Solving the Navier-Stokes equations using PINNs and PINNsFormer

Vinicius Nunes Pereira
Department of Informatics
Federal University of Espírito Santo
Vitória, Brazil
vinicius.pereira.10@edu.ufes.br

Gabriel Inácio Barboza
Department of Informatics
Federal University of Espírito Santo
Vitória, Brazil
gabriel.i.barboza@edu.ufes.br

***Abstract*—In recent years, Physically Informed Neural Networks (PINNs) have emerged as a promising alternative to traditional methods for numerically solving differential equations by embedding physical laws into neural network training. However, the default PINNs architecture, which relies on a feedforward neural network to minimize residual errors from space-time domain samples, faces challenges in accurately capturing spatial-temporal dependencies, especially in complex fluid dynamics problems. The transformer architecture, renowned for its self-attention mechanism, has demonstrated a remarkable ability to model sequential dependencies and long-range correlations across diverse data types. PINNFormer integrates the original PINNs framework with the advantages of the multi-head attention mechanism, enhancing the network's capacity to represent intricate spatial-temporal relationships. In this paper, we investigate the performance of PINNFormer compared to traditional PINNs by analyzing the numerical solutions for the transient cavity flow problem — a benchmark case of the Navier-Stokes equations representing incompressible fluid flow inside a square cavity. We highlight key differences between architectures, assess convergence rates, and examine the accuracy of the predicted solutions. The results don't indicate a substancial improvement on the numerical solution provided by PINNsFormers over vanilla PINNs, suggesting that further investigation is necessary.**

***Index Terms*—PINNs, PINNsFormer, Scientific Machine Learning.**

I. Introduction

Many real-world problems can be effectively modeled using differential equations, which serve as powerful mathematical tools to describe dynamic systems and phenomena across a wide range of disciplines. Fields such as fluid mechanics, biological dispersion, and finance frequently rely on differential equations to develop models that capture the underlying behavior of complex systems. However, many differential equations, particularly those that describe high-dimensional or nonlinear systems, lack known analytical solutions. This limitation has driven the development and widespread adoption of numerical methods that provide approximate solutions to these equations. Traditional numerical techniques, such as the finite difference method and the finite element method, have long been the cornerstone of computational approaches to solving differential equations. Although these methods are robust and well-established, they often require significant computational resources, and their performance may deteriorate when applied to complex domains, high-dimensional problems, or systems with intricate boundary conditions.

The emergence of machine learning, particularly neural networks, has introduced new possibilities for solving differential equations. The capacity of neural networks to approximate complex functions makes them a promising tool for capturing the behavior of dynamic systems governed by differential equations. Although the use of neural networks for this purpose can be traced back to the late 1980s with the demonstration that neural networks are universal approximators [1], it was not until 2017, with the introduction of physically informed neural networks (PINNs) [2], that this approach gained widespread attention. PINNs integrate physical laws, expressed as differential equations, directly into the neural network's loss function, enabling the model to learn solutions that are consistent with the underlying physics. This approach offers several advantages over traditional numerical methods, including the ability to handle high-dimensional problems, adapt to irregular domains, and seamlessly incorporate additional constraints or data.

Despite these benefits, traditional feedforward PINNs face limitations, particularly in capturing temporal dependencies and long-range interactions, which are common in many physical systems [3]. As the complexity of the problem increases, PINNs may struggle to effectively propagate information across spatial and temporal dimensions, leading to convergence issues and reduced accuracy in capturing intricate physical behaviors.

To address these challenges, [3] introduced Physically Informed Transformers (PINNsFormer), which build on the transformer architecture originally developed for natural language processing [4]. Transformers have revolutionized sequential data modeling through their self-attention mechanism, which allows the network to weigh the importance of different input elements when generating predictions. By leveraging this mechanism, PINNsFormer transform point-wise inputs into pseudo-sequences and employ multiple multi-head attention layers to better model dependencies across time and space. This architectural innovation enables PINNsFormer to capture complex spatial-temporal relationships more effectively, making them particularly well-suited for problems in fluid dynamics, wave propagation, and other systems with strong spatio-temporal interactions.

