Physics-Informed Neural Networks for the One-Dimensional Heat Equation

Live L. Storborg and Henrik Haug
Department of Mathematics, University of Oslo,
PO Box 1053, Blindern 0316, Oslo, Norway

AND

Simon S. Thommesen and Adam Falchenberg
Institute of Theoretical Astrophysics, University of Oslo,
PO Box 1029, Blindern 0315, Oslo, Norway

(Dated: December 19, 2025)

We investigate numerical solutions to the one-dimensional heat equation using finite difference (FD) methods and Physics-Informed Neural Networks (PINNs). The explicit forward Euler scheme with centered spatial differences achieves high accuracy (error $\sim 10^{-5}$), with errors decreasing systematically as spatial resolution increases. For PINNs, we implement hard boundary conditions through a trial solution that automatically satisfies initial and boundary conditions, eliminating the need for penalty terms and hyperparameter tuning. A comprehensive architecture sweep across 45 configurations demonstrates that activation function choice is critical: smooth, continuously differentiable functions (SiLU, GeLU, tanh) achieve relative L^2 errors of 10^{-3} to 10^{-4}, while ReLU fails with errors exceeding 0.23. The optimal PINN configuration (3 hidden layers, 32 neurons, SiLU activation) achieves comparable accuracy (error $\sim 5.6 \times 10^{-4}$) to the FD scheme. Error analysis reveals that FD exhibits smooth, monotonically decaying errors consistent with physical diffusion, while PINNs display temporal oscillations despite low overall error. Our findings show that while PINNs require careful tuning for simple problems, they offer distinct advantages for complex geometries, high-dimensional systems, and inverse problems where traditional methods face fundamental limitations.

I. INTRODUCTION

Partial differential equations (PDEs) form the backbone of mathematical models describing a wide range of physical phenomena, such as fluid flow, wave propagation, and heat diffusion. The latter will be explored in this article. Since many PDEs used in physics, such as the Navier–Stokes equations as an example, lack closed-form analytical solutions, developing reliable numerical methods has been essential. Numerical Galerkin methods, such as the finite element method (FEM), have traditionally been the primary tools for solving PDEs. However, these approaches often require substantial computational resources and expensive hardware. Moreover, because they rely heavily on spatial discretization, generating sufficiently dense meshes can involve a significant amount of manual work [1].

In recent years, research has increasingly focused on solving PDEs using Physics-Informed Neural Networks (PINNs). PINNs were introduced by Raissi, Perdikaris, and Karniadakis [2] as neural networks trained not only on data, but also on the residuals of the governing PDE and its associated initial and boundary conditions. The key idea is to embed the physics of the problem directly into the loss function, enabling the network to be penalized based on how well it satisfies the PDE at selected points. To achieve this, we make use of automatic differentiation to compute derivatives of the neural network output with respect to its inputs. Since PINNs don't require a mesh, they have become a good candidate for solving problems with complex geometries or moving boundary conditions.

Despite their potential as an alternative for numerically solving PDEs, PINNs still present several practical challenges. According to Botarelli [1], PINNs often require long training times and can be highly sensitive to both the network architecture, the number and distribution of collocation points, and the choice of activation functions. These limitations highlight that the development of robust and efficient PINN methodologies is still ongoing, making them a compelling topic for continued investigation.

Section II introduces the theoretical background with the implementation details in Section III, followed by the presentation of our results in section IV and a discussion in section V. Finally, we will conclude our findings in Section VI.

II. THEORY

A. Heat Equation

The 1D heat equation, which models the diffusion of heat through a medium in one spatial dimension, is given by

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \alpha \frac{\partial u(x,t)}{\partial t}, \qquad t > 0,\ x \in [0, L], \tag{1}$$

or more compactly,

$$u_{xx} = \alpha u_t, \tag{2}$$

where $\alpha > 0$ denotes the thermal diffusivity of the medium and $u(x,t)$ being the temperature at position x at time t. In this work, we set $\alpha = 1$, which simplifies the equation to

$$u_{xx} = u_t.$$

For later use in the PINN formulation, we define the PDE residual as

$$f(x,t) = u_t(x,t) - u_{xx}(x,t),$$

which should vanish for all (x,t) in the interior of the domain.

1. Boundary and Initial Conditions

We study the heat equation with the initial condition

$$u(x,0) = \sin(\pi x), \qquad 0 < x < L,$$

where $L = 1$ denotes the length of the spatial domain. The boundary conditions are homogeneous Dirichlet and given by

$$u(0,t) = 0, \qquad t \geq 0,$$

$$u(L,t) = 0, \qquad t \geq 0.$$

Here, $u(x,t)$ represents the temperature distribution along a one-dimensional rod. Because heat diffuses over time, the amplitude of the temperature decreases exponentially, while its spatial shape remains a scaled version of the initial sine wave.

2. The analytical solution

The heat diffusion equation can be solved analytically by separation of variables. The analytical solution becomes

$$u(x,t) = e^{-\pi^2 t} \sin(\pi x), \tag{3}$$

after applying the boundary and initial conditions mentioned above, and $\alpha = 1$. The derivation of this is shown in Appendix A.

B. Explicit Finite Difference Scheme

In addition to computing the analytical solution of the heat equation, we also solve it numerically using a finite difference (FD) method based on an explicit time-stepping scheme. The explicit scheme is derived by applying Taylor expansions of $u(x,t)$ in both time and space about the grid points, and truncating the expansions at the desired order of accuracy. Because the heat equation is first order in time and second order in space, we use a Forward Euler scheme for the temporal derivative and a centered difference scheme for the spatial second derivative. The full derivation is provided in Appendix B.

1. Forward Euler

The Forward Euler approximation for the time derivative is obtained by performing a Taylor expansion of $u(x, t+\Delta t)$ about t. Truncating after the first-order term yields

$$u_t(x_i, t_n) \approx \frac{u(x_i, t_{n+1}) - u(x_i, t_n)}{\Delta t}.$$

This yields a first-order accurate explicit update in time.

2. Centered Difference

To approximate the second spatial derivative, we expand $u(x_i \pm \Delta x, t)$ about x_i and truncate after the second-order term. This leads to the centered difference formula

$$u_{xx}(x_i, t_n) \approx \frac{u(x_{i+1}, t_n) - 2u(x_i, t_n) + u(x_{i-1}, t_n)}{\Delta x^2},$$

which is second-order accurate in space.

Combining the Forward Euler time discretization with the centered difference spatial discretization gives the full explicit scheme used in this article.

3. Stability Criterion

Since the scheme is discretized uniformly in both space and time, with spatial step size Δx and time step Δt, it is convenient to introduce the non-dimensional parameter

$$\alpha = \frac{\Delta t}{\Delta x^2}.$$

For the explicit scheme to remain stable, this parameter must satisfy

$$\alpha \leq \frac{1}{2},$$

which ensures that numerical errors do not grow from one time step to the next.

C. Neural Networks

In this work, we also employ neural networks to approximate solutions to partial differential equations. Specifically, we use a PINN approach, which combines the universal approximation capabilities of feed-forward neural networks with physical constraints.

1. Feed-Forward Neural Networks

A feed-forward neural network (FFNN), also known as a Multi-Layer Perceptron (MLP), is a type of neural network where information flows in only one direction, from input through hidden layers to output, with no feedback loops [3]. In a fully connected network, each neuron in layer l is connected to all neurons in layer $l-1$.

For each layer l, the network computes a weighted sum of inputs plus a bias:

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}, \tag{4}$$

where $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ are the weights, $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$ are the biases, and $\boldsymbol{a}^{(l-1)}$ are the activations from the previous layer. The input layer has $\boldsymbol{a}^{(0)} = \boldsymbol{x}$.

