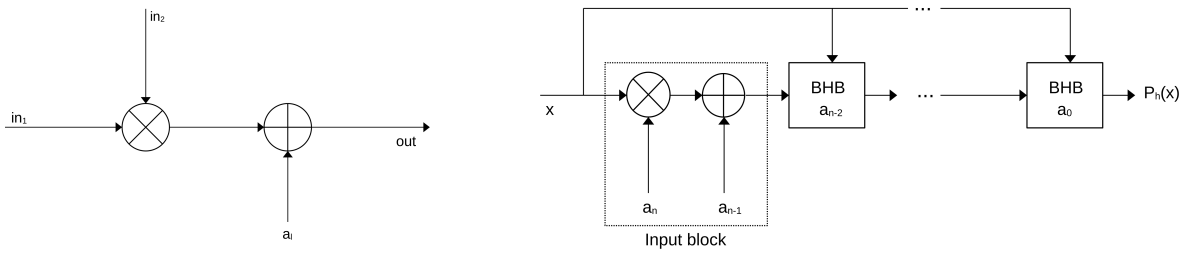
We observe that the solutions obtained using only 10 or 13 learnable parameters closely match the exact solutions, demonstrating the effectiveness of the proposed architecture.

When compared to the baseline models presented in Section III, we observe that our proposed paradigm utilizes significantly fewer learning parameters (approximately one order of magnitude) while achieving lower error rates by nearly three orders of magnitude. This remarkable improvement in both efficiency and accuracy demonstrates the effectiveness of our approach in solving differential equations with minimal computational resources.

B. Cheby

In this example, we use Chebyshev polynomials of the first kind as the basis for the neural network model. The Chebyshev polynomials of the first kind are defined as $T_n(\cos(t)) = \cos(nt)$, or recursively as

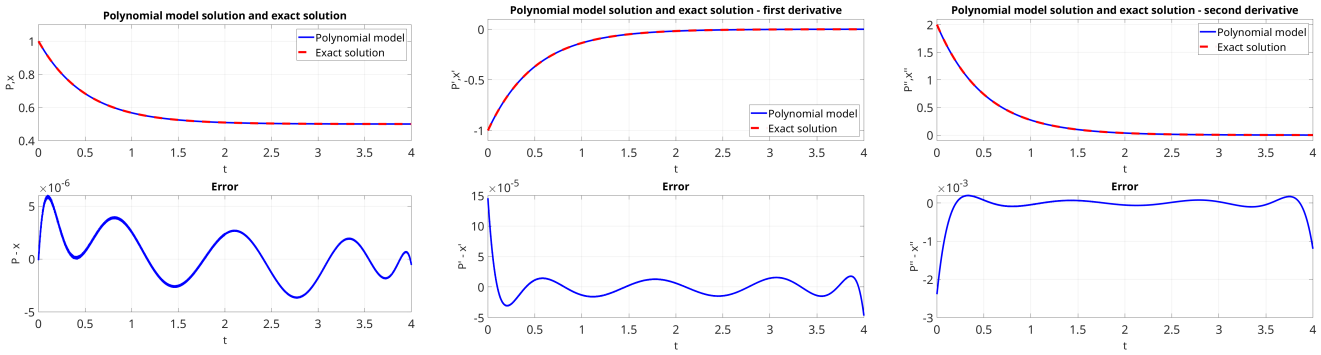
$$\begin{aligned} T_0(t) &= 1, \\ T_1(t) &= t, \\ T_{n+1}(t) &= 2tT_n(t) - T_{n-1}(t). \end{aligned} \tag{9}$$



