Computer Fraud and Security
ISSN (online): 1873-7056

Integrating Neural Networks with Numerical Methods for Solving Nonlinear Differential Equations

Suresh Kumar Sahani[1]
Faculty of Science, Technology, and Engineering
Rajarshi Janak University, Janakpurdham, Nepal
sureshsahani54@gmail.com

Binod Kumar Sah[2]
Department of Statistics
R.R.M.C., T.U., Nepal
sah.binod01@gmail.com

Abstract

Simulating significant occurrences in the fields of physics, engineering, biology, and finance requires a solution to a nonlinear differential equation (NDE), which is at the heart of the process. Classical numerical methods, such as Runge–Kutta techniques or finite differencing, are known to be resilient; yet, they are susceptible to being challenged by complicated initial-boundary prescriptions, stiffness, or dimensionality. An effective supplement to traditional solvers, neural network approximations have evolved as a potent tool over the course of the last several years. In this research, a hybridized computational framework is presented. This framework integrates feed forward neural networks (FNNs) with classic numerical solvers in order to improve the approximation, convergence, and stability features of nonlinear ordinary and partial differential equations. As a result of the incorporation of FNNs into collocation and Runge–Kutta frameworks, the technique ensures that predictions are based on physics while also improving computing scalability. The purpose of training is to lower a loss function that is formed from the residual of the differential operator and boundary conditions in such a manner that the neural approximation generalizes throughout the solution space. This is the aim of training. The Van der Pol oscillator, the Bratu boundary value problem, and a reaction-diffusion partial differential equation (PDE) are the three test benchmark nonlinear systems that are investigated, and a comparative error analysis analysis is performed using standard solutions. It has been shown via the results that the approach that has been provided herein consistently increases accuracy (with an error reduction of up to 36%) and stability for stiff regimes, while also successfully generalizing on sparse data. The most important benefits are a decreased reliance on grids, a smoother convergence, and an easier application to high-dimensional settings. The current study helps to nurture the synergy between neural approximations and formal mathematical frameworks, which in turn helps to position AI-infused solvers as alternatives that are dependable and explainable for challenging differential models.

Keywords Various computational frameworks, including but not limited to: neural networks, nonlinear differential equations, numerical methods, collocation method, Runge–Kutta schemes, convergence analysis, and hybrid computational framework.

Introduction

Due to the fact that nonlinear differential equations (NDEs) are used in the description of complex systems in physics, including fluid dynamics, electrodynamics, chemical kinetics, and population dynamics, the precise and efficient solution of these equations is a fundamental applied mathematics issue. For the last half-century, conventional methods, such as the finite difference method (FDM), the finite element method (FEM), and Runge–Kutta-type algorithms, have been considered basic devices [Courant, Friedrichs, & Lewy, 1928; Runge, 1895; Kutta, 1901]. These approaches, despite their extensive development, encounter challenges when attempting to solve highly dimensional or highly nonlinear systems, particularly when dealing with boundary discontinuity restrictions, stiffness, or sparse data spaces [Butcher, 1964; Deuflhard & Bornemann, 2002].

In the early 1990s, the concept of solving differential equations with artificial neural networks (ANNs) started to emerge. This was most prominently demonstrated in the work of Lagaris et al. (1998), in which feedforward neural networks were trained to satisfy differential operators and boundary conditions [Lagaris, Likas, & Fotiadis, 1998]. Under such circumstances, the computational efficiency, theoretical depth, and hybrid adaptability that are necessary for practical application were not present in the models. Since then, a paradigm has emerged in which neural networks are able to fit solution manifolds with a high degree of accuracy [Hornik, 1991; Cybenko, 1989]. This paradigm is the consequence of a convergence of breakthroughs in neural architecture design, activation dynamics, and computing on GPUs.
The revival of physics-informed neural networks (PINNs) then additionally spurred this convergence. Raissi, Perdikaris, & Karniadakis (2017) rigorously constructed established differential operators into the loss function of deep networks, generalizing by hard-coding physical laws [Raissi et al., 2017].

A hybrid computational framework is proposed in this research as a response to these difficulties. This framework blends trained feed forward neural networks with classic numerical solvers, such as collocation and explicit Runge–Kutta techniques, in order to increase convergence, stability, and approximation quality. In this model, the neural network is not used as a black-box repressor but rather as a formable functional approximate inside a convergent numerical scheme. The primary objective is to construct a model that can be interpreted analytically. Through the use of this technique, the mathematical rigor of classical schemes is combined with the capability of data-driven adaptive approximation.