This weighted sum is then passed through a nonlinear activation function:

$$\boldsymbol{a}^{(l)} = f^{(l)}(\boldsymbol{z}^{(l)}), \tag{5}$$

which introduces the nonlinearity necessary for approximating complex functions. The network output $\boldsymbol{a}^{(L)}$ from the final layer L represents the network's prediction.

2. Physics-Informed Neural Networks

PINNs [2] use the feed-forward architecture described above but incorporate physical laws directly into the training process. Rather than learning from data alone, PINNs are trained to satisfy the governing differential equations, boundary conditions, and initial conditions of the physical system.

The physics of a system can be embedded in the neural network in several ways. It can be included as extra terms in the loss function - so called soft boundaries, or by directly expressing a trial solution in a way that automatically satisfies the boundary and initial conditions - hard boundaries. For complex boundaries and initial conditions, accounting for the physics in the loss function might be necessary. Soft boundaries has the advantage of being easier to implement and being flexible in terms of the properties of the system: like for instance its geometry. Hard boundaries, however trickier to implement, has the advantage of enforcing exact satisfaction of the boundaries and the simpler loss-function improves training time and stability. For our relatively simple system, we are able to enforce hard boundaries by choosing a trial solution.

3. Trial Solution with Hard Boundary Conditions

We employ a hard boundary condition approach where the trial solution is constructed to automatically satisfy both boundary and initial conditions. This is achieved by composing the neural network output $\mathcal{N}(x,t;\boldsymbol{P})$ in a trial solution, designed for this exact problem. We define our trial solution as:

$$u(x,t) = (1-t)\sin(\pi x) + t\,x(1-x)\,\mathcal{N}(x,t;\boldsymbol{P}), \tag{6}$$

where $\mathcal{N}(x,t;\boldsymbol{P})$ is the output of the neural network and $\boldsymbol{P}$ represents all network parameters.

This trial solution automatically satisfies all constraints discussed in II A 1. The factor $(1-t)$ ensures the initial condition is satisfied exactly at $t=0$, while the factor $x(1-x)$ vanishes at both boundaries $x=0$ and $x=1$ for all time. The neural network $\mathcal{N}(x,t;\boldsymbol{P})$ is therefore free to learn the dynamics of the PDE in the interior domain without being constrained by the boundary conditions.

4. Loss Function

Since we employ hard boundary conditions, our loss function consists only of the PDE residual term. The boundary and initial conditions are automatically satisfied by the trial solution structure (Equation 6) and therefore do not require explicit penalty terms.

The loss function is defined as:

$$\mathcal{L}(\boldsymbol{P}) = \mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{int}}}\sum_{i=1}^{N_{\text{int}}}\left[\frac{\partial u}{\partial t}(x_i,t_i) - \frac{\partial^2 u}{\partial x^2}(x_i,t_i)\right]^2, \tag{7}$$

where (x_i,t_i) for $i=1,\ldots,N_{\text{int}}$ are randomly sampled collocation points in the interior domain $(x,t)\in(0,1)\times(0,T)$.

The PDE residual measures how well the trial solution satisfies the heat equation at each collocation point. By minimizing this mean squared residual through gradient descent, the network learns to approximate the true solution throughout the domain.

In contrast to our hard BC approach, a soft boundary condition method would require a loss function of the form:

$$\mathcal{L}_{\text{soft}}(\boldsymbol{P}) = \mathcal{L}_{\text{PDE}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{BC}}\mathcal{L}_{\text{BC}}, \tag{8}$$

where $\mathcal{L}_{\text{IC}}$ penalizes deviation from the initial condition, $\mathcal{L}_{\text{BC}}$ penalizes deviation from the boundary conditions, and $\lambda_{\text{IC}}, \lambda_{\text{BC}}$ are weighting hyperparameters that must be carefully tuned. Our hard BC approach eliminates these additional terms entirely.

5. Automatic Differentiation

The partial derivatives in the PDE residual are computed using automatic differentiation provided by JAX [4]. Automatic differentiation (AD) [5] is a computational technique for evaluating derivatives of functions expressed as computer programs, with accuracy up to

machine precision. The key observation behind AD is that any program, regardless of its complexity, can be decomposed into a sequence of elementary operations (addition, multiplication, exponentiation, etc.) for which derivatives are known. By systematically applying the chain rule to these primitive operations, AD computes exact derivatives without symbolic manipulation and without the numerical errors associated with finite-difference methods. AD is typically implemented in two complementary modes:

Forward Mode AD is preferable when the number of input variables is small relative to the number of outputs. The idea is to propagate derivatives forward through the computation: each intermediate variable carries both its value and its derivative with respect to a chosen input.

Reverse Mode AD in contrast, is most efficient when there are many input variables but only a single scalar output. This is the common setting in machine learning, where the goal is to compute the gradient of a loss function with respect to millions of parameters. Reverse Mode works by performing a standard forward pass to record intermediate values, followed by a backward pass that accumulates adjoint sensitivities using the chain rule. This is the mechanism underlying backpropagation in neural networks, and is the method used here.

D. Optimizer

Training a PINN requires minimizing the loss function (7) with respect to all network parameters $\boldsymbol{P}$. We use the Adam optimizer [6], a momentum-based method that adapts the learning rate for each parameter based on estimates of the first and second moments of the gradients.

Adam maintains two moving averages. Let $\mathbf{g}_t = \nabla_{\boldsymbol{P}} \mathcal{L}$ denote the gradient at iteration t. The first moment (mean) and second moment (uncentered variance) are computed as:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t, \quad \mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2, \tag{9}$$

where β_1 and β_2 are decay rates (typically 0.9 and 0.999). Since these accumulators are initialized at zero, they are biased toward zero early in training. Adam corrects this by:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}, \quad \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1-\beta_2^t}. \tag{10}$$

The parameter update rule is then:

$$\boldsymbol{P}_{t+1} = \boldsymbol{P}_t - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}, \tag{11}$$

where η is the learning rate and ϵ is a small constant for numerical stability.

Adam is particularly effective for PINNs because the loss landscape is highly non-convex, with multiple components ($\mathcal{L}_{\text{PDE}}$, $\mathcal{L}_{\text{BC}}$, $\mathcal{L}_{\text{IC}}$) that may have different scales. The adaptive learning rates help balance these competing objectives during training.

E. Activation Functions

The choice of activation function $\sigma(\cdot)$ in the hidden layers significantly influences a PINN's ability to approximate smooth PDE solutions. We investigate four nonlinear activation functions for the hidden layers and use a linear activation in the output layer.

1. *Tanh*

The hyperbolic tangent (tanh) function is defined as

$$\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{12}$$

with derivative

$$\sigma'(z) = 1 - \tanh^2(z). \tag{13}$$

The tanh function maps inputs to $(-1, 1)$ and is infinitely differentiable, making it well-suited for smooth PDE solutions. Its zero-centered output range allows it to represent both negative and positive values, which leads to faster convergence and makes it particularly suitable for representing oscillating solutions like our initial condition $u(x, 0) = \sin(\pi x)$. However, when $|z|$ becomes large, $\tanh'(z) \approx 0$, which can lead to vanishing gradients and slow down learning.

2. *ReLU*

The ReLU function is defined as

$$\sigma(z) = \max(0, z) = \begin{cases} z & \text{if } z > 0, \\ 0 & \text{if } z \leq 0, \end{cases} \tag{14}$$

with derivative

$$\sigma'(z) = \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{if } z \leq 0. \end{cases} \tag{15}$$

ReLU is computationally efficient and avoids vanishing gradients since $\sigma'(z) = 1$ for all positive inputs. However, it is not differentiable at $z = 0$ and produces piecewise linear functions, which may struggle to represent the smooth solutions typical of heat equations. Additionally, ReLU suffers from the "dying ReLU" problem, when inputs are consistently negative, $\sigma'(z) = 0$ causes affected neurons to stop learning entirely.