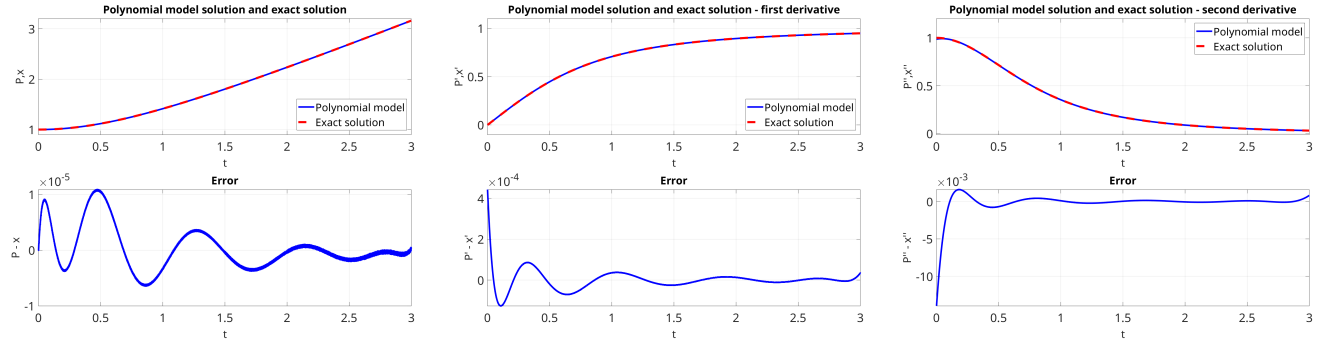
(a) Basic Horner Block (BHB)

(b) Horner Network

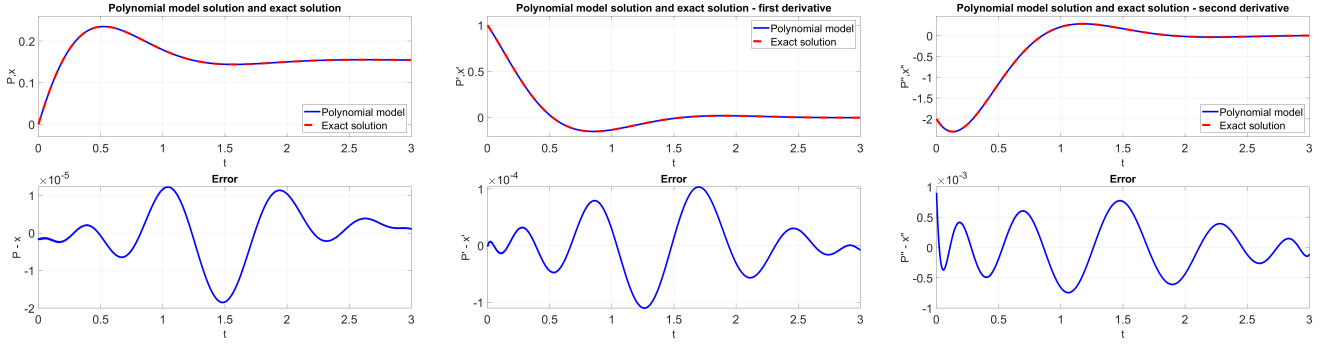
Fig. 5: Basic Horner Block and Horner Network



(a) Type 1 - solution (b) Type 1 - first derivative (c) Type 1 - second derivative



(d) Type 2 - solution (e) Type 2 - first derivative (f) Type 2 - second derivative



(g) Type 3 - solution (h) Type 3 - first derivative (i) Type 3 - second derivative

Fig. 6: Comparison of solutions and their derivatives predicted by the Horner neural network with the exact solutions for the three types of differential equations.