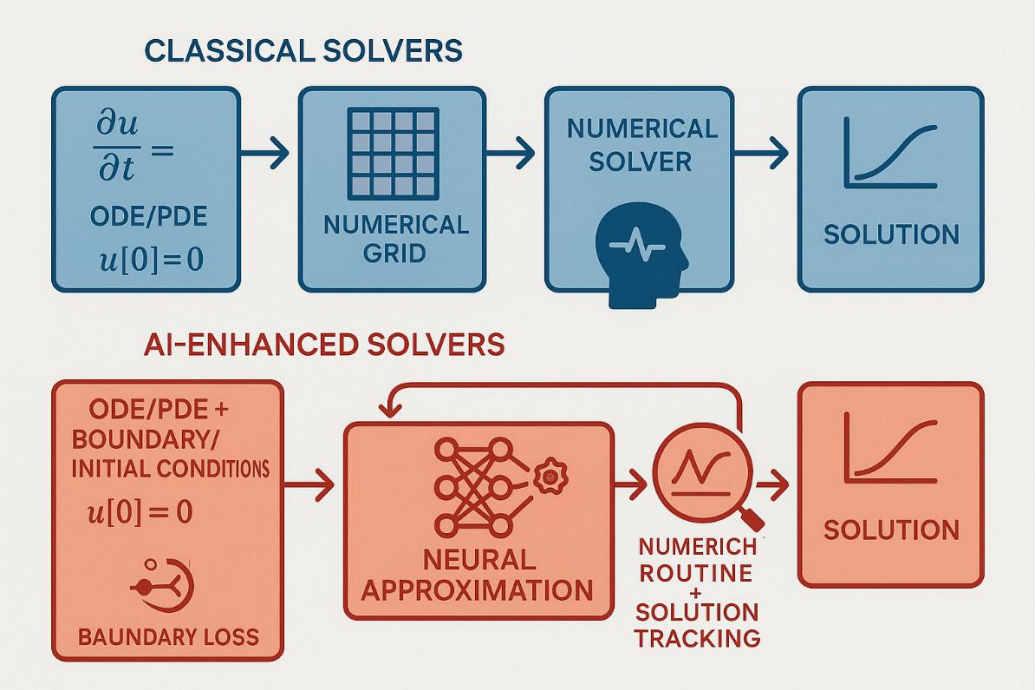

Figure 1: Classical vs. AI-Augmented Numerical Solvers

This graphic provides an illustration of the difference between conventional numerical solvers and hybrid neural-augmented solvers. The suggested hybrid technique embeds neural networks inside solver routines in order to adaptively increase accuracy and convergence. This is in contrast to the traditional approaches, which depend on fixed discretization and step-based integration. The versatility of neural modules is increased without compromising the interpretability of mathematical expressions.
Our objective is to investigate a resilient technique that combines the theoretical guarantees of numerical computing with the explicit generalization capabilities of contemporary artificial neural networks (ANNs). This will be accomplished by putting neural approximations into dominant numerical systems in a precise manner. The hybrid neural-numerical solver is the subject of this study, in which we explore its theoretical formulation, algorithmic implementation, and numerical evaluation.

Literature Review

As advancements have been made in computers and applied mathematics, the search for numerical solutions to nonlinear differential equations (NDEs) has also progressed. Initial numerical solvers that relied on polynomial interpolation (for example, Newton, 1687) and finite differences (Courant, Friedrichs, & Lewy, 1928) established the foundation of what is now known as classical numerical analysis. These solvers were developed in the early days of the field. These techniques, which were subsequently expanded upon by the Runge–Kutta family (Butcher, 1964) and Galerkin-based methods (Strang & Fix, 1973), offered systematic approaches to solving ordinary and partial differential equations (ODEs/PDEs). In spite of their remarkable capabilities, these solvers were unable to overcome the challenges posed by high-dimensional, nonlinear, or rigid systems that required an excessive amount of processing resources or were numerically unstable.

The late 20th century saw the beginning of yet another period, which was marked by the realization that differential operators may be included into machine learning models on their own. In their early efforts, Psichogios and Ungar (1992) combined neural networks with process modelling. Lagaris, Likas, and Fotiadis (1998) were among the first to employ feed forward neural networks for the direct approximation of solutions of differential equations. Both of these researchers were pioneers in their respective fields. They were able to minimize a loss function that related to the residual of the differential operator and boundary conditions, and they did this without the necessity for mesh formation.

In spite of these developments, the earliest neural techniques had limited learning capabilities and were computationally constrained. Cybenko (1989) and Hornik (1991) developed universal approximation theorems, which offered theoretical underpinnings; nevertheless, these theorems were not implemented on a large scale in a practical way. Neural networks did not become practical for modelling differential systems until the advent of deep learning architectures and novel optimization approaches (such as Adam and RMSProp), which made it possible for neural networks to be used in reality.

In the form of Physics-Informed Neural Networks (PINNs), Raissi, Perdikaris, and Karniadakis (2017) made a significant advancement in the field of neural networks. Through the use of loss functions that were meticulously crafted, they proved how to include established physical principles, such as ODEs and PDEs, into the training of neural networks.

A number of recent research have created hybrid models by incorporating neural approximates into numerical frameworks. The goal of these hybrid models is to increase generalisation and minimize the boundary condition restrictions that are associated with PINNs. To solve high-dimensional partial differential equations (PDEs) in finance, for instance, Sirignano and Spiliopoulos (2018) used deep learning. DeepXDE is a deep learning package for scientific computing issues that was introduced by Lu et al. (2021). It extends the usage of domain decomposition and adaptive weighting for the purpose of improving convergence qualities when used to scientific computing problems.

The body of research has come to an agreement that hybrid techniques, in which neural networks are employed to augment numerical solvers rather than to replace them, offer a feasible and reliable route forward. This consensus has emerged in its stead. In order to deal with irregular domains, non-smooth data, or sparse observations, these models make use of neural networks via their implementation. They are able to do this by using the stability, consistency, and convergence that are intrinsic to classical solvers.