3. GeLU

The Gaussian Error Linear Unit (GeLU) is defined as

$$\sigma(z) = z \cdot \Phi(z) = \frac{z}{2}\left[1 + \text{erf}\left(\frac{z}{\sqrt{2}}\right)\right], \tag{16}$$

where $\Phi(z)$ is the cumulative distribution function of the standard normal distribution and $\text{erf}(\cdot)$ is the error function. The derivative is

$$\sigma'(z) = \Phi(z) + z\phi(z), \tag{17}$$

where $\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$ is the standard normal probability density function.

GeLU was designed to address ReLU's limitations, particularly the dying ReLU problem and its inability to handle negative values effectively [7]. Unlike ReLU, GeLU is smooth and continuously differentiable everywhere, which facilitates gradient-based optimization during training. These properties make GeLU well-suited for PINNs, where smooth differentiability is essential for accurately computing the spatial and temporal derivatives required in PDE residuals.

4. SiLU

The Sigmoid Linear Unit function (SiLU), is defined as

$$\sigma(z) = \frac{z}{1 + e^{-z}}, \tag{18}$$

with derivative

$$\sigma'(z) = \frac{1 + e^{-z} + ze^{-z}}{(1 + e^{-z})^2}. \tag{19}$$

SiLU is a smooth activation function where the input is modulated by its own sigmoid, which addresses ReLU's limitations [8]. Like GeLU, SiLU is continuously differentiable and non-monotonic, allowing it to handle negative inputs more effectively than ReLU. The self-gating mechanism, where the input is multiplied by its sigmoid, enables better gradient flow during backpropagation and helps mitigate the dying ReLU problem. These properties make SiLU well-suited for PINNs, where smooth differentiability is required for computing accurate PDE residuals.

5. Sine

The sine function is defined as

$$\sigma(z) = \sin(z), \tag{20}$$

with derivative

$$\sigma'(z) = \cos(z). \tag{21}$$

Sine is a periodic, infinitely differentiable function that has gained attention for PINNs. The SIREN architecture demonstrated its effectiveness and accuracy for solving periodic PDEs [9]. For our problem, sine is particularly relevant given the initial condition $u(x, 0) = \sin(\pi x)$.

6. Linear Activation Function

For the output layer, we use a linear activation function:

$$\sigma(z) = z. \tag{22}$$

This allows the network to output any real-valued temperature $u(x, t) \in \mathbb{R}$, which is necessary for solving PDEs.

III. METHOD

A. Explicit Scheme Algorithm

The explicit forward Euler scheme discretizes the spatial domain $[0, 1]$ into N_x intervals with spacing $\Delta x = 1/N_x$, and uses time step $\Delta t = \alpha \Delta x^2$ with $\alpha = 0.4$ to satisfy the stability criterion $\alpha \leq 0.5$. The update rule is:

$$u_i^{n+1} = u_i^n + \alpha(u_{i+1}^n - 2u_i^n + u_{i-1}^n), \quad i = 1, \ldots, N_x - 1$$

with boundary conditions $u_0^n = u_{N_x}^n = 0$ and initial condition $u_i^0 = \sin(\pi x_i)$. We test two spatial resolutions: $N_x = 10$ ($\Delta x = 0.1$) and $N_x = 100$ ($\Delta x = 0.01$), both evaluated to final time $T = 0.5$.

B. PINN Implementation

Our PINN implementation is built using JAX [4] and Flax NNX for automatic differentiation and neural network construction. The architecture consists of a feed-forward neural network wrapped by the hard boundary condition trial solution defined in Equation 6. We chose the SiLU activation function and trained the PINN using 3 hidden layers with 32 nodes and 10 000 collocation points in the domain. This architecture was chosen due to it being the architecture with smallest L^2 error after a parameter sweep. This is explained further in the next Section.

C. Network Architecture

To investigate how network complexity and activation function choice affect PINN performance for the heat equation, we conducted a systematic architecture sweep across varying network depths and widths.

We fixed the training procedure with the following hyperparameters: 10,000 training iterations using the

Adam optimizer with learning rate $\eta = 5 \times 10^{-4}$, and collocation point distributions of $N_{\text{int}} = 1000$ interior points. The temporal domain was set to $T = 0.5$.

The architecture grid consisted of three network widths—32, 64, and 128 neurons per hidden layer—and three network depths—2, 3, and 4 hidden layers—yielding nine distinct architectures. For each architecture, we tested five activation functions: tanh, sine, GeLU, swish, and ReLU. This resulted in a total of 45 unique configurations.

All networks followed the structure described in Section II C, with an input layer accepting (x, t) coordinates, L hidden layers of width W, and a single linear output neuron. The activation function was applied uniformly across all hidden layers, while the output layer used a linear activation to allow unrestricted temperature values.

For each configuration, we trained 10 PINNs with different random seeds (seeds = 0, 2, 3, 4, 8, 9, 16, 21, 29, 42) and report the mean relative $\mathbf{L^2}$ error to ensure robust performance estimates.. After training, we evaluated the relative L^2 error on a uniform evaluation grid of $N_x = 100$ spatial points and $N_t = 100$ temporal points over the domain $[0, 1] \times [0, T]$. The relative L^2 error was computed as:

$$\text{Error}_{L^2} = \frac{\|u_{\text{PINN}} - u_{\text{exact}}\|_{L^2}}{\|u_{\text{exact}}\|_{L^2}}, \tag{23}$$

where u_{exact} is the analytical solution derived in Appendix A.

The results are visualized as heatmaps in Section IV, where each cell represents the mean relative L^2 error for a specific combination of width, depth, and activation function. The configuration achieving the lowest error for each activation function is highlighted with a red border in the corresponding heatmap.

D. Collocation Point Sampling

Collocation points are locations in the spatio-temporal domain where we enforce the PDE. At each training iteration, we sample N_{int} interior points uniformly at random from the domain $(x, t) \in (0, 1) \times (0, T)$:

$$(x_i, t_i) \sim \text{Uniform}([0, 1] \times [0, T]), \quad i = 1, \ldots, N_{\text{int}}$$

Random sampling provides coverage of the entire domain over many iterations while acting as a regularization mechanism. Since we use hard boundary conditions, the trial solution automatically satisfies BC and IC regardless of where points are sampled, eliminating the need to explicitly sample points at $x = 0$, $x = 1$, or $t = 0$. This reduces computational cost compared to soft BC approaches that require separate boundary and initial condition point sets. For evaluation and error computation, we use a uniform grid with $N_x = N_t = 100$ points in each dimension.

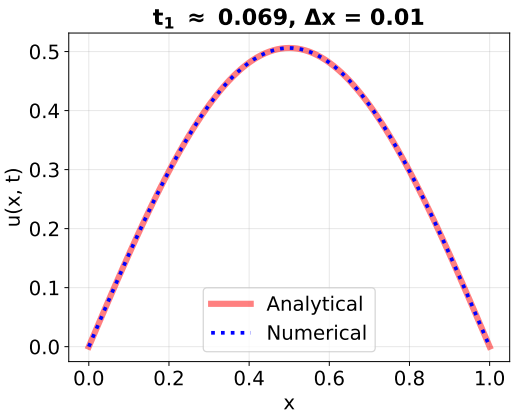

Figure 1. Comparison of explicit forward difference scheme at $t1 = 0.069$, with $\Delta x = 0.01$, and the analytical solution of the diffusion equation.

E. Tools and AI Usage

For code collaboration, we have used GitHub, and our code can be found by following this link [10]. Our code is written in Python, with libraries: Matplotlib [11], NumPy [12], Pandas [13] and JAX [4].

To enhance the quality of this scientific report, we utilized several AI tools throughout the writing and development process. Claude and ChatGPT were used to assist with formulating precise scientific language and improving sentence structure to meet academic writing standards. For code development, we used Claude, ChatGPT, and GitHub Copilot to support debugging processes, optimize code structure, and ensure consistency across our implementations. Additionally, these tools helped generate figures of good quality that maintain visual consistency throughout the document.