The solution model is represented by a neural network expressed as a linear combination of Chebyshev polynomials

$$P_c(t) = \sum_{i=0}^{m} b_i T_i(t), \tag{10}$$

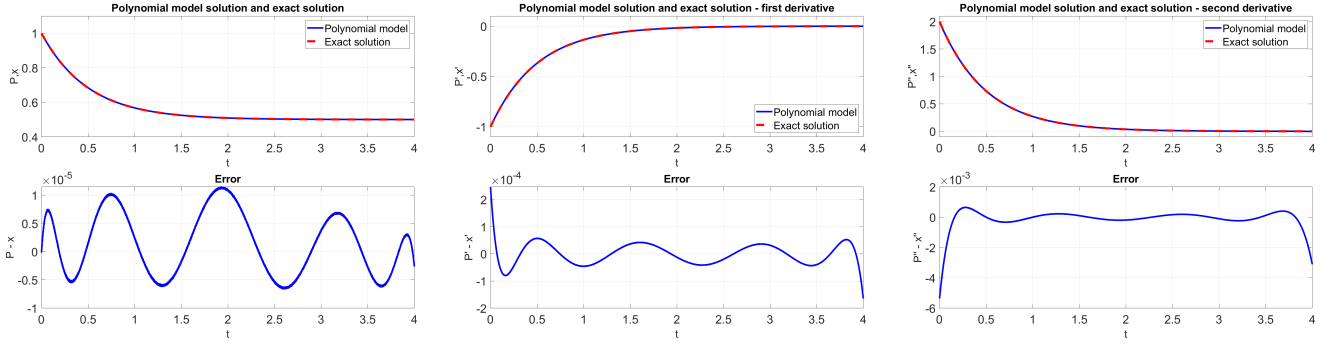
where b_i are the learnable parameters of the model.

We apply this network to solve the differential equation of Type **a**, using the same hyperparameters and learning scheme as in the Horner network example. However, in this case, we incorporate the initial conditions into the loss function, defined as

$$\mathcal{L}_{\text{loss}} = \frac{1}{M} \sum_{i=1}^{M} \left(F(t_i, \mathcal{N}(t_i), \frac{d}{dt}\mathcal{N}(t_i), ..., \frac{d^n}{dt^n}\mathcal{N}(t_i)) - u(t_i) \right)^2 + \sum_{j=0}^{n-1} \lambda_j \left| \frac{d^j}{dt^j}\mathcal{N}(0) - x_j \right|. \tag{11}$$

For our first-order differential equation, only one term from the second summation in the loss function is necessary. This term corresponds to the hyperparameter λ_0, set to 0.1.

Figure 7 presents the results. We observe that the Chebyshev polynomial-based network yields results very similar to those of the Horner network. The magnitude of errors for the original solution, first derivative, and second derivative is of the same order in both models. Moreover, when applying the Chebyshev-based network to other types of differential equations, the results remain consistent with those of the Horner network, demonstrating the robustness of both approaches.



(a) Type 1 - solution (b) Type 1 - first derivative (c) Type 1 - second derivative

Fig. 7: Comparison of the solution and its derivatives predicted by the Chebyshev polynomial-based network with the exact solutions.

C. PDE - Horner Network

In addition to ordinary differential equations that describe phenomena dependent on a single variable (e.g., time, position, temperature), many natural processes are influenced by multiple variables simultaneously. Such phenomena are modeled by partial differential equations (PDEs).

We first explain how to generalize the 1D Horner network model presented in Section V.A to handle two variables. A general polynomial of order n in two variables can be expressed as

$$P_2(x,y) = P_{1,n}(x) + y \cdot P_{1,n-1}(x) + y^2 \cdot P_{1,n-2}(x) + \cdots + y^n \cdot P_{1,0}(x) = \sum_{i=0}^{n} y^i P_{1,n-i}(x),$$

where $P_{1,k}(x)$ are polynomials of order k in a single variable. This expression can be rewritten using Horners scheme as

$$P_2(x,y) = P_{1,n}(x) + y\left(P_{1,n-1}(x) + y\big(P_{1,n-2}(x) + \cdots + y\big(P_{1,1}(x) + y \cdot P_{1,0}(x)\big)\ldots\big)\right).$$

Each polynomial $P_{1,k}(x)$ can be implemented using the Horner network architecture introduced in Section V.A, denoted as $\mathcal{H}(P_{1,k})$. Moreover, the same architectural design generalizes to higher dimensions. The learnable parameters a_0, a_1, …, a_m from Fig. 5 and Eq. 8 are now entire neural networks $\mathcal{H}(P_{1,k})$.