This paper aims to provide a comparison of the mechanisms underlying PINNs and PINNsFormer, highlighting their respective strengths and limitations. Additionally, we evaluate the performance of PINNs and PINNsFormer in solving the cavity flow problem, a classic benchmark in fluid mechanics. By analyzing the accuracy, efficiency, and robustness of these methods, we seek to provide insights into their applicability to real-world problems and identify potential areas for future improvement. Our findings contribute to the growing body of research on machine learning-based approaches to solving differential equations, offering a deeper understanding of how these innovative methods can complement or even surpass traditional numerical techniques.

The rest of this paper is organized as follows: Section II presents the fundamental theory behind Physically Informed Neural Networks , compares PINNsFormer novel mechanisms with the original Transformer architecture, and discusses the Navier-Stokes equations and the benchmark problem. Section III details the implementation of both PINNs and PINNsFormer, as well as the methodology adopted to compare both frameworks. Section IV outlines the experimental setup and the metrics used to evaluate the numerical solutions. Section V analyzes the experimental results. Finally, Section VI offers insights and conclusions.

II. Literature Review

A. PINNs

Presented in Figure 1, the basic framework of Physically Informed Neural Networks (PINNs) consists of transforming the differential equation model into residual functions, which represent the deviations from the governing physical laws. These residual functions are incorporated into the loss function, guiding the neural network to approximate solutions that aim to satisfy both the physical constraints and the available observational data. By applying the backpropagation algorithm, the framework searches for the optimal combination Θ of weights and biases that minimizes these residual functions as defined by Figure 1. The neural network is then trained on a set of sample points collected across the space-time domain, enabling it to approximate a solution that satisfies the underlying physical constraints and the available observational data.

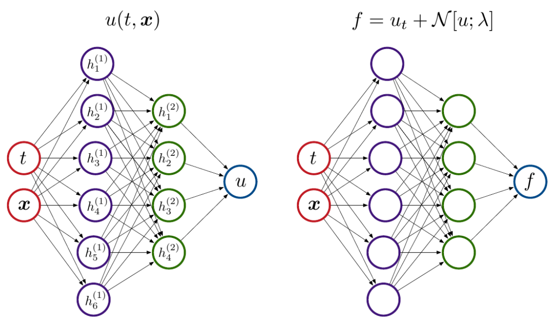

Fig. 1. PINNs schematic.

1) Loss Function: The loss function of PINNs generally includes the residual of the equation and the initial/boundary conditions. Thus, given a coordinate input x at time t and a set θ of weights and biasses, we define the loss function as:

$$J(\theta) = \omega_L J_L(x,t,\theta) + \omega_B J_B(x,t,\theta) + \omega_I J_I(x,t_0,\theta)$$

- $J_L(x,t,\theta)$, $J_B(x,t,\theta)$ and $J_I(x,t_0,\theta)$ are the squared value of the residual of the differential equations, boundary and initial conditions, respectively, measured in the $\|\cdot\|_{\Omega\times I}$ norm (for example, the Euclidian norm).
- The parameters ω_L, ω_B and ω_I are weights that can be used to eliminate the imbalance among the different loss terms.

Different implementations may apply some changes on the formula above. For example, different boundary conditions could be multiplied by different weights to balance the loss if there is a considerable difference in the quantity of data for each boundary [5].

2) Collocation points: Colloction points are a set of sampling points that discretize the difference equation space-time domain. Common sampling techniques includes the Latin Hyper Cube, Hammersley sequencea and Sobol sequence [5]. This is the approach presented by [6] for direct problems. For details on inverse problems, consult [2]. Figure 2 provides a visual representation of collocation points across the space-time domain.

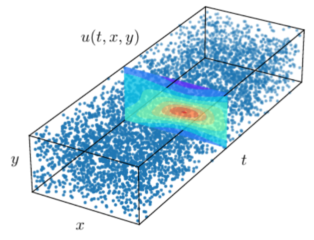

Fig. 2. Representation of collocation points throught time.