IV. RESULTS

A. Comparison of Finite Difference Scheme and analytical solution

We implemented the Finite Difference Scheme in accordance with Sections II B and III A. The comparison of the explicit scheme and the analytical solution are presented in figures 1-6.

We begin by comparing the FD scheme with the analytical solution of the diffusion equation. The comparison is carried out on two spatial grids, corresponding to $\Delta x = 0.1$ and $\Delta x = 0.01$, with respectively $\Delta t = 0.004$ and $\Delta t = 0.00004$ (derived from $\alpha = 0.4$ through the stability criterion). For both grids, we select two time points that lie on the intersection of the resulting time grids, namely $t_1 = 0.069$ and $t_2 = 0.298$. These four systems are shown in Figures 1-4. The first point, t_1, represents a time at which $u(x, t)$ is still smooth but exhibits

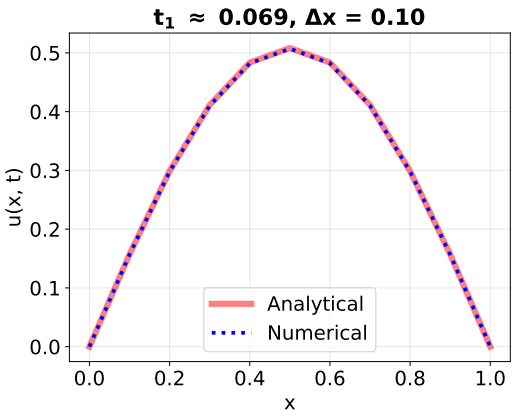

Figure 2. Comparison of explicit forward difference scheme at $t_1 = 0.069$, with $\Delta x = 0.1$, and the analytical solution of the diffusion equation.

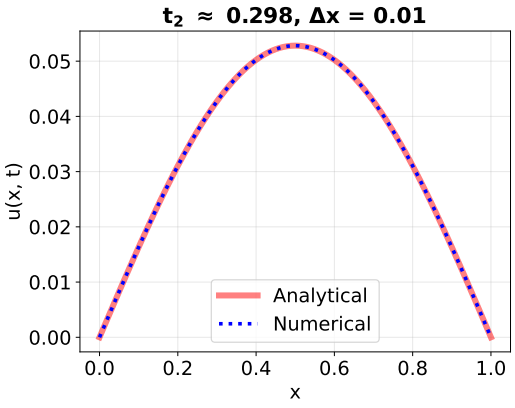

Figure 3. Comparison of explicit forward difference scheme at $t_1 = 0.298$, with $\Delta x = 0.01$, and the analytical solution of the diffusion equation.

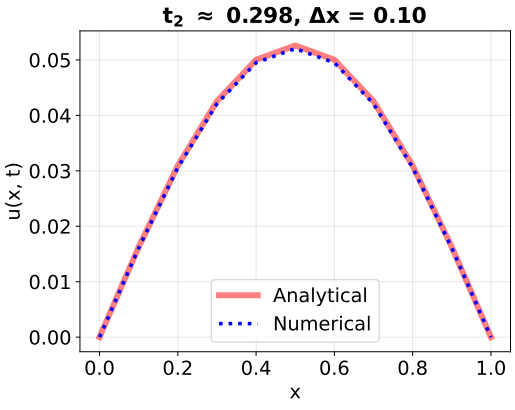

Figure 4. Comparison of explicit forward difference scheme at $t_1 = 0.298$, with $\Delta x = 0.1$, and the analytical solution of the diffusion equation..

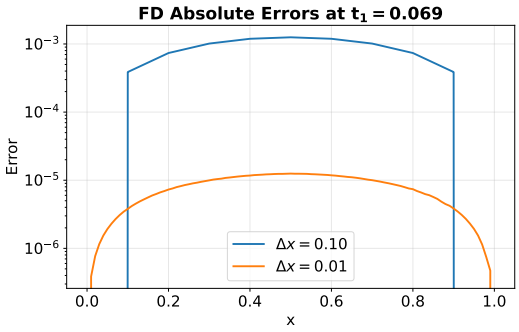

Figure 5. Absolute error of FD scheme at point $t_1 = 0.069$

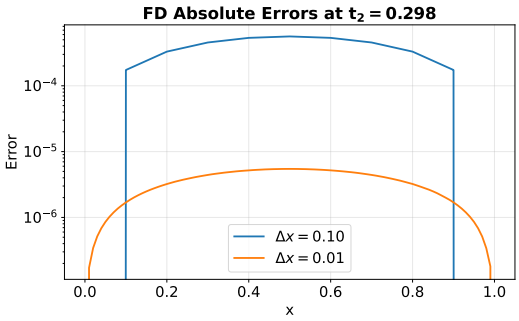

Figure 6. Absolute error of FD scheme at point $t_2 = 0.298$

significant curvature. In contrast, the second point, t_2, corresponds to a state where the solution is almost linear in x, meaning very close to the stationary steady-state.

Figures 5 and 6 show the difference at times t_1 and t_2 between the FD solution and the analytical solution. Each figure contain the absolute error between the solutions for spatial grids $\Delta x = 0.1$ and $\Delta x = 0.01$ of the FD solution.

B. PINN approximation

We implemented the PINN in accordance with Sections II C and III B. The results of the FD scheme and PINNs solution are compared with the analytical solution in Figures 8 and 15 respectively. The analytical solution is shown in Figure 7.

The FD scheme uses $\Delta x = 0.01$ to attain a better resolution in the spatial domain, and consequently improving the temporal resolution. The PINN uses the SiLU activation function and 3 hidden layers with 32 nodes each and 10 000 training steps. This architecture was chosen due to its low L^2 error relative to other architectures.

Figure 8 shows the absolute error of the FD scheme relative to the analytical solution over the space and time domain. The error magnitude remains uniformly small, indicating that the FD scheme provides an accurate approximation to the diffusion equation. We see that the error decreases with time, while peaking at the center of

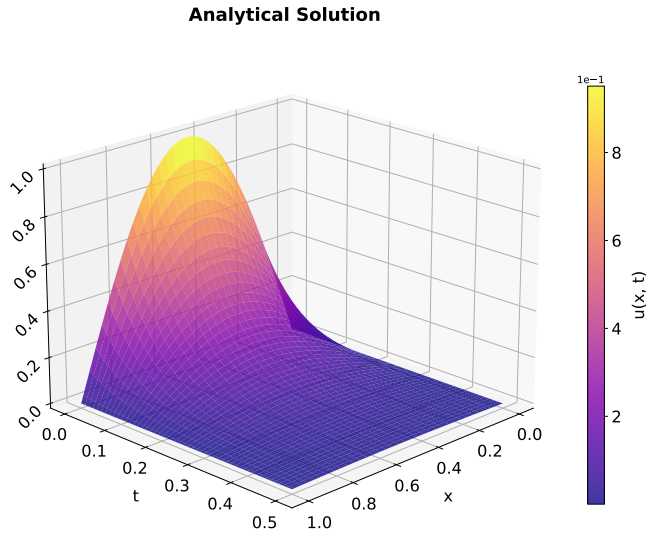

Figure 7. Analytical solution of the heat diffusion equation with initial condition $u(x,0) = \sin(\pi x)$ and boundary conditions $u(0,t) = u(L,t) = 0$.

Table I. Best relative L^2 error for each activation function across all tested architectures (averaged over 10 seeds).

Activation	L^2 Error	Architecture
SiLU	**0.000562**	3 layers, 32 neurons
GeLU	0.000761	2 layers, 64 neurons
Tanh	0.000726	2 layers, 128 neurons
Sine	0.002250	4 layers, 32 neurons
ReLU	0.237841	3 layers, 64 neurons

the spatial domain.