We implemented this extended model to solve the initial-boundary value problem for the heat equation

$$\begin{aligned} \frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial x^2} &= 0, \\ u(x,0) &= f(x), \\ u(0,t) = u(L,t) &= 0, \end{aligned} \tag{12}$$

where we set $k = 0.1$, $L = 1$, and $f(x) = \sin(\pi x)$. The exact solution to this PDE is

$$u(x,t) = \sin(\pi x)e^{-0.1\pi^2 t}.$$

The input data consists of a point cloud $(x_i, t_i, g(x_i,t_i))$ for $i = 1,2,\ldots,M_1$, where $g(x,t)$ represents the excitation. The initial and boundary conditions are also provided as point clouds

$$(x_j, 0, f(x_j)), \quad j = 1,2,\ldots,M_2,$$

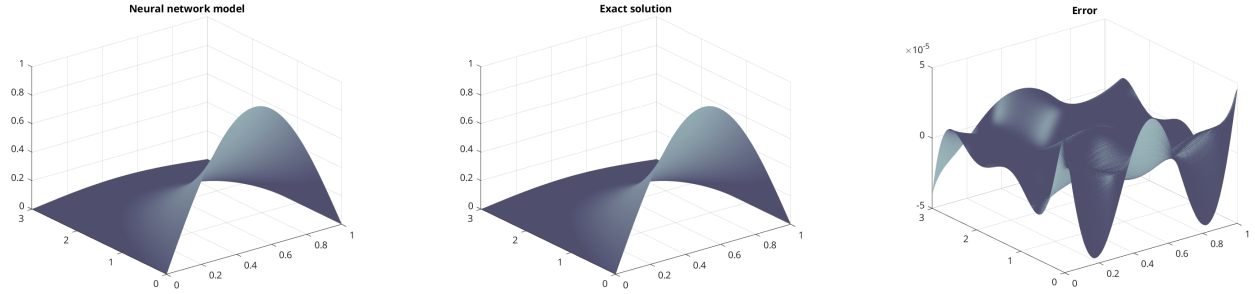
$$(0, t_k, h_1(t_k)), \quad k = 1, 2, \ldots, M_3,$$
$$(L, t_p, h_2(t_p)), \quad p = 1, 2, \ldots, M_4.$$

In our example, we used $M_1 = 5000$ and $M_2 = M_3 = M_4 = 2500$. The right-hand side of the PDE is set to zero ($g(x_i, t_i) = 0$) because the equation is homogeneous. The initial condition is $f(x_j) = \sin(\pi x_j)$, and the boundary conditions are $h_1(t_k) = 0$ and $h_2(t_p) = 0$. The initial and boundary conditions are enforced directly through the loss function

$$\mathcal{L}_{\text{loss}} = \frac{1}{M_1} \sum_{i=1}^{M_1} \left(F(t_i, \mathcal{N}(t_i), \frac{d}{dt}\mathcal{N}(t_i), ..., \frac{d^n}{dt^n}\mathcal{N}(t_i)) \right)^2 + \frac{\lambda}{M_2} \sum_{j=1}^{M_2} \left(\mathcal{N}(x_j, 0) - f(x_j)\right)^2 +$$
$$\frac{\mu}{M_3} \sum_{k=1}^{M_3} \left(\mathcal{N}(0, t_k) - h_1(t_k)\right)^2 + \frac{\nu}{M_4} \sum_{p=1}^{M_4} \left(\mathcal{N}(L, t_p) - h_2(t_p)\right)^2 .$$

We set the hyperparameters as $\lambda = 0.5$ and $\mu = \nu = 0.25$. The total number of learnable parameters in the model is 45. Training is performed as described previously.