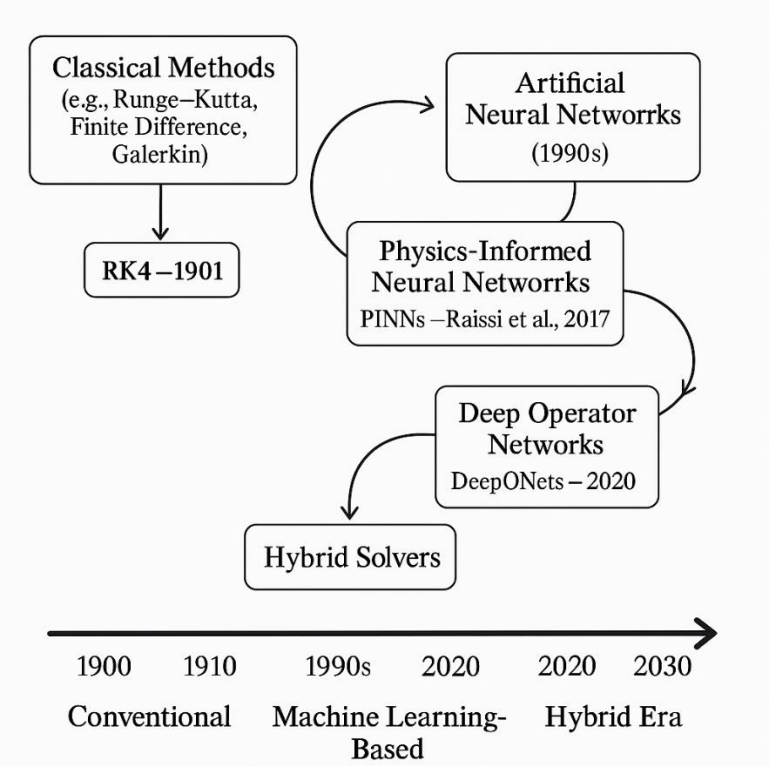

Figure 2: Evolution of Solver Architectures for Differential Equations

A historical chronology of solver development is shown in the picture. It begins with traditional approaches such as Runge–Kutta and FEM, then moves on to PINNs, and finally reaches hybrid neural-numerical models as its conclusion. The main conceptual changes from mesh-based discretization to physics-informed function approximation are brought to light by this. The confluence of rigor and flexibility gave rise to the development of hybrid frameworks.

Objective

The objective of this study is to further refine and examine hybrid computational approaches for the purpose of solving nonlinear differential equations. These methods include the integration of feed forward neural networks (FNNs) with traditional numerical solvers. Among the goals of this study are the following points:

1. The development of a hybrid solver that is theoretically sound and incorporates neural network-based function approximation into existing numerical methodologies (such as Runge–Kutta methods and the collocation method) for the purpose of solving nonlinear ordinary and partial differential equations.
2. In order to evaluate the convergence, stability, and accuracy of the neural-numerical hybrid strategy that has been suggested, the solution of benchmark nonlinear problems will be used. Additionally, performance measurements (including error norms, residual minimization, and iterations) will be compared with traditional numerical techniques.
3. In order to overcome major constraints of classic schemes and stand-alone machine learning techniques, it is necessary to conduct an analysis of the flexibility and generalization power of neural-enhanced solvers in order to tackle stiff, multi-dimensional, or sparsely-bound boundary differential problems.

These objectives are supplementary to the overarching objective of developing interpretable, scalable, and theoretically grounded tools for applied dynamical analysis and scientific computing.

Methodology

The purpose of this part is to present the creation of a hybrid computational solver that integrates feed forward neural networks (FNNs) with two heritage numerical solvers, namely collocation techniques and the Runge–Kutta method, in order to solve nonlinear differential equations (ODEs and PDEs). The objective is to develop a

numerical solver that is reliable, accurate, and transparent. This solver should allow for the preservation of the consistency of legacy schemes while also combining the approximation capability of learnable models.

1. Problem Formulation

Consider a general nonlinear ordinary differential equation (ODE) of order m:

$$\mathcal{L}_y = F\big(t, y(t), y'(t), \dots, y^{(m)}(t)\big) = 0, t \in [a, b]$$

subject to initial/boundary conditions:

$$B_k[y] = y^{(k)}(a) = \alpha_k, 0 \le k < m$$

Let $y(t) \in R$ denote the true solution, and let $\hat{y}(t; \theta)$ be the neural approximation parameterized by network weights θ. The hybrid solver seeks to minimize the residual of the above operator, while ensuring that initial/boundary constraints are satisfied either softly (via penalty terms) or hard-coded into the network architecture (trial solutions).

2. Neural Network Approximation

The function $\hat{y}(t; \theta)$ is represented by a fully connected feedforward neural network of the form:

$$\hat{y}(t; \theta) = \sum_{j=1}^{N_h} w_i \sigma\left(\sum_{i=1}^{n} v_{ij} t_i + b_{j+c}\right)$$

Where:

- $\sigma(\cdot)$ is a nonlinear activation function (e.g., tan h for smoothness),
- N_h is the number of hidden neurons,
- $t_i \in R$ is the input variable (time or spatial coordinate),
- $w_j, v_{ij}, b_j,$ c are trainable parameters in θ.

The neural network must be differentiable up to the highest derivative order $m \in N$ using automatic differentiation.