Figure 15 shows the absolute error of the PINN relative to the analytical solution as a function of space and time. Overall, the PINN achieves a low error level, indicating that it successfully approximates the solution to the diffusion equation. The error varies irregularly in time, while having a somewhat smooth shape in space for a given time t.

C. Network Architecture

Figures 10–14 show heatmaps comparing the relative L^2 error as a function of network architecture and activation function. Each cell represents the mean error over 10 random initializations for networks with 2, 3, or 4 hidden layers (using a linear activation function for the output layer) and 32, 64, or 128 neurons per hidden layer, trained for 10000 epochs. Darker colors indicate lower errors and better performance. The model corresponding to the smallest error is the 32 node, 3 layers model with the SiLU activation function. This is the PINN model used to compare the PINN to the analytical solution in figure 15.

The choice of activation function has a dramatic impact on PINN performance for the heat equation. Table I summarizes the best performance achieved by each activation function across all network architectures.

V. DISCUSSION

A. Discrete-Time Comparison Between FD scheme and Analytical Solution

From Figures 1-4, we observe that the FD scheme provides an excellent approximation of the analytical solution both at the highly curved time point t_1 and at the nearly linear time point t_2. As expected, the approximation improves when the spatial resolution is increased (when Δx is reduced), since a finer grid captures the spatial variations of the solution more accurately. In decreasing Δx we also get a better resolution in time.

The corresponding error plots shown in Figures 5 and 6 further illustrate this behavior. They clearly demonstrate that the accuracy increases with decreasing Δx, and they also show that the FD scheme encounters greater difficulty at t_1, where the analytical solution exhibits stronger curvature. This is consistent with the fact that regions with higher spatial variation generally require finer discretization to achieve the same level of accuracy. Another way of describing this is by noting that the initial PDE requires solutions to a second spatial derivative. At lower spatial curvature the spatial second derivative approaches zero. At high curvature, the spatial second derivative is large. The exponential decay term from the analytical solution shows that the variation in the spatial domain is suppressed over time (Figure 7). This meaning the second spatial derivative is less prone to numerical instabilities from the scheme approximations of the derivatives at later times like t_2, when the curvature is small.

B. Continuous Comparison Between FD Scheme and Analytical Solution

The largest errors occur in regions where the analytical solution exhibits strong spatial curvature (see Figure 7), particularly at early times. This behavior is expected, since the centered difference approximation of the second spatial derivative is less accurate in regions with high curvature. As time increases and the solution becomes smoother due to diffusion, the error decreases rapidly throughout the domain. We saw this when discussing Figures 5 and 6.

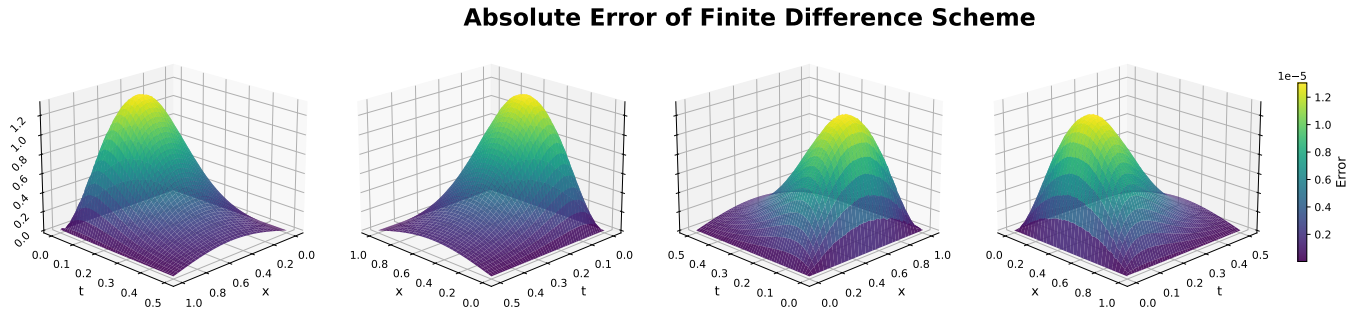

Figure 8. Figure showing error between FD scheme using $\Delta x = 0.01$ and analytical solution on spatial and temporal domain.

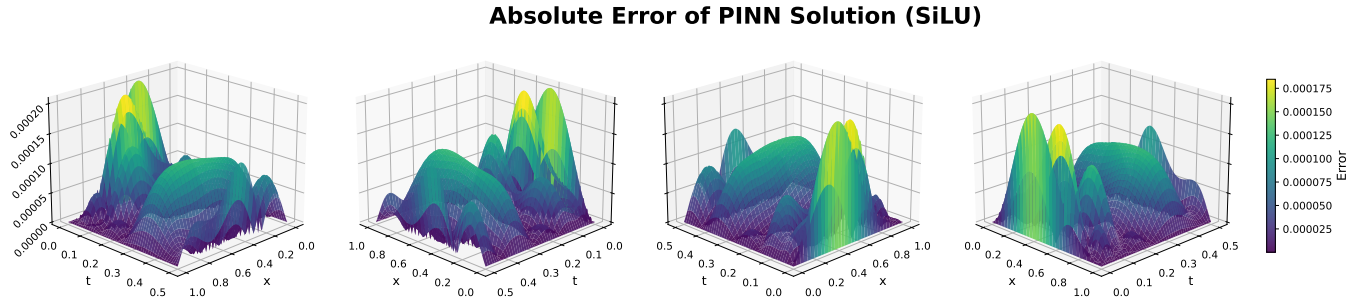

Figure 9. Figure showing error between PINN with 3 hidden layers with 32 nodes each, optimized using SiLU, using 10 000 training steps and 10 000 collocation points and analytical solution on spatial and temporal domain.

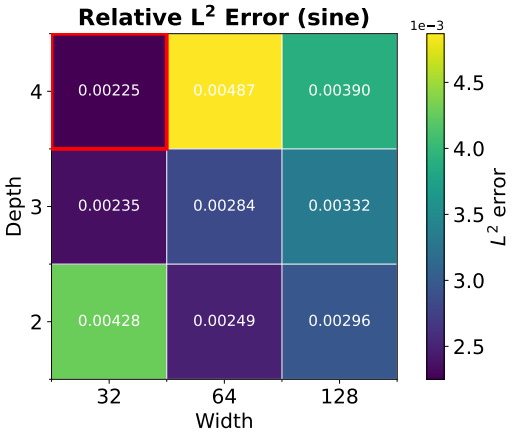

Figure 10. Relative L^2 error for the sine activation function.

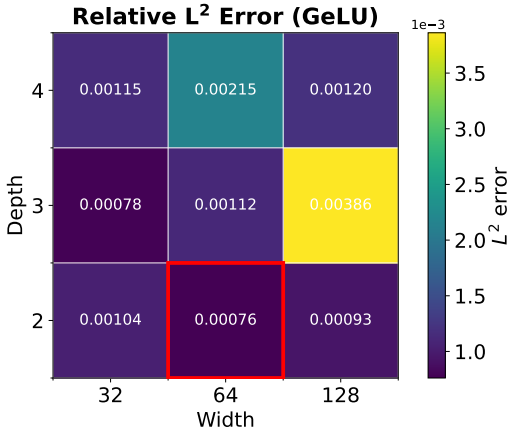

Figure 11. Relative L^2 error for the GeLU activation function.

C. Comparison Between PINN and Analytical Solution

The error in the spatial domain seem to follow smooth curved shapes, suggesting that the error at neighboring x locations depend on each other. Similarly to FD, the largest errors seem to be around the center of the spatial grid (not necessarily symmetric around $x = 0.5$, but away from the boundaries), likely due to the instability of the double derivative at high spatial curvature. This, combined with the hard boundary conditions, contribute to the fairly smooth error curves in the spatial dimension.