Fig. 8 compares the solution obtained from the neural network with the exact solution. The results demonstrate that the error is minimal despite using only 45 parameters, highlighting the efficiency and accuracy of the multidimensional Horner network.



(a) Horner model (b) Exact solution (c) Error

Fig. 8: Comparison of the solution obtained from the multidimensional Horner network with the exact solution of the heat equation. The error is minimal, even with only 45 learnable parameters.

VI. Spline-like Model

In this section, we extend the Horner network by introducing a spline-like approach to further reduce the approximation error when solving differential equations. The key advantage of this approach lies in its ability to balance accuracy with computational efficiency by keeping the number of learnable parameters to a minimum (slightly higher than the baseline model), even as the model complexity increases.

This technique is particularly beneficial for problems involving complex differential equations, where traditional single-network architectures may struggle to provide accurate solutions without a significant increase in the number of parameters. By leveraging the spline-like approach, we achieve an efficient trade-off between model size and accuracy, making it suitable for applications where computational resources are limited or precision is critical.

Let the interval of interest, where we solve the differential equation, be denoted as $[c, d]$. We partition this interval into subintervals such that $c = c_0 < c_1 < c_2 < \cdots < c_l = d$. On each subinterval $C_i = [c_i, c_{i+1}]$, for $i = 0, 1, \ldots, l-1$, we train a separate Horner network with a small number of parameters (corresponding to a lower-order polynomial). By employing a spline-like approach, we ensure through the loss function that the overall solution model belongs to the class $C^j([c, d])$ for a suitable value of j, ensuring continuity and smoothness.

This procedure is illustrated schematically in Fig. 9. The input value x is fed into a demultiplexer and logic module, which selects the corresponding network HN_i based on the subinterval C_i containing x. The output from the selected network is then passed through a multiplexer to produce the output of the entire spline-like model.

Thus, we have l Horner networks in total, with each network HN_i responsible for learning the solution on its respective subinterval C_i. As stated earlier, the loss function enforces the model to be of class $C^j([c,d])$.

We test this model on a Type **a** differential equation. In this case, the loss function is defined as

$$\mathcal{L}_{\text{loss}} = \frac{1}{M}\sum_{i=1}^{M}\left(F(t_i,\mathcal{N}(t_i),\frac{d}{dt}\mathcal{N}(t_i),...,\frac{d^n}{dt^n}\mathcal{N}(t_i)) - u(t_i)\right)^2 + \lambda_0\left|\mathcal{N}(0)-x(0)\right| + \sum_{j=0}^{l-2}\mu_j\left|\mathcal{N}_j(t_{j,r})-\mathcal{N}_{j+1}(t_{j+1,l})\right| + \sum_{j=0}^{l-2}\nu_j\left|\frac{d}{dt}\mathcal{N}_j(t_{j,r})-\frac{d}{dt}\mathcal{N}_{j+1}(t_{j+1,l})\right|.$$

Here, $t_{j,l}$ and $t_{j,r}$ denote the left and right endpoints of subinterval C_j. The third and fourth terms in the loss function enforce continuity of the solution and its first derivative at the subinterval boundaries.

For training the Type **a** differential equation solution, we used $M = 200$ data points. The hyperparameters were set to $\lambda_0 = 1$, $\mu_j = 0.5$, and $\nu_j = 0.5$. The subintervals were chosen as $[0,1]$, $[1,2]$, $[2,3]$, and $[3,4]$, resulting in a total of 4 Horner networks. Each network is characterized by 8 learnable parameters. We employ the Adam optimizer and the starting learning rate is set to 10^{-3}. The network is trained for 10,000 epochs.