B. PINNsFormer

PINNsFormer combine PINNs with the power of multi-head attention to incorporate temporal dependencies when solving partial differential equations (PDEs). They achieve this through three key components: a pseudo-sequence generator, a spatio-temporal mixer, and a wavelet activation function. The architecture follows an encoder-decoder structure, where the encoder consists of stacked multi-head attention layers to learn spatio-temporal dependencies, while the decoder, similar to a vanilla Transformer, also uses multi-head attention to selectively focus on relevant dependencies. By integrating these mechanisms, PINNsFormer significantly enhances the accuracy and generalization of PDE solutions, overcoming the temporal limitations of traditional PINNs.

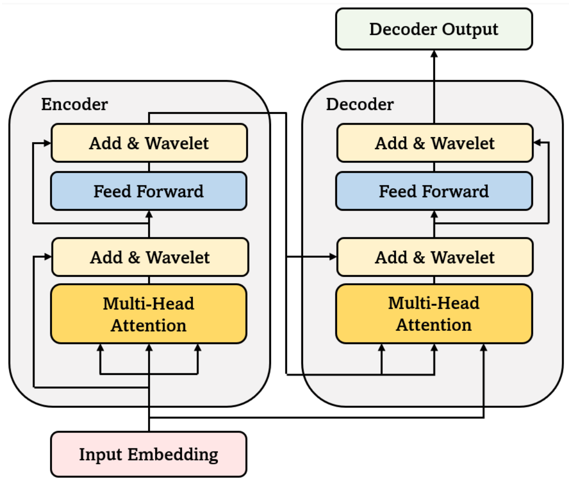

Fig. 3. PINNsFormer archtecture.

1) Pseudo-Sequence Generator: The pseudo-sequence generator converts pointwise spatio-temporal inputs into sequential data, enabling compatibility with Transformer-based architectures. For a given spatial input $x \in R^d$, a temporal input $t \in R$ and a $k \in N$ the pseudo-sequence generator maps an input (x, t) to a set of k points:

$$[x, t] \longmapsto ([x, t], [x, t + \Delta t], [x, t + 2\Delta t], ..., [x, (k - 1)\Delta t]$$

These points are then concatenated, forming an input that extends a single spatio-temporal point into a sequence, effectively transforming the problem into a sequential learning task. This allows the network to model temporal dependencies by leveraging the attention mechanism over multiple discrete time steps [3].

The hyper-parameters k and Δt play a crucial role in the network's performance. The parameter k defines the number of steps to look ahead, while Δt determines the temporal gap between consecutive points. Selecting high values for k and can significantly increase the computational complexity of PINNsFormer, whereas choosing small values may impair the network's ability to capture long-range dependencies [3]. Therefore, proper tuning of these hyper-parameters is essential to balance model complexity and temporal awareness.

2) Spatio-Temporal Mixer: Playing a role similar to embeddings in natural language processing, the spatio-temporal mixer mitigates the limitations of low-dimensional PDE features — such as spatial coordinates and time — by employing a fully connected Multi-Layer Perceptron (MLP). This MLP projects the inputs into a higher-dimensional space, effectively mixing all raw spatio-temporal features. As a result, the network's capacity to capture intricate spatial and temporal relationships is enhanced.

3) The wavelet activation function: While the *ReLU* activation function is the most common choice for transformers [7], it is not well-suited for PINNs [8]. To address this issue, [3] introduced The wavelet activation function, which

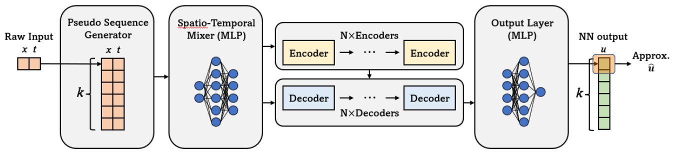

Fig. 4. PINNsFormer input processing.

leverages Fourier properties to enhance PINNs' approximation capabilities, acting as a universal function approximator. The activation function is defined as:

$$wavelet(x) = \omega_1 \sin(x) + \omega_2 \cos(x)$$

Where ω_1 and ω_2 are learnable parameters. A preliminary demonstration of the *wavelet* activation function's capacity as a universal function approximator is presented in [3].