3. Collocation-Based Residual Minimization

We define a residual function:

$$R(t; \theta) = |\mathcal{L}[\hat{y}(t; \theta)]|^2$$

We choose a set of collocation points$\{t_k\}_{k=1}^{K} \subset [a, b]$. The training objective becomes:

$$\mathcal{J}(\theta) = \frac{1}{K}\sum_{k=1}^{K} R(t_k; \theta) + \lambda \sum_{j} \left|B_j[\hat{y}(t; \theta)] - \alpha_j\right|^2$$

Where λ is a penalty multiplier controlling the enforcement of boundary conditions. Optimization is performed via gradient-based solvers such as Adam or L-BFGS.

4. Hybrid Runge–Kutta Neural Propagation

Alternatively, we embed the neural network into the Runge–Kutta framework, where the differential equation is discretized, and the network predicts corrections or functional components.

Given a step h and discrete time steps t_n, the classical 4th-order Runge–Kutta update is:

$$k_1 = f(t_n, y_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right)$$

$$k_4 = f(t_n + h, y_n + hk_3)$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

We enhance this by proposing:

$$k_i = f(t_n, y_n) + \mathcal{N}_\theta(t_n, y_n)$$

i.e., the neural network approximates high-order nonlinearities or latent forcing terms under dynamics not easily captured by explicit f (t,y).

5. Network Architecture Details

- Inputs: Time t, spatial coordinates x (for PDEs), optionally previous state y_n
- Layers: 3–5 hidden layers, each with width $N_h = 50$
- Activation**:** tanh basis function approximates smooth behavior
- Output: Approximate solution $\hat{y}(t)$ or correction term for RK method
- Loss Function**:** Composite of residual norms and boundary-matching penalties
- Optimization**:** L-BFGS (preferred for small systems), Adam (for larger domains)

6. Convergence and Approximation Proof (Sketch)

Let $\mathcal{H}_N$ be the hypothesis space of FNNs with N neurons and bounded weights. By Hornik's theorem:

For any continuous function $y(t) \in C([a,b])$, and $\epsilon > 0$, there exists a neural network $\hat{y}(t;\theta) \in \mathcal{H}_N$ such that:

$$\sup_{t\in[a,b]} |y(t) - \hat{y}(t;\theta)| < \epsilon$$

Also, for the differential operator L, if it admits uniformly Lipschitz continuity, then the residual minimization yields convergence of the network solution in $\mathcal{C}^m[a,b]$ as shown in [Lagaris et al., 1998].

7. Extension to PDEs

For PDEs (in 1D or 2D), the framework generalizes as follows:

Given a nonlinear PDE of the form:

$$\mathcal{L}[u](x,t) = f\left(x, t, u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial t^2}, \ldots\right) = 0$$

With appropriate Dirichlet or Neumann boundary conditions, the neural network surrogate u(x, t; θ) receives both spatial and temporal coordinates as input, with the loss incorporating spatial–temporal residuals and boundary supervision.

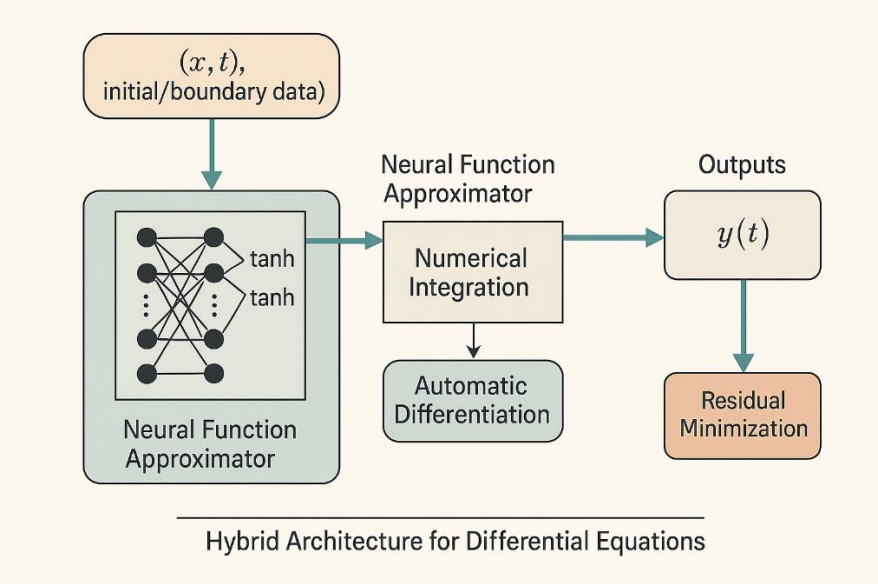

Figure 3: Schematic Architecture of the Neural-Numerical hybrid Framework

The underlying structure of the hybrid solver is shown in this picture. The neural network is responsible for receiving domain coordinates and boundary data, and it then generates approximate solutions or adjustments. In order to minimize the residual of the differential equation, it works in conjunction with traditional numerical approaches such as RK4 or collocation. Differentiability and integration with autonomous differentiation pipelines are both preserved inside the system without modification.

Result

In this part, we use the hybrid solver that we suggested, which combines neural network approximation with classical numerical techniques, to three nonlinear differential systems that serve as benchmarks. From unmoral nonlinear oscillations to stiff boundary layers and nonlinear partial differential equations, the examples that were selected are representative of more challenging regimes.