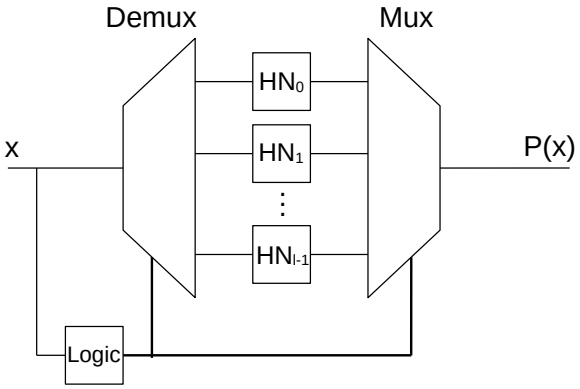
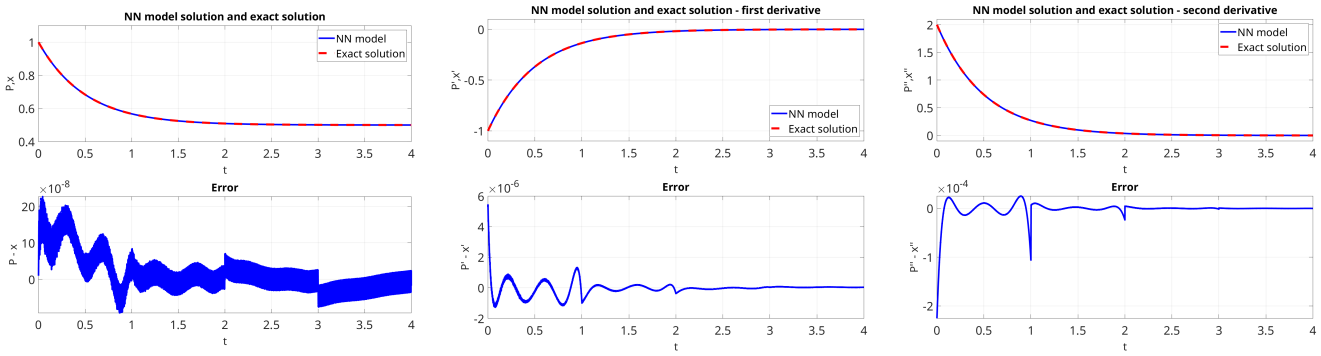


Fig. 9: Spline-like model architecture. The input value x determines which Horner network HN_i is selected based on the interval it falls into, ensuring a piecewise smooth representation of the solution.

The results are presented in Fig. 10. We observe that, in this example, the spline-like model achieves at least an order of magnitude improvement in accuracy over the baseline model presented in Section V.A.



(a) Type 1 - solution

(b) Type 1 - first derivative

(c) Type 1 - second derivative

Fig. 10: Comparison of the solution and its derivatives predicted by the spline-like Horner network with the exact solutions. The results demonstrate improved accuracy and smoothness across the subinterval boundaries.

VII. Conclusion

In this paper, we have demonstrated that natural phenomena can be successfully modeled and solved using neural networks with a minimal number of learnable parameters. By leveraging carefully designed architectures, we have shown that even with a significantly reduced parameter count, accurate and reliable solutions to differential equations can be achieved.

Our models, based on the Horner scheme and Chebyshev polynomials, achieved exceptional results with only around 10 learnable parameters. These models exhibited improvements of nearly three orders of magnitude in accuracy compared to standard MLP neural networks and networks with periodic activation functions, both of which typically require at least twice the number of parameters. This highlights the efficiency of our proposed architectures in scenarios where computational resources and parameter budgets are limited.

The spline-like approach further improved accuracy by effectively reducing the approximation error while only minimally increasing the number of learnable parameters. This method ensured continuity and smoothness across subinterval boundaries, making it particularly effective for applications requiring fine-grained precision and dynamic adaptation to problem complexity.

By solving the heat equation, we generalized the Horner-based approach to two-dimensional domains. Our results demonstrated that even with as few as 50 learnable parameters, we could accurately solve partial differential equations (PDEs). This shows that our method is not limited to one-dimensional problems but can be extended to more complex multidimensional scenarios with minimal computational overhead.