C. Navier-Stokes Equations

The Navier-Stokes equations, formulated independently by Claude-Louis Navier in 1822 and George Gabriel Stokes in 1845, are a set of parabolic partial differential equations that govern the motion of incompressible fluid flow in two and three dimensions. These equations are derived from the principles of conservation of momentum and mass, capturing the intricate balance between inertial forces, viscous forces, and external influences acting on fluid particles.

The Navier-Stokes equations play a fundamental role in fluid dynamics, being applied to model the behavior of fluids such as air and water in various scientific and engineering applications, including aerodynamics, weather forecasting, oceanography, and turbulence modeling. However, solving these equations analytically is notoriously challenging due to their nonlinear nature, especially in three dimensions. In fact, proving the existence and smoothness of solutions for the three-dimensional Navier-Stokes equations is one of the unsolved problems in the Millennium Prize set by the Clay Mathematics Institute, with a \$1 million prize offered for a definitive proof.

A special case of the Navier-Stokes Equations is the transient lid-driven cavity flow. This problem is a classic benchmark for testing and validating numerical methods. It involves studying the flow of a viscous, incompressible fluid in a square or rectangular cavity where the top wall (the "lid") moves tangentially at a constant velocity, while the other walls are stationary. This problem can be modeled using the following equations.

1) Momentum Equation:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\nabla p + \frac{1}{Re} \Delta u$$

This equation represents the balance of forces (inertial, pressure, and viscous forces) in the fluid.

- u is the velocity vector field of the fluid.
- ∇u represents the gradient of the velocity field.
- $u \cdot \nabla u$ is the convective term, which accounts for the nonlinear advection of momentum.

- p is the pressure field.
- ∇p is the pressure gradient, which drives the flow.
- $\frac{1}{Re}$ is the viscous term, where Δu is the Laplacian of the velocity field, and Re is the Reynolds number, a dimensionless parameter that characterizes the flow regime (laminar vs. turbulent).

2) Continuity Equation:

$$\nabla \cdot \ u = 0$$

This equation expresses the conservation of mass for an incompressible fluid.

- $\nabla \cdot u$ is the divergence of the velocity field, and setting it to zero means that the fluid is incompressible (i.e., the density of the fluid remains constant).

III. Methodology

This section provides implmentation details of both architectures and the methodoly applied to compare PINNs and PINNsFormer.

A. Implementation

Our first strategy for implementating both architectures consisted in using the DeepXDE framework [5]. However, due to difficulties faced during the process of adapting the PINNsFormer archtecture to DeepXDE workflow, we switched up to a pure PyTorch [9] implementation using the baseline implementation provided by [6] (PINNs) and [3] (PINNsFormer). The source code for this paper can be found at https://github.com/DravenPie/PINNFormers-Navier-Stokes-Equations.

B. Velocities and Pressure Heatmaps

We evaluated both models by analyzing the numerical solutions when it reaches stability at $t = t_f$ by analyzing the heatmaps for both the velocities and pressure. We expected horizontal flow at the lid for velocity u, vortexes in opposite directions for velocity v, pressure p concentrated at the top right corner of the cavity and negative concentration at the left top corner. Those are the expected behaviors of variable u, v and p for laminar flows.

C. Numerical Comparisons

We compare velocities u and v predicted by both PINNs and PINNsFormer against the reference values provided by [10] for $Re = 100$. These are the expected values for velocity u (horizontal) alongside axis y and for velocity y (vertical) alongside axis x.

D. Loss Analysis

One of the main advantages claimed by [3] is that PINNsFormer achieves faster convergence, compensating the more costly training process compared to PINNs. In this work, we analyze the convergence rate of both architectures to assess the validity of this claim.