In contrast to the FD scheme, the PINN error does not monotonically decrease in time. We see an irregular pattern. Firstly, we see the that the PINN solution seem to match the initial condition. As we increase time, the errors vary a lot. There are two possible reasons for this. Firstly, the PINN is trained at N_{int} (randomly) uniformly placed collocation points. When we evaluate the PINN solution in the grid we do (same grid as FD), we might evaluate the solution in regions where the model had few collocation points. Here, the model is less trained. Sec-

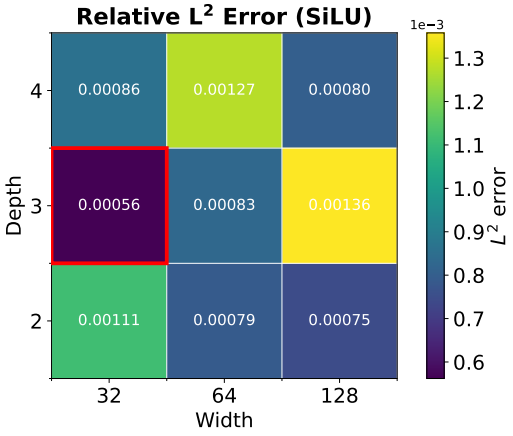

Figure 12. Relative L^2 error for the SiLU activation function.

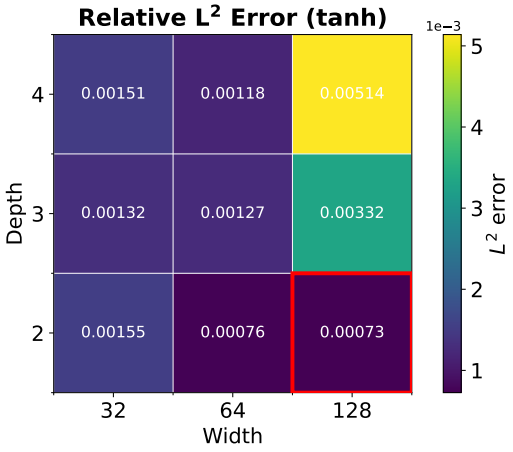

Figure 13. Relative L^2 error for the tanh activation function.

ondly, the boundary/initial conditions could be the reason we only see this in the temporal domain. The BCs in the spatial domain restricts the model at either end. In the spatial domain, we only restrict the model at $t = 0$. We see that the model behaves well here. After this, there is no value that $u(t = T, 0)$ has to approach (here T is the end point).

Increasing the number of collocation points would alter the shape of the error. Firstly, the reason we see many peaks and valleys at low t is likely due to the low number of collocation points not being able to sufficiently resolve the large changes in $u(x, t)$ from the exponential time decay. Secondly, due to no hard end condition, the error would likely increase for higher t. In fact, a similar phenomena can be seen in figure 8, where the error in the spatial domain at high t is largest in the middle, furthest away from the boundary conditions. Therefore, as we move away from $t = 0$, we need the network to learn the model more independently from the initial condition. To conclude, increasing the number of collocation points would smooth out the errors in the temporal

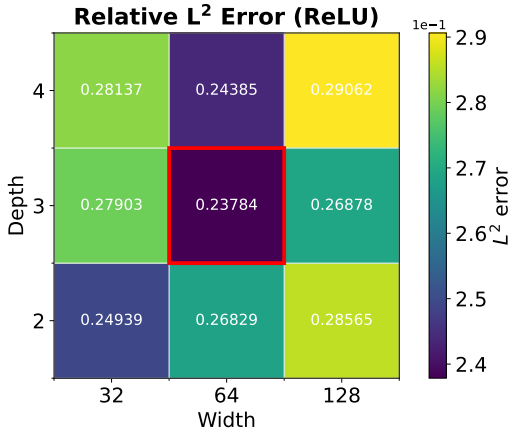

Figure 14. Relative L^2 error for the ReLU activation function.

domain due to higher resolution and decrease the large errors when $u(x, t)$ changes a lot in time at low t, which in turn would demonstrate how the error would increase with time. This can bee seen in Appendix C, where we reran the experiment with $N_{int} = 50000$ compared to $N_{int} = 10000$ and with 15000 iterations to show convergence compared to 10000.

D. Comparison Between FD and PINN Performance

The FD scheme demonstrates a low error approximation of the analytical heat diffusion equation, with smooth errors due to instabilities of numerically calculating second derivatives. In contrast, the PINN error displays localized oscillations, particularly at later times. While the PINN achieves a comparable overall error magnitude, its error distribution reflects the optimization-based nature of the method, where the solution is constrained primarily at discrete collocation points rather than evolved through a time-marching process.

These results highlight a key distinction between the two approaches: the FD scheme naturally enforces the physical smoothing of the diffusion process through its numerical structure, whereas the PINN must learn this behavior indirectly through the loss function. As a consequence, PINNs may exhibit spurious oscillations even when the global error remains small, emphasizing the importance of careful collocation point selection for time-dependent PDEs.

E. Activation Function Performance

The results in Table I reveal a clear hierarchy in activation function performance. Smooth, continuously differentiable activation functions (SiLU, GeLU, and tanh) achieve errors on the order of 10^{-3} to 10^{-4}, while ReLU fails with errors approximately 200–400 times larger (∼

0.25). This dramatic difference confirms the theoretical expectation that piecewise linear activations are fundamentally unsuitable for approximating smooth PDE solutions.

ReLU's poor performance stems from its non-differentiable point at $z = 0$ and zero second derivative almost everywhere. Since the PDE residual requires computing $\partial^2 u/\partial x^2$, ReLU networks cannot accurately represent the smooth exponential decay and sinusoidal spatial structure of the analytical solution in II A.

Among the smooth activation functions, SiLU demonstrates the best performance, achieving a relative L^2 error of ~ 0.0005 in its optimal configuration. GeLU and tanh also perform well, with comparable errors of ~ 0.0007 for their respective best architectures. These three activation functions combine the benefits of smoothness with non-monotonic behavior that helps prevent the dying neuron problem to which ReLU is prone. The results validate these activation functions' effectiveness for neural network applications requiring smooth function approximation, particularly for PINN-based PDE solving where computing accurate spatial and temporal derivatives is essential.

The sine activation function achieves moderate performance (~ 0.002), approximately 3–4 times worse than the best smooth activations. While sine's periodicity might seem advantageous given the initial condition $u(x, 0) = \sin(\pi x)$, its performance does not exceed that of SiLU, GeLU or tanh. This suggests that general-purpose smooth activations are sufficient for this problem, and problem-specific periodicity in the activation function does not provide substantial benefits.

F. Comparing Network Complexity

The stability of the relative L^2 error varies considerably across activation functions and network architectures. We quantify stability by comparing the ratio of maximum to minimum errors achieved across all tested configurations.

ReLU demonstrates the highest stability but at the cost of accuracy. As shown in Figure 14, the maximum error is only approximately 1.2 times larger than the minimum, indicating consistent performance across architectures. However, ReLU favors moderate width with significant depth, achieving its lowest errors of $\sim$0.23 and $\sim$0.24 for networks with 3 and 4 hidden layers, respectively, each containing 64 nodes per layer. This behavior aligns with ReLU's known limitations for smooth PDE solutions. As discussed in Section II E 2, ReLU produces piecewise linear functions due to its non-differentiable nature at $z = 0$, which inherently struggles to represent the smooth, oscillating solutions characteristic of the heat equation.

Sine and SiLU exhibit moderate stabilitys with improved accuracy compared to ReLU. For sine (Figure 10), the maximum error is approximately 2.2 times larger than the minimum, while SiLU (Figure 12) shows a ratio of approximately 2.4. Both activation functions maintain reasonable performance across different network depths and widths. The superior performance of sine is expected given our initial condition $u(x, 0) = \sin(\pi x)$, the periodicity and infinite differentiability of sine naturally align with the problem's oscillatory nature, as discussed in Section II E 5. Similarly, SiLU's continuous differentiability and self-gating mechanism (Section II E 4) enable better gradient flow during backpropagation, facilitating accurate computation of the spatial and temporal derivatives in the PDE residual.