Further reductions in the number of learnable parameters in the spline-like model can be achieved by embedding boundary conditions of subintervals directly into the neural network architecture. This eliminates the need to enforce continuity or derivative constraints through the loss function, thus simplifying the training process and improving efficiency. We also propose potential extensions of the Horner-based approach to higher-order PDEs, offering exciting opportunities for addressing complex dynamic systems in future work.

An additional challenge lies in integrating the spline-like approach into models that solve complex partial differential equations (PDEs). This involves extending the current architecture to handle higher-dimensional domains and ensuring that continuity, smoothness, and physical boundary conditions are maintained directly through the network design. Overcoming this challenge will allow for scalable solutions to PDEs in fields such as fluid dynamics, electromagnetism, and thermodynamics, where traditional methods often face limitations in terms of parameter efficiency and computational complexity. A further challenge involves learning suitable subintervals for 1D cases, or patches in higher dimensions, to minimize the number of parameters or reduce the error, rather than keeping them fixed. Addressing these challenges will be a step in further advancing neural network-based approaches for scientific modeling.

In conclusion, our work provides a robust and scalable framework for solving differential equations using compact neural networks. Future efforts will focus on generalizing the approach to higher-order PDEs and exploring real-world applications in physics, engineering, and computational science, where accuracy, parameter efficiency, and adaptability are critical.

References

[1] Y. Jiang, M. Balaban, Q. Zhu, and S. Mirarab, "Depp: Deep learning enables extending species trees using single genes," *Systematic Biology*, vol. 72, no. 1, p. 1734, Apr. 2022. [Online]. Available: http://dx.doi.org/10.1093/sysbio/syac031

[2] W. P. Walters and R. Barzilay, "Applications of deep learning in molecule generation and molecular property prediction," *Accounts of Chemical Research*, vol. 54, no. 2, p. 263270, Dec. 2020. [Online]. Available: http://dx.doi.org/10.1021/acs.accounts.0c00699

[3] R. Aggarwal, V. Sounderajah, G. Martin, D. S. W. Ting, A. Karthikesalingam, D. King, H. Ashrafian, and A. Darzi, "Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis," *npj Digital Medicine*, vol. 4, no. 1, Apr. 2021. [Online]. Available: http://dx.doi.org/10.1038/s41746-021-00438-z

[4] C. Shen, D. Nguyen, Z. Zhou, S. B. Jiang, B. Dong, and X. Jia, "An introduction to deep learning in medical physics: advantages, potential, and challenges," *Physics in Medicine and Biology*, vol. 65, no. 5, p. 05TR01, Mar. 2020. [Online]. Available: http://dx.doi.org/10.1088/1361-6560/ab6f51

[5] S. S and J. S. Raj, "Analysis of deep learning techniques for early detection of depression on social media network - a comparative study," *March 2021*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235193557

[6] J. M. Bioucas-Dias, A. Plaza, G. Camps-Valls, P. Scheunders, N. Nasrabadi, and J. Chanussot, "Hyperspectral remote sensing data analysis and future challenges," *IEEE Geoscience and Remote Sensing Magazine*, vol. 1, no. 2, p. 636, Jun. 2013. [Online]. Available: http://dx.doi.org/10.1109/MGRS.2013.2244672

[7] J.-P. Briot and F. Pachet, "Deep learning for music generation: challenges and directions," *Neural Computing and Applications*, vol. 32, no. 4, p. 981993, Oct. 2018. [Online]. Available: http://dx.doi.org/10.1007/s00521-018-3813-6

[8] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, p. 145, Mar. 2024. [Online]. Available: http://dx.doi.org/10.1145/3641289

[9] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[10] OpenAI, "Chatgpt (feb 11 version)," 2025, large language model. Accessed: February 11, 2025. [Online]. Available: https://openai.com