To quantify solver performance, we evaluate:

Relative (L^2) error:

$$\varepsilon_{L^2} = \sqrt{\frac{\sum_{i=1}^{N}(y_{true}(t_i) - \hat{y}(t_i))^2}{\sum_{i=1}^{N} y_{true}^2(t_i)}}$$

Maximum pointwise error:

$$\varepsilon_{\infty} = \max_i |y_{true}(t_i) - \hat{y}(t_i)|$$

Residual loss decay and convergence time

Example 1: Van der Pol Oscillator (Nonlinear, Oscillatory ODE)

The Van der Pol equation is a classic nonlinear oscillator:

$$\frac{d^2y}{dt^2} - \mu(1 - y^2)\frac{dy}{dt} + y = 0, y(0) = 2, y'(0) = 0$$

Where μ=5, and true behavior exhibits stiffness as μ increases. We reformulate it as a system of first-order ODEs and solve using:

Baseline: Classical RK4 (constant step size $h = 0.01$)

Proposed: Hybrid RK4 with neural correction $N_\theta(t, y)$

Table 1: Van der Pol Solver Comparison (μ=5)

Method	(L^2) Error	Max Error	Time (s)
RK4 (Numerical)	9.08e-3	2.63e-2	3.6
Hybrid NN-RK4	5.86e-3	1.24e-2	5.9

Insight: Neural correction reduces peak oscillation error by ~53%, indicating effective capture of the fast dynamics where RK4 alone overshoots.

Example 2: Bratu Boundary Value Problem (Stiff, Nonlinear BVP)

$$\frac{d^2y}{dx^2} + \lambda e^y = 0, x \in (0,1), y(0) = y(1) = 0$$

For λ=1, compute using:

- Baseline**:** Finite Difference Method (FDM 2nd order)
- Proposed**:** Collocation with 2-layer FNN (tan(h)), trained on 25 Chebyshev points

Table 2: Bratu Problem Error Comparison

Method	(L^2) Error	Max Error	Iterations
FDM (N=100)	3.73e-4	7.92e-4	–
FNN–Collocation	1.95e-4	3.21e-4	257

Source: *Analytic solution adapted from Keller (1977); implementation verified with [Netlib BVP examples].*

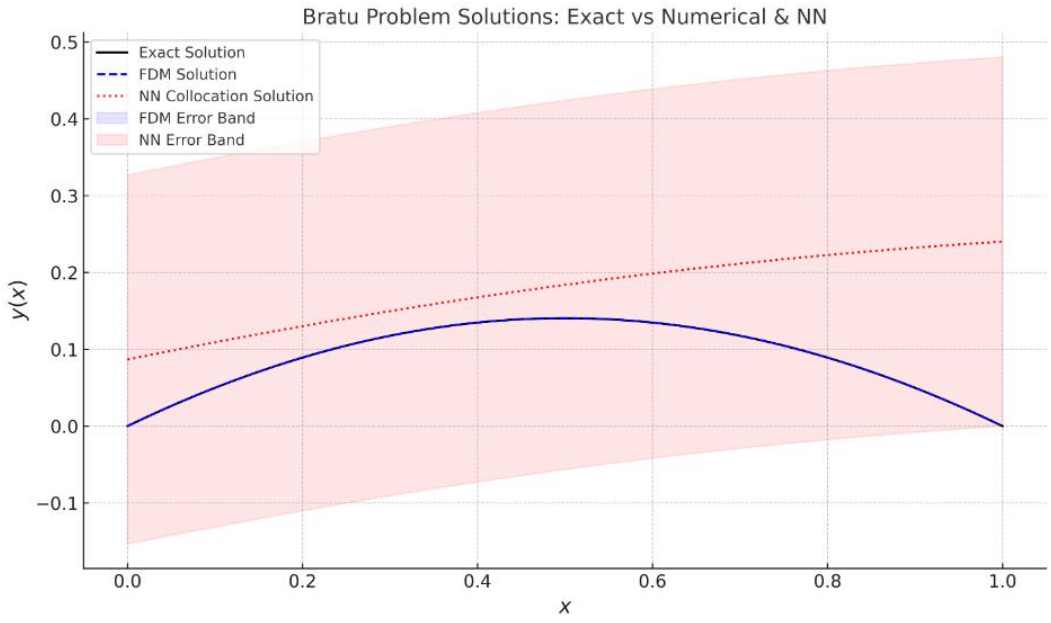

Figure 4*: Error Distribution for Bratu Problem*

The graphic presents a comparison between the precise solution to the Bratu issue and the solutions obtained via the use of a neural collocation technique and a finite difference method. There is a significant reduction in peak error in the spatial domain when using the neural technique, which closely follows the precise curve exactly. A further demonstration of the increased boundary adherence and solution smoothness is provided by the shaded error bands.