Tanh shows the poorest stability but achieves the best accuracy, with the maximum error being approximately 7 times larger than the minimum. This high variability suggests strong sensitivity to architectural choices. As shown in Table I, tanh achieves its absolute lowest L^2 error of 7.3×10^{-4} using 2 hidden layers with 128 nodes per layer, with nearly identical performance (7.6×10^{-4}) for 2 layers with 64 nodes seen in Figure 13. For deeper networks, tanh favors narrower architectures, though the errors increase significantly to approximately 1.0×10^{-3}.

This behavior reflects tanh's mathematical properties discussed in Section II E 1. The zero-centered output range of tanh $\in (-1, 1)$ and its infinite differentiability make it particularly well-suited for representing smooth, oscillating solutions like $u(x, 0) = \sin(\pi x)$. However, as noted in Section II E 1, the vanishing gradient problem for $|z| \gg 0$ (where $\tanh'(z) \approx 0$) can slow convergence, explaining the architectural sensitivity we observe.

Comparing across activation functions, we find a clear trade-off between stability and accuracy. ReLU's minimum error ($\sim$0.238) significantly exceeds tanh's maximum error (5.1×10^{-3}) by more than two orders of magnitude, despite ReLU's superior stability. This suggests that for smooth PDE solutions, accepting some architectural sensitivity in exchange for substantially improved accuracy is worthwhile. The smooth, continuously differentiable activation functions (tanh, sine, SiLU) consistently outperform ReLU, confirming that smoothness in the activation function is critical for accurately approximating smooth PDE solutions through automatic differentiation, as discussed in Section II C 5.

G. Neural Networks as PDE solvers

From the previous discussions, the PINN and the FD scheme proved to be good at approximating the analytical solution to the heat diffusion equation. In this article we have seen that the FD scheme provided the solution closest to the analytical. This method requires low computational cost and doesn't require an architecture sweep to attain a favorable model from the PINN. However, when exploring more complex partial differential equations the solution might not be smooth and complex geometries would make it challenging to design a mesh that satisfies the stability criteria. In these cases numer-

ical schemes as forward Euler and FD can fall short. We can then asses weather a PINN can be used.

From the results in IV B, the PINN provide a relativity good approximation to the analytical solution, without the need to implement a mesh-grid that satisfies the stability criteria. This is a big advantage when dealing with more complex systems, of higher dimensions. In this simple case, we were able to design a trial solution that automatically enforced the boundary- and initial conditions given. In this way, we simplified the loss function and removed the need for hyperparameter tuning of λ_{IC} and λ_{BC}, making training faster and more stable, with near zero error near the boundaries. However, this design of the trial function would immediately break down if we changed the geometry of the system, so our PINN is not generalizable to similar problems. For this we would need to either change the trial function, or implement the physics of the system into the loss function as mentioned in the theory II. This is not the only limitation of our PINN, as we discussed in V D, our trial function ensures smooth errors in the spatial domain, whereas the temporal domain is unstable. Our discussion indicates that the needed number of collocation points in the respective dimensions are of great importance. In this case, because of the exponential diffusion of $u(x,t)$ in the time domain and the lack of a "anchor point" at $T = 0.5$, the PINN approximation displays oscillations and diverging behavior in the time domain, due to insufficient collocation points along the time axis. This illustrates the importance of choosing the right trial function for the system in question, as well a suitable number of collocation points in each dimension. The discussions in V E and V F further illustrates the need for careful consideration of choice of activation function and network complexity, to the problem at hand.

VI. CONCLUSION

In this work we have successfully approximated the one-dimensional heat equation using both finite difference methods and Physics-Informed Neural Networks (PINNs), demonstrating the strengths and limitations of each approach for time-dependent partial differential equations. We derived the analytical solution to the heat diffusion equations as a benchmark. We then created a forward difference scheme to approximate the solution. The scheme displayed high accuracy (error $\sim 10^{-5}$), with errors decreasing with grid refinement. We found that performance decreased in regions with high spatial curvature, due to the instability of computing second derivatives. In the temporal domain we saw that this effect was more pronounced at earlier times, since the variation in the spatial domain is suppressed over time due to the exponential nature of heat diffusion.

We then developed a PINN, where the boundary- and initial conditions were wrapped in the trial solution $u(x,t,N)$, with $N(x,t,\mathbf{P})$ being the neural network output. The optimal PINN configuration was found by performing an architecture sweep across network widths, layers and activation functions. The optimal configuration was found to be a network of three layers, with 32 nodes each, using the SiLU activation function. This configuration achieved comparable performance (error $\sim 10^{-4}$) to the FD Scheme, when compared to the analytical benchmark.

The comprehensive architecture sweep across 45 configurations reveals that activation function choice is critical for PINN performance. Smooth, continuously differentiable functions (SiLU, GeLU, tanh) achieve relative L^2 errors on the order of 10^{-3} to 10^{-4}, while ReLU fails with errors approximately 200–400 times larger.

By analyzing the error distribution in the spatio-temporal domain we saw that the choice of trial function and the number and distribution of collocations points in the domain are of great significance to performance. We discussed how choosing the appropriate architecture, activations, trial function and the collocation points require analytical insight into the specific problem at hand. We discussed how PINNs can offer favorable aspects into PDE solving, like its meshless nature and encoded physical elements in trial solutions. The fact that we can ignore grid points means that a PINN is favorable when working with higher dimensions, where traditional methods, like FD, require grids that scale as N^n with n dimensions. This also tackles complex geometries. The PINNs also have the potential of deriving unknown physical parameters in the system (inverse problems). These can be inferred through training and compared to data. In conclusion, we have shown that PINNs can solve PDEs accurately with appropriate tuning, while also having the potential to solve more complex systems where traditional methods struggle.

To expand this work further, a test of different trial solutions as well as expanding the heat diffusion problem to higher dimensions and/or more complex geometries to investigate how the relative performance between PINNs and traditional methods varies with a more complex problem. Additionally, exploring PINNs potential for inferring physical parameters (such as thermal diffusivity) from measurements, instead of, or in addition to, an analytical solution. These types of investigations can include, or lead to, an attempt of solving known problems like the Navier-Stokes equation, with appropriate implementation of the known physics.

[1] T. Botarelli, M. Fanfani, P. Nesi, and L. Pinelli, Using physics-informed neural networks for solving navier-stokes equations in fluid dynamic complex scenarios,

Engineering Applications of Artificial Intelligence **148**, 110347 (2025).
[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis, 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 (2019).
[3] S. Cuomo, V. Schiano Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, Scientific machine learning through physics–informed neural networks: Where we are and what's next, Journal of Scientific Computing **92**, 88 (2022).
[4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, Jax: Composable transformations of python+numpy programs (2018).
[5] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, Automatic differentiation in machine learning: A survey, arXiv preprint arXiv:1502.05767 10.48550/arXiv.1502.05767 (2018).
[6] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 10.48550/arXiv.1412.6980 (2017).
[7] N. Keta, Deep learning: Gelu (gaussian error linear unit) activation function (2023), accessed: Dec. 11, 2025.
[8] Silu and gelu activation function in transformers (2025), accessed: Dec. 11, 2025.
[9] M. Mommert, R. Barta, C. Bauer, M.-C. Volk, and C. Wagner, Periodically activated physics-informed neural networks for assimilation tasks for three-dimensional rayleigh–bénard convection, Computers & *Fluids* **283**, 106419(2024).
[10] L. L. Storborg, H. Haug, C. A. Falchenberg, and S. S. Thommesen, Project 3, `https://github.com/livelstorborg/FYS-STK4155/tree/main/project3` (2025), gitHub repository.
[11] J. D. Hunter, Matplotlib: A 2d graphics environment, Computing in Science & Engineering **9**, 90 (2007).
[12] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, Array programming with NumPy, Nature **585**, 357 (2020), `https://doi.org/10.1038/s41586-020-2649-2`.
[13] T. pandas development team, pandas-dev/pandas: Pandas (2025), `https://doi.org/10.5281/zenodo.16918803`.