IV. Experiments

A. Problem settings

Following the problem setting proposed by [10], the domain is the rectangle $[0, 1]^2$, $Re = 100$, with the following Dirichlet boundary conditions:

Stationary walls:

$$v(0, y, t) = v(x, 0, t) = v(1, y, t) = v(x, 1, t) = 0$$

$$u(0, y, t) = u(x, 0, t) = u(1, y, t) = 0$$

Cavity top wall (the "lid") moving tangentially at a constant velocity:

$$u(x, 1, t) = 1$$

Preassure at the bottom of the cavity:

$$p(0, 0, t) = 0$$

The time interval is $[0, 2]$ with the following initial conditions:

$$u(x, y, 0) = 0 \qquad u(x, y, 0) = 0$$

B. Network Configuration

Both PINNs and PINNsFormer were trained using Adam optimizer for 15000 iterations. The feedfoward networks, both the PINNs itself as the internal feedfoward networks of PINNsFormer, have 125 nodes in the hidden layer. For PINNsFormer pseudo-sequence generator, we have choosed the same values used by [3], these begin $k = 5$ steps ahead and step size equals to $1e - 4$. Emulating the experiment set of [3], all loss weights were set to 1 ($\lambda_{res} = \lambda_{ic} = \lambda_{bc}$ = 1) for the sake of simplicity. Seed values were setted at 0 (zero) for experiment reproducbility.

V. Results

In this section, we compare the numerical results obtained from both architectures against the benchmark values provided by [10].

A. Velocities and Pressure

Figures 5 and 8 show velocity u (horizontal velocity) for, respectively, PINNs and PINNsFormer. Both models had no trouble capturing the horizontal velocity concentrated at the top of the cavity as expected, but failed to capture the negative values at the center of the cavity. Check figures 11 and 12.

Figures 6 and 9 show velocity v (vertical velocity), for, respectively, PINNs and PINNsFormer. Both heatmaps display the expected vortexes in opposite directions, but the vortexes at 13 are smoother. These erratic vortexes match the values found at figures 13 and 14.

Both networks seem to satisfy the Dirichlet boundary condition for pressure p as indicated by figures 7 and 10. However, vanilla PINNs was able to better capture the expected pressure concentrations.

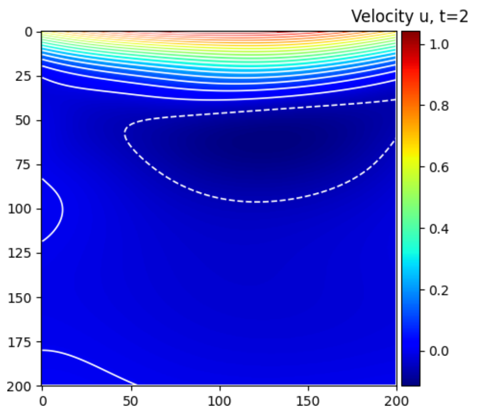

Fig. 5. Velocity u - PINNs.

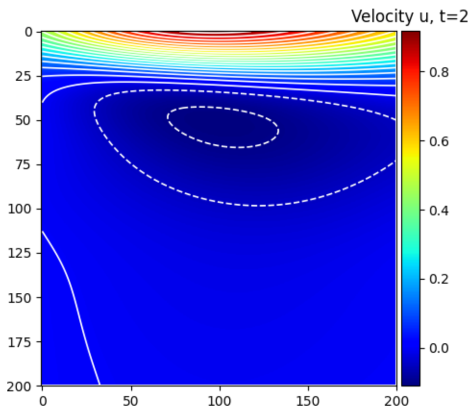

Fig. 8. Velocity u - PINNsFormer.

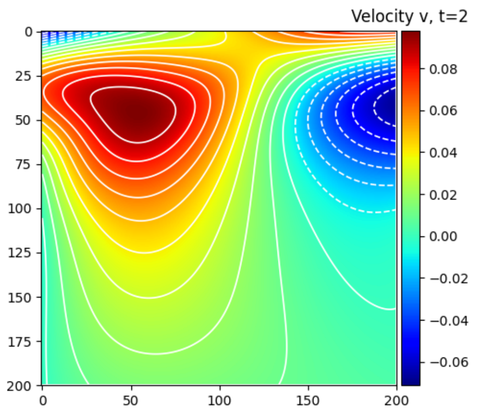

Fig. 6. Velocity v - PINNs.