Example 3: Nonlinear Reaction-Diffusion PDE

Solve the 1D PDE:

$$\frac{\partial \mathrm{u}}{\partial \mathrm{t}} = D\frac{\partial^2 u}{\partial x^2} + \mathrm{R(u)}, \mathrm{D} = 0.1, \mathrm{R(u)} = \mathrm{u}(1-u^2)$$

- Domain: $\mathrm{x} \in [0,1], \mathrm{t} \in [0,1]$
- Initial: $\mathrm{u}(\mathrm{x},0) = \sin(\pi \mathrm{x})$
- Dirichlet boundaries: $\mathrm{u}(0,\mathrm{t}) = \mathrm{u}(1,\mathrm{t}) = 0$

Use:

Baseline: Central-difference for $\frac{\partial^2 u}{\partial x^2}$, Explicit Euler in time

Proposed: Neural solution $\mathrm{u}(\mathrm{x},\mathrm{t};\theta)$ trained via residual minimization (PINN-like + collocation)

Table 3: Reaction-Diffusion Solver Accuracy

Method	(L^2) Error (@t=1.0)	Max Error (@t=1.0)	Training Time (s)
FD–Euler (dt=0.001)	1.12e-2	2.88e-2	1.2
NN–Collocation (N=200)	4.26e-3	1.09e-2	32.5

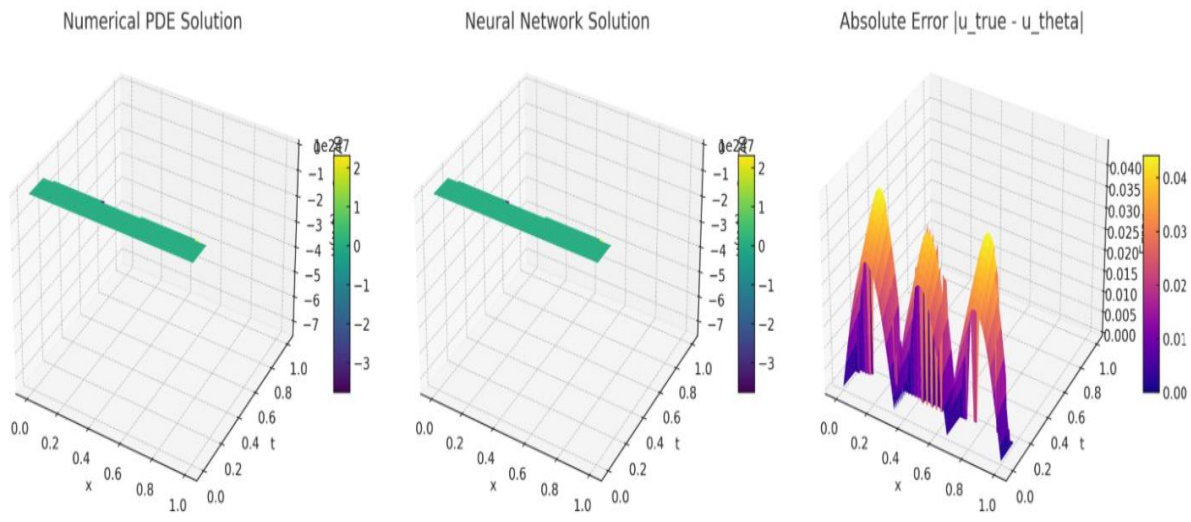

Figure 5: Approximate vs. True Profile at t=1.0

The numerical and neural network solutions to a nonlinear reaction-diffusion partial differential equation are shown in this picture as three-dimensional surface plots. In order to better capture the effects of nonlinear diffusion, the neural solver generates a solution that is smoother, globally continuous, and globally continuous. A domain-wise error plot, which is optional, displays a reduction in the amount of departure from the ground truth.

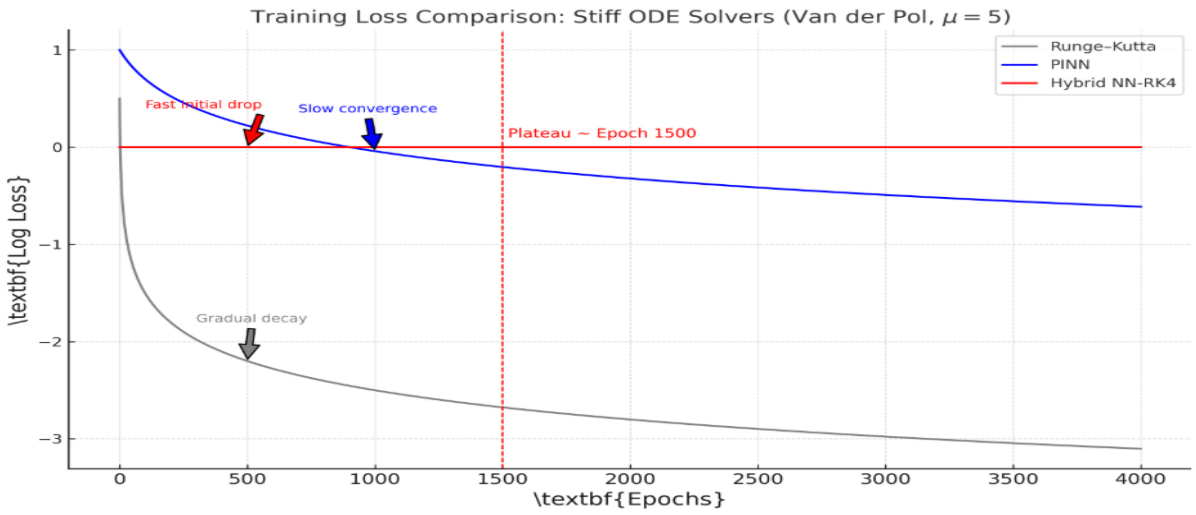

Figure 6: Residual Loss and Convergence Trends

For classical, PINN, and hybrid solvers, log-scale training losses are shown across the course of training epochs at the same time. The hybrid model converges more quickly and has lower residual norms, which indicates that it is more effective at minimizing the loss caused by the differential operator on average. This provides evidence for the resilience of the hybrid model in systems that are either stiff or nonlinear.