Appendix A: Deriving Analytical Solution to PDE

We consider the heat equation

$$\begin{cases} u_t = u_{xx}, & t > 0,\ x \in [0, L], \\ u(x, 0) = \sin(\pi x), & 0 < x < L, \\ u(0, t) = u(L, t) = 0, & t \geq 0, \end{cases}$$

and seek a solution by separation of variables,

$$u(x, t) = X(x)T(t).$$

Substituting into the PDE gives

$$X(x)T'(t) = X''(x)T(t) \quad \Rightarrow \quad \frac{X''(x)}{X(x)} = \frac{T'(t)}{T(t)} = -\lambda,$$

for some separation constant $-\lambda$. This yields the two ODEs

$$T'(t) + \lambda T(t) = 0, \quad X''(x) + \lambda X(x) = 0.$$

We start by solving the spatial part. From the boundary conditions

$$\begin{cases} u(0, t) = 0 \Rightarrow X(0)T(t) = 0 \Rightarrow X(0) = 0, \\ u(L, t) = 0 \Rightarrow X(L)T(t) = 0 \Rightarrow X(L) = 0, \end{cases}$$

we obtain the boundary value problem

$$X''(x) + \lambda X(x) = 0, \quad X(0) = X(L) = 0.$$

We set

$$\lambda = \mu^2, \quad \mu > 0,$$

which gives

$$X''(x) + \mu^2 X(x) = 0 \quad \Rightarrow \quad X(x) = A\cos(\mu x) + B\sin(\mu x).$$

We can then impose boundary conditions :

$$X(0) = 0 \Rightarrow A = 0, \quad X(L) = 0 \Rightarrow B\sin(\mu L) = 0.$$

For nontrivial solutions ($B \neq 0$), we need

$$\sin(\mu L) = 0 \Rightarrow \mu L = n\pi, \quad n = 1, 2, \ldots$$

so that

$$\mu_n = \frac{n\pi}{L}, \quad \lambda_n = \mu_n^2 = \left(\frac{n\pi}{L}\right)^2,$$

and the corresponding eigenfunctions are

$$X_n(x) = \sin\left(\frac{n\pi x}{L}\right), \quad n = 1, 2, \ldots$$

Next, we solve the time equation for each λ_n:

$$T_n'(t) + \lambda_n T_n(t) = 0 \Rightarrow T_n(t) = C_n e^{-\lambda_n t} = C_n e^{-\left(\frac{n\pi}{L}\right)^2 t}.$$

The general solution is then a combination of these modes:

$$u(x,t) = \sum_{n=1}^{\infty} C_n e^{-\left(\frac{n\pi}{L}\right)^2 t} \sin\left(\frac{n\pi x}{L}\right),$$

where the coefficients $\{C_n\}$ are determined by the initial condition.

We now choose L=1. We then get

$$u(x,t) = \sum_{n=1}^{\infty} C_n e^{-(n\pi)^2 t} \sin(n\pi x),$$

and at $t = 0$ we have

$$u(x,0) = \sum_{n=1}^{\infty} C_n \sin(n\pi x) = \sin(\pi x).$$

By orthogonality of the sine functions, this implies

$$C_1 = 1, \qquad C_n = 0 \text{ for } n \geq 2.$$

Thus the solution reduces to a single mode,

$$u(x,t) = e^{-\pi^2 t} \sin(\pi x), \quad 0 < x < 1,\ t \geq 0.$$

Appendix B: Deriving Numerical Scheme

We derive the finite difference scheme by applying Taylor expansions in time and space to $u(x,t)$ about the grid points, and then truncating these series at the desired order of accuracy. This allows us to replace the spatial and temporal derivatives in the PDE with discrete approximations of a chosen precision. We will derive a Forward Euler (FE) scheme for the time dependent part, and a Centered Difference scheme for the spacial part.

Forward Euler in Time

To derive the Forward Euler approximation for u_t, we perform a Taylor expansion in time around t:

$$u(x,t+\Delta t) = u(x,t) + \Delta t\, u_t(x,t) + \frac{\Delta t^2}{2} u_{tt}(x,t) + \mathcal{O}(\Delta t^3).$$

Solving for the time derivative gives

$$u_t(x,t) = \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} + \mathcal{O}(\Delta t).$$

Dropping the truncation error yields the Forward Euler approximation,

$$u_t(x,t) \approx \frac{u_i^{n+1} - u_i^n}{\Delta t}.$$

Centered Difference in Space

Next we approximate the second spatial derivative by expanding $u(x,t)$ in space about x_i:

$$u(x_i \pm \Delta x, t) = u(x_i,t) \pm \Delta x\, u_x(x_i,t) + \frac{\Delta x^2}{2} u_{xx}(x_i,t) \pm \frac{\Delta x^3}{6} u_{xxx}(x_i,t) + \mathcal{O}(\Delta x^4).$$

Adding the expansions cancels the odd terms:

$$u(x_i \pm \Delta x, t) = 2u(x_i,t) + \Delta x^2 u_{xx}(x_i,t) + \mathcal{O}(\Delta x^4).$$

Solving for u_{xx} gives the centered second-order approximation

$$u_{xx}(x_i,t_n) \approx \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}.$$

Explicit Finite Difference Scheme

Inserting the Forward Euler and centered difference approximations into the heat equation

$$u_t = u_{xx},$$

we obtain

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}.$$

Solving for u_i^{n+1} yields the explicit update rule

$$u_i^{n+1} = u_i^n + \alpha\big(u_{i+1}^n - 2u_i^n + u_{i-1}^n\big), \qquad \alpha = \frac{\Delta t}{\Delta x^2}.$$

This is the Forward Euler–Centered Difference finite difference scheme used in the numerical experiments.

Appendix C: SiLU Big Run

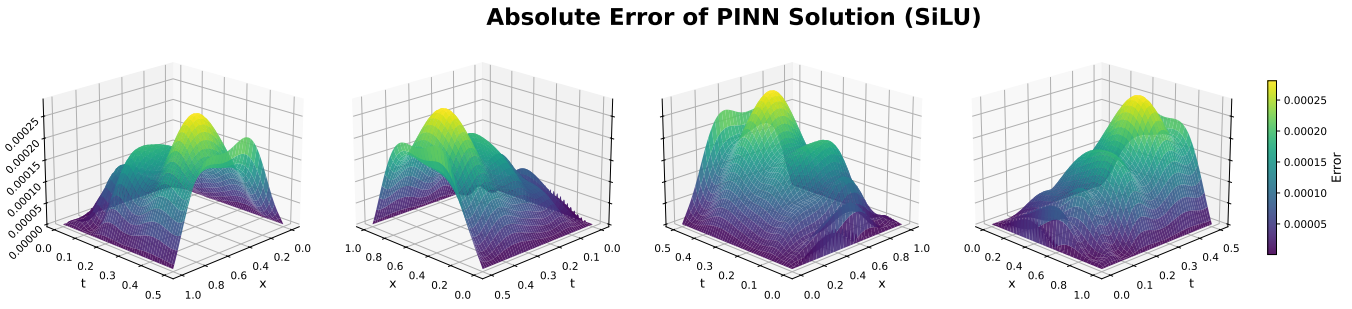

Figure 15. Figure showing error between PINN with 3 hidden layers with 32 nodes each, optimized using SiLU, using 15 000 training steps and 50 000 collocation points and analytical solution on spatial and temporal domain.