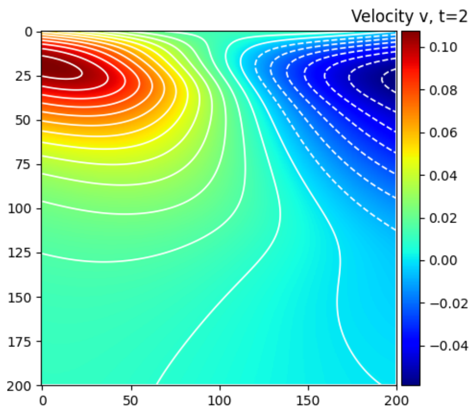

Fig. 9. Velocity v - PINNsFormer.

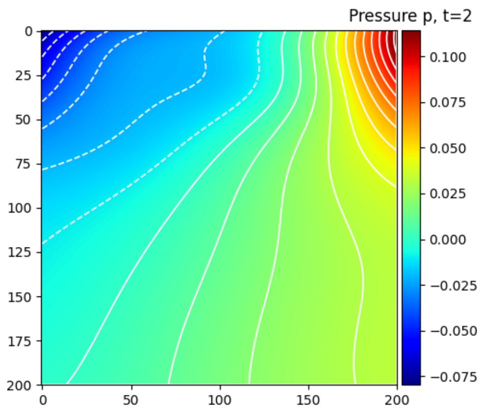

Fig. 7. Pressure p - PINNs.

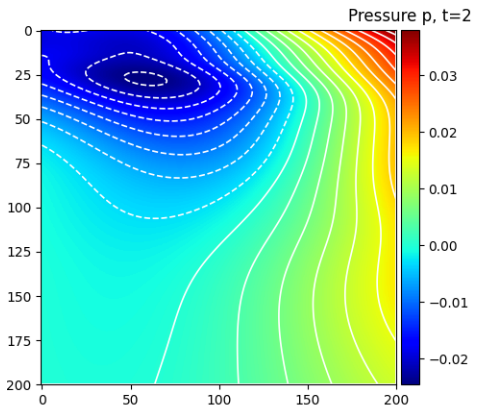

Fig. 10. Pressure p - PINNsFormer.

B. Reference solution

Figures 11 and 12 show the predicted values of both models against the reference values provided by [10] for horizontal velocity u alongside axis y. Both models failed to capture the negative values in the center of the cavity and have an abrupt slope, failing to capture the smooth change in velocity u.

Figures 13 and 14 display the predicted values for vertical velocity v. Both models were unable to capture the slow change in absolute value and sudden direction change of velocity v alongside axis x.

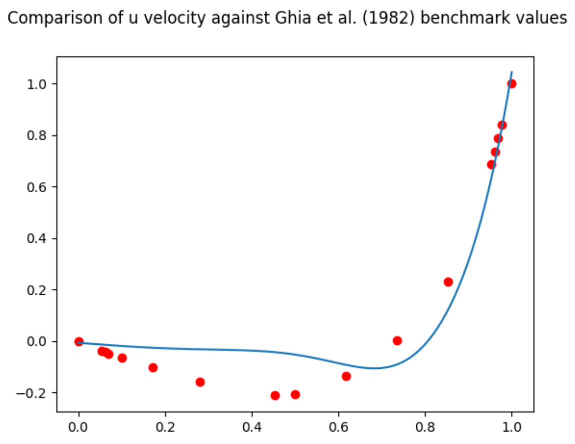

Fig. 11. Velocity u benchmark - PINNs.

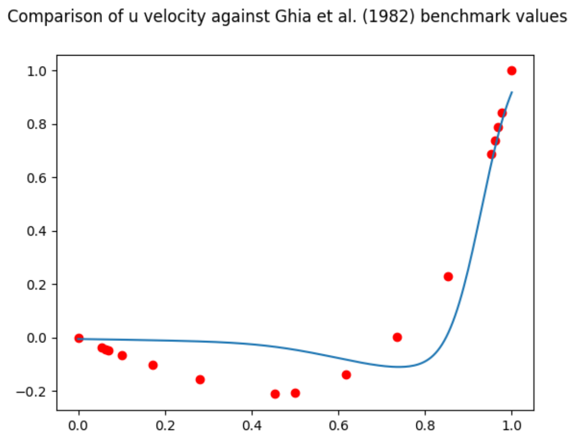

Fig. 12. Velocity u benchmark - PINNsFormer.