[11] DeepSeek-AI, :, X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu, H. Gao, K. Gao, W. Gao, R. Ge, K. Guan, D. Guo, J. Guo, G. Hao, Z. Hao, Y. He, W. Hu, P. Huang, E. Li, G. Li, J. Li, Y. Li, Y. K. Li, W. Liang, F. Lin, A. X. Liu, B. Liu, W. Liu, X. Liu, X. Liu, Y. Liu, H. Lu, S. Lu, F. Luo, S. Ma, X. Nie, T. Pei, Y. Piao, J. Qiu, H. Qu, T. Ren, Z. Ren, C. Ruan, Z. Sha, Z. Shao, J. Song, X. Su, J. Sun, Y. Sun, M. Tang, B. Wang, P. Wang, S. Wang, Y. Wang, Y. Wang, T. Wu, Y. Wu, X. Xie, Z. Xie, Z. Xie, Y. Xiong, H. Xu, R. X. Xu, Y. Xu, D. Yang, Y. You, S. Yu, X. Yu, B. Zhang, H. Zhang, L. Zhang, L. Zhang, M. Zhang, M. Zhang, W. Zhang, Y. Zhang, C. Zhao, Y. Zhao, S. Zhou, S. Zhou, Q. Zhu, and Y. Zou, "Deepseek llm: Scaling open-source language models with longtermism," 2024. [Online]. Available: https://arxiv.org/abs/2401.02954

[12] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physicsinformed neural networks: Where we are and whats next," *Journal of Scientific Computing*, vol. 92, no. 3, Jul. 2022. [Online]. Available: http://dx.doi.org/10.1007/s10915-022-01939-z

[13] J. Donnelly, A. Daneshkhah, and S. Abolfathi, "Physics-informed neural networks as surrogate models of hydrodynamic simulators," *Science of The Total Environment*, vol. 912, p. 168814, Feb. 2024. [Online]. Available: http://dx.doi.org/10.1016/j.scitotenv.2023.168814

[14] X.-X. Chen, P. Zhang, and Z.-Y. Yin, "Physics-informed neural network solver for numerical analysis in geoengineering," *Georisk: Assessment and Management of Risk for Engineered Systems and Geohazards*, vol. 18, no. 1, p. 3351, Jan. 2024. [Online]. Available: http://dx.doi.org/10.1080/17499518.2024.2315301

[15] D. Xu, P. Wang, Y. Jiang, Z. Fan, and Z. Wang, "Signal processing for implicit neural representations," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 13 404–13 418. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/575c450013d0e99e4b0ecf82bd1afaa4-Paper-Conference.pdf

[16] Y. Strümpler, J. Postels, R. Yang, L. V. Gool, and F. Tombari, *Implicit Neural Representations for Image Compression*. Springer Nature Switzerland, 2022, p. 7491. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-19809-0_5

[17] X. Chen, J. Pan, and J. Dong, "Bidirectional multi-scale implicit neural representations for image deraining," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024, pp. 25 627–25 636.

[18] S. Yang, M. Ding, Y. Wu, Z. Li, and J. Zhang, "Implicit neural representation for cooperative low-light image enhancement," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 12 918–12 927.

[19] C. l. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, K. Kashinath, M. Mustafa, H. A. Tchelepi, P. Marcus, M. Prabhat, and A. Anandkumar, "Meshfreeflownet: A physics-constrained deep continuous space-time super-resolution framework," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Nov. 2020, p. 115. [Online]. Available: http://dx.doi.org/10.1109/SC41405.2020.00013

[20] V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," in *Proc. NeurIPS*, 2020.

[21] R. Patel, C.-W. Hsing, S. Sahin, S. S. Jahromi, S. Palmer, S. Sharma, C. Michel, V. Porte, M. Abid, S. Aubert, P. Castellani, C.-G. Lee, S. Mugel, and R. Orus, "Quantum-inspired tensor neural networks for partial differential equations," 2022. [Online]. Available: https://arxiv.org/abs/2208.02235