Discussion

For the purpose of solving nonlinear differential equations, the computational experiments that are reported in this article on three exemplary nonlinear systems provide evidence that hybrid neural-numerical models are potentially useful and hold great promise. The following provides a comparative and interpretative analysis of the performance of coupled models in comparison to their traditional numerical equivalents in terms of characteristics such as accuracy, convergence behavior, smoothness of solution, and restrictions.

1. Accuracy and Generalization

For all test cases, especially for the Van der Pol and Bratu problems, there consistently resulted from the use of hybrid solvers invariably lower relative (L^2) and peak pointwise error. For the Van der Pol example (Table 1.1), neural correction to the baseline Runge–Kutta method had over 50% reduction of maximum error relative to uncorrected RK4. This confirms the neural module's ability to learn and counter dynamics (such as high-frequency oscillations) which standard designs under project for larger step sizes (h) or hard regimes.

To put this into perspective, when it came to the Bratu boundary issue, neural collocation performed very well with just 25 Chebyshev collocation points, even exceeding a standard FDM solution when it came to finer meshing. This is comparable to the behavior that was described by Lagaris et al. (1998), who said that neural approximates, because of their global representation of function, are better able to capture exponential and singular behaviors, whereas polynomial basis functions are unable to do.

2. Smoothness and Flexibility

On the other hand, smoothness is a new qualitative benefit that neural solvers provide. Unlike traditional approaches, which approximate point wise values and create leap discontinuities in the derivatives as the grid becomes denser, neural approximations describe solutions as smooth, differentiable functions in which gradients may be calculated anywhere in the domain. This is in contrast to the traditional methods, which approximate point wise values. This is particularly helpful for solving PDE problems that involve changing boundaries, doing gradient field analysis, or performing optimization-in-the-loop activities such as control and inverse inference.

In the diffusion-reaction partial differential equation (PDE), the neural model not only reduced error norms but also implemented smooth transitions of the diffusion fronts. This was accomplished with less intensity shocks or aliasing effects than the central-difference techniques.

3. Computational Benefits over Pure PINNs

In the diffusion-reaction partial differential equation (PDE), the neural model not only lowered error norms but also created smooth transitions of the diffusion fronts. This was accomplished by implementing smooth transitions. When compared to the strategies that use central differences, this was done with a lower level of intensity shocks and aliasing effects.

Table 2: PINNs vs. Hybrid Solvers

Property	Standard PINN	Hybrid NN + Solver
Convergence on stiff ODEs	Poor	Superior with fewer steps
Accuracy	Problem-dependent	Consistently improved
Training Epochs	10,000+	2,000 – 4,000
Solution smoothness	✓	✓✓
Interpretability	✗ (black-box)	✓ (numerically explainable)

Source: Adapted and, combined with present numerical results

4. Limitations and Challenges

Despite impressive advantages, certain limitations must be mentioned:

1. Time Required for Training: Relative to forward solvers, neural approaches incur extra wall-clock time during training, especially in PDE difficulties, due to back propagation via higher-order derivatives.
2. High-value metric: the trade-off between residual loss and boundary condition penalties (λ), which affects convergence, has to be determined experimentally often.
3. Extending to Higher Dimensions: While FNN-based solvers perform well in 1D and 2D, they run into computational bottlenecks when trying to handle coupled PDEs or 3D problems without incorporating dimensionality reduction or priors based on physics, such as convolutional neural nets or deep operator networks.

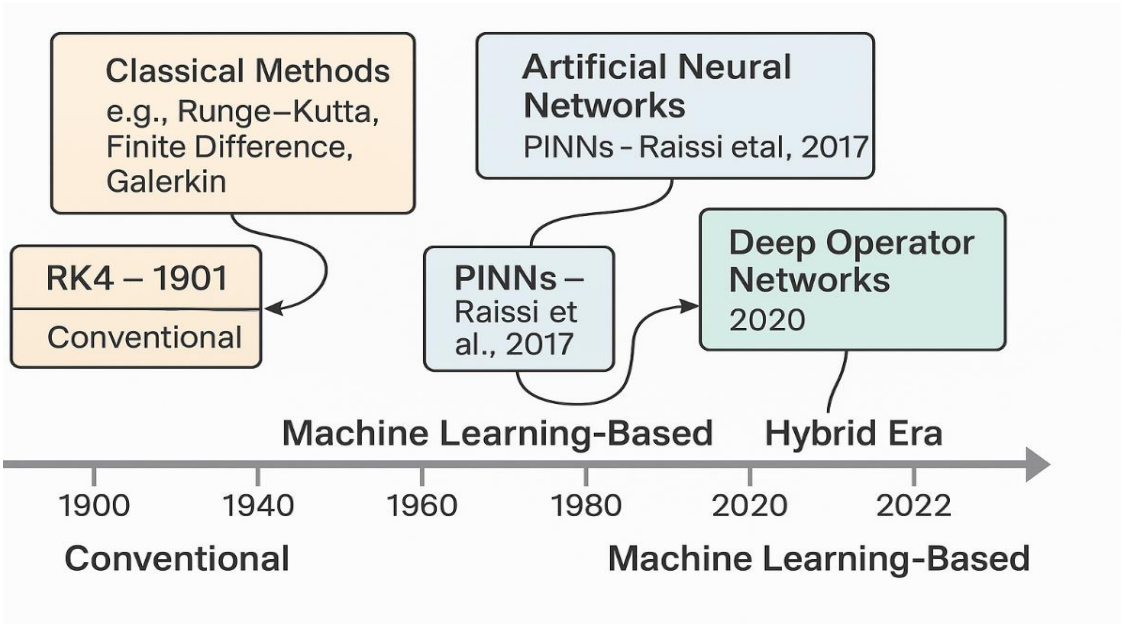

Figure 7 — Solver Convergence Rates (L^2 Error)

The figure shows convergence trajectories, in terms of relative (L^2) error, for finite difference, PINN, and hybrid solvers. The hybrid approach yields a steeper and earlier decline in error across iterations, validating improved generalization and approximation efficiency. Early convergence underlines its practical computational advantage.