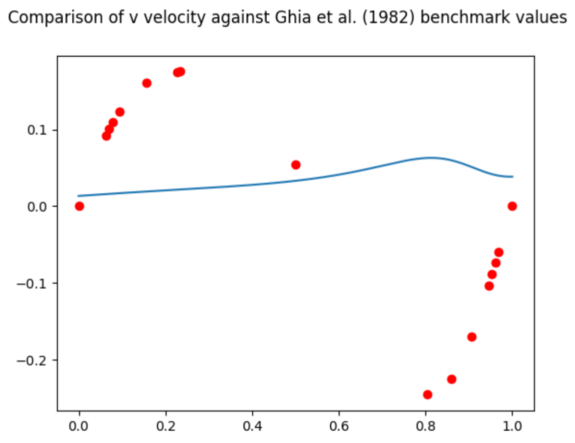

Fig. 13. Velocity v benchmark - PINNs.

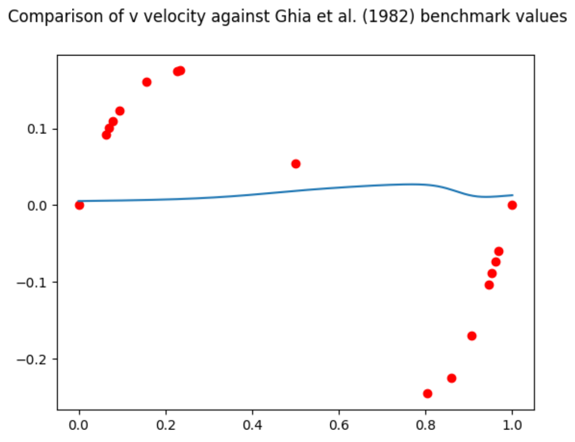

Fig. 14. Velocity v benchmark - PINNsFormer.

C. Loss Analysis

Losses across epochs are displayed at figures 15 and 16. Figures 17 and 18 give a zoom at the last epochs. Figures 19 and 20 provide a variation landscape of both models losses.

PINNsFormer suffered from peaks of extremely high loss values, followed by drastic drops. This erratic and unexpected behavior made impossible to verify the claims made by [3] regarding PINNsFormer smoother and faster convergence, compensating the more costly training. Table I lists the final loss value for both models and PINNsFormer ended up having a considerably higher final loss value than fully-connected PINNs.

TABLE I
FINAL LOSS VALUE

Model	**Loss**
PINNs	0.2353239506483078
PINNsFormer	0.33705413341522217

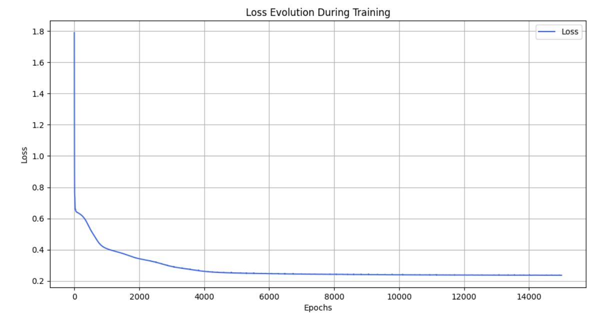

Fig. 15. Loss - PINNs.

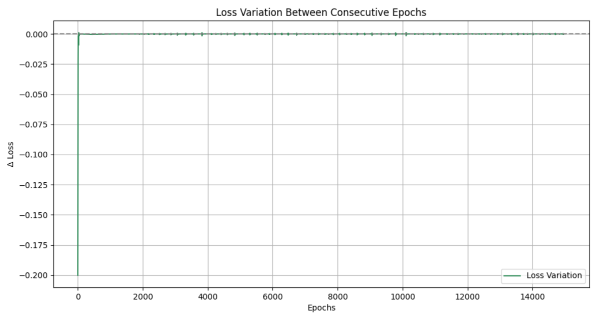

Fig. 19. Loss variation - PINNs.

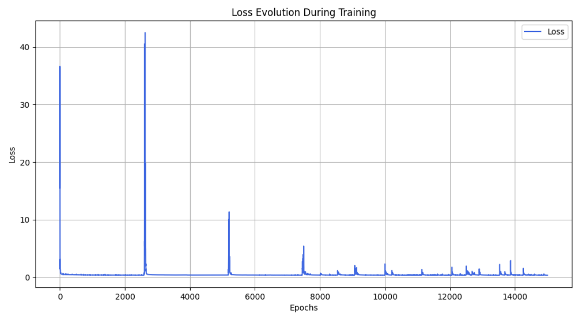

Fig. 16. Loss - PINNsFormer.

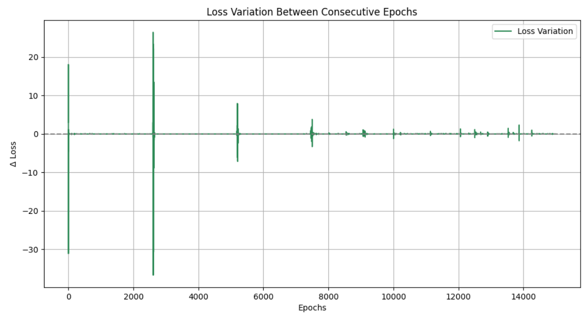

Fig. 20. Loss variation - PINNsFormer.

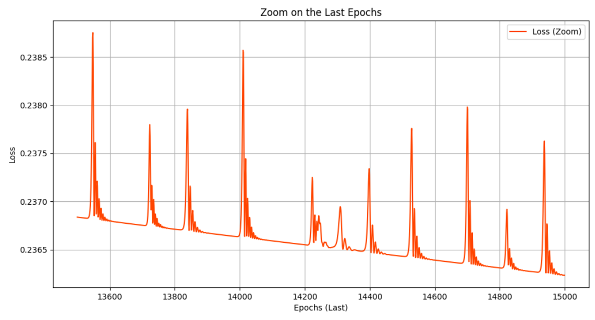

Fig. 17. Last epochs - PINNs.

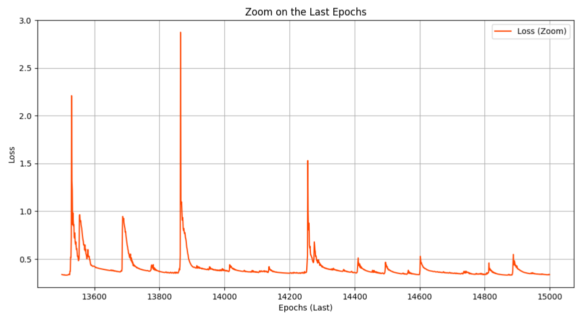

Fig. 18. Last epochs - PINNsFormer.

VI. Conclusions

While Transformers have demonstrated outstanding capabilities in handling sequential data, and PINNsFormer has successfully incorporated many of these ideas into physics-informed neural networks (PINNs) for solving differential equations, our experiments were unable to conclusively verify the advantages of PINNsFormer over vanilla PINNs. This suggests that a more careful examination of the experimental settings—including parameter values, implementation details, and the specific nature of the problems being solved—is necessary. Further investigation is required to fully understand the potential benefits and limitations of PINNsFormer, and to determine the conditions under which it may outperform traditional PINNs.

References

[1] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations,” 2017.

[3] Z. Zhao, X. Ding, and B. A. Prakash, “Pinnsformer: A transformer-based framework for physics-informed neural networks,” 2024.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023.

[5] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, “Deepxde: A deep learning library for solving differential equations,” *SIAM Review*, vol. 63, p. 208–228, Jan. 2021.

[6] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations,” 2017.

[7] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," 2017.
[8] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017.
[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.
[10] U. Ghia, K. Ghia, and C. Shin, "High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method," *Journal of Computational Physics*, vol. 48, no. 3, pp. 387–411, 1982.