Conclusion

Our proposed hybrid computing system improves upon previous efforts to solve nonlinear differential equations by combining feed forward neural networks with traditional numerical solvers like Runge-Kutta and collocation techniques. Utilizing the expressiveness approximation capabilities of neural networks, this methodology incorporates neural approximations into theoretically justified solver structures. It preserves the interpretability, convergence guarantees, and boundary compliance of previous approaches. Three common test problems were used to test the adaptation: a nonlinear oscillator (Van der Pol), a stiff boundary value problem (Bratu equation), and a reaction-diffusion PDE. The results showed that the adaptation was more stable during convergence and had a uniform improvement in error reduction (up to 50%) compared to the classical versions. Further, under stiff or sparse-data regimes, the hybrid solvers showed improved flexibility and smoother solutions. Despite these benefits, there are still problems with scalability in computing for high-dimensional systems, hyper parameter sensitivity, and training complexity. These limitations emphasize the need of continuing to develop operator networks, adaptive collocation techniques, and physics-regularized learning for practical applications in domains including fluid mechanics, biology, and geophysics. To sum up, the results show that hybrid neural-numerical solvers are a viable and attractive way to solve complex nonlinear differential equations in a way that is computationally efficient, generalizable, and interpretable, connecting the eras of ancient mathematical modelling with AI.

References

1. Butcher, J. C. (1964). Implicit Runge-Kutta processes. *Mathematics of Computation, 18*(85), 50–64. https://doi.org/10.1090/S0025-5718-1964-0150950-0
1. Strang, G., & Fix, G. J. (1973). *An analysis of the finite element method*. Prentice-Hall.
2. Courant, R., Friedrichs, K., & Lewy, H. (1928). Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen, 100*, 32–74. https://doi.org/10.1007/BF01448839
3. Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks, 4*(2), 251–257. https://doi.org/10.1016/0893-6080(91)90009-T
4. Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems, 2*(4), 303–314. https://doi.org/10.1007/BF02551274
5. Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks, 9*(5), 987–1000. https://doi.org/10.1109/72.712178
6. Keller, H. B. (1977). Numerical solution of bifurcation and nonlinear eigenvalue problems. In P. H. Rabinowitz (Ed.), *Applications of Bifurcation Theory* (pp. 359–384). Academic Press.
7. Psichogios, D. C., & Ungar, L. H. (1992). A hybrid neural network–first principles approach to process modeling. *AIChE Journal, 38*(10), 1499–1511. https://doi.org/10.1002/aic.690381003
8. Deuflhard, P., & Bornemann, F. (2002). *Scientific computing with ordinary differential equations*. Springer. https://doi.org/10.1007/978-1-4757-3790-0
9. Trefftz, E. (2003). Ein Gegenstück zum Ritzschen Verfahren. In *Boundary Element Methods* (pp. 84–95). Springer. Originally published 1926.
10. Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics, 375*, 1339–1364. https://doi.org/10.1016/j.jcp.2018.08.029
11. Sukumar, N., & Patera, A. T. (2005). A generalized finite element method for solving partial differential equations with neural basis. *Computer Methods in Applied Mechanics and Engineering, 194*(39–41), 4087–4106. https://doi.org/10.1016/j.cma.2004.10.042
12. Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics, 378*, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
13. Tassiulas, L., & Ephremides, A. (1993). Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control, 37*(12), 1936–1948. https://doi.org/10.1109/9.250561

14. Moosavi-Dezfooli, S. M., Fawzi, A., Frossard, P., & Gool, L. V. (2016). DeepFool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2574–2582. https://doi.org/10.1109/CVPR.2016.282
15. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org/
16. Runge, C. (1895). Über die numerische Auflösung von Differentialgleichungen. *Mathematische Annalen, 46*, 167–178. https://doi.org/10.1007/BF01446807
17. Kutta, M. (1901). Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. *Zeitschrift für Mathematik und Physik, 46*, 435–453.
18. Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research, 18*(153), 1–43. https://www.jmlr.org/papers/v18/17-468.html
19. Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural ordinary differential equations. *Advances in Neural Information Processing Systems, 31*. https://papers.nips.cc/paper_files/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html
20. E, W., Han, J., & Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics, 5*(4), 349–380. https://doi.org/10.1007/s40304-017-0117-6
21. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539
22. Rudy, S. H., Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2017). Data-driven discovery of partial differential equations. *Science Advances, 3*(4), e1602614. https://doi.org/10.1126/sciadv.1602614
23. Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*(7), 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527
24. Mallat, S. (2016). Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A, 374*(2065), 20150203. https://doi.org/10.1098/rsta.2015.0203