# Facts and issues of neural networks for numerical simulation

Imad Kissami, Christophe Cérin, Fayssal Benkhaldoun, Fahd Kalloubi

# Facts and issues of neural networks for numerical simulation

Imad Kissami[1], Christophe Cérin[2], Fayssal Benkhaldoun[3], and Fahd Kalloubi[1,4]

[1] Mohammed VI Polytechnic University, Morocco,
{imad.kissami,fahd.kalloubi}@um6p.ma
[2] University Sorbonne Paris Nord & INRIA Datamove team, France,
christophe.cerin@univ-paris13.fr
[3] University Sorbonne Paris Nord, France,
fayssal.benkhaldoun@univ-paris13.fr
[4] Université Chouaib Doukkali, LTI Laboratory, Morocco

**Abstract.** Deep learning and artificial intelligence (AI) have transformed computer science, becoming the main method for addressing various problems, from partial differential equations to molecular discovery. Traditional numerical simulation techniques use finite differences to solve equations, but are computationally costly, often taking hours to days on powerful machines. Simulations need to be restarted for shape changes, making the process inefficient. Artificial neural networks now enable accurate simulations quickly and inexpensively. This paper reviews recent developments in neural networks, their advantages, disadvantages, and unresolved issues, often demonstrated through partial differential equations.

## 1 Introduction

With AI's rise, governments tasked public institutions to lead the development of AI tools for research, energy, and security. In France[5] and the United States[6], communities highlighted AI's potential for science in numerous reports. The US report defines surrogate models as "simpler yet faithful" representations of complex systems, trained on outputs from other models. It advises launching pilot programs for developing surrogate models of plasma turbulence and Earth's oceans.

Deep learning for solving PDEs [61, 22, 40, 41] is central in scientific machine learning, using the universal approximation and expressivity of neural networks. Neural networks can approximate any high-dimensional function with enough training data, but they overlook the problem's physical characteristics. Their accuracy relies heavily on specifying problem geometry and initial and boundary conditions. Without this, solutions may lack uniqueness and physical correctness.

[5] https://www.cnrs.fr/fr/le-centre-artificial-intelligence-science-science-artificial-intelligence-aissai
[6] https://www.anl.gov/sites/www/files/2023-06/AI4SESReport-2023-v6.pdf

Neural networks for PDEs use governing equations during training. They are crafted to meet training data and these equations, allowing even sparse and incomplete data to guide their development. These networks can find optimal solutions based on physical constraints, even with limited knowledge of boundary conditions. With some understanding of the problem's physical traits and minimal data, they achieve high-fidelity solutions.

Neural networks for PDEs address diverse computational science problems and offer groundbreaking mesh-free solvers. They provide alternatives to traditional methods like CFD, Electromagnetics, and Quantum Mechanics, or data-driven model inversion techniques. Once trained, they can predict values on grids of varying resolutions without retraining. They utilize automatic differentiation (AD) for superior derivative computation compared to numerical or symbolic methods.

Neural networks for solving PDE are considered a breakthrough in scientific machine learning but face challenges such as unclear convergence, scalability issues in complex domains, and suboptimal neural architectures. Approaches such as Physics-Informed Neural Networks (PINNs) [65, 31], Neural Operators (NO), Transformer Networks (TN), and Newton-Informed Neural Operators (NINO) help overcome these challenges, enhancing efficiency and expanding applicability across various scientific problems.

Advanced methods are not applicable to all problems. Scientific issues require customized approaches based on their unique characteristics. These enhancements are based on neural networks for PDEs, but must be sufficiently assessed for each situation.

This paper aims to explore the scientific and technical issues of using neural networks for PDE-based numerical simulations. Section 2 provides introductory material: a brief history of numerical simulation and neural networks, a comparison of conventional and data-driven methods, and an overview of relevant neural networks such as Physics Informed Neural Networks, along with examples of PDEs like the Advection, Burgers', Shallow Water, and Navier-Stokes Equations.

Section 3 introduces Neural Network architectures for solving PDEs, highlighting challenges such as costly training compared to resolution. Models lack reproducibility due to retraining for different domains or initial conditions. An improved understanding of theoretical convergence is needed for scalability and robustness.

Section 4 addresses performance evaluation, questioning whether to combine machine learning and HPC metrics or treat them separately. We also discuss frugal methods and a System view of the ecosystem. Section 5 presents web-based programming use cases to demonstrate concept practicality, while Section 6 concludes the paper.

## 2 Vocabulary and fundamentals

### 2.1 A brief history of numerical simulation and neural networks

**Neural Networks** Neural networks have a long history, beginning with early attempts to understand how the brain functions and behaves. In 300 BC, Aristotle established the notion of Associationism [8], marking the beginning of humans' efforts to understand the brain. In the 19th century, Alexander Bain introduced Neural Groupings as early neural network models, eventually influencing the Hebbian Learning Rule [64]. Donald Hebb is regarded as the father of neural networks after inventing the Hebbian Learning Rule in 1949, which laid the groundwork for current neural networks. McCulloch and Pitts introduced the MCP Model in 1943 [45], regarded as the progenitor of artificial neural models. In 1958, Frank Rosenblatt introduced the first perceptron [54], resembling modern perceptions. Over the years, neural networks have evolved significantly, leading to the emergence of deep learning models such as Convolutional Neural Networks (CNNs) [38], [21], Deep Belief Networks (DBNs) [26] [28], and Recurrent Neural Networks (RNNs) [47] [46]. These models have revolutionized various fields, including computer vision, natural language processing, and speech recognition. In recent years, many enhancements were proposed to improve neural networks convergence, accuracy, and training speed, such as Batch Normalization [30], Dropout [58], GELU [25], Transformer [62], etc.

**Numerical simulation** Numerical simulation [59] originated from the 18th-century mathematicians Euler and Lagrange, who advanced differential equations and calculus [19] [37]. In the twentieth century, John von Neumann's work with computers introduced algorithms for solving partial differential equations in hydrodynamics and aerodynamics. The finite element method (FEM) emerged, breaking down the structures into smaller elements [18], while the finite difference method (FDM) approximated the derivatives on a grid [53]. The 1970s saw growth in fluid dynamics (CFD) and the introduction of the finite-volume method (FVM) [1], focusing on conserving fluxes for CFD applications [4] [43]. Simulation software made these tools more accessible to all industries [2].

### 2.2 Conventional methods versus data-driven methods

**Conventional methods** The computational modeling of a physical process involves two key stages: (1) developing a mathematical model, and (2) computer simulation. First, a mathematical model is created using the governing equations for various phenomena. For instance, the Navier-Stokes equations [12] describe fluid dynamics, elasticity equations [27] cover solid deformation under forces, and Maxwell's equations [29] explain electromagnetism. Due to their complexity, these equations often require simplifications for analytical or numerical solutions, such as using the Laplace equation for potential flow when the viscosity of the fluid is negligible.

After creating a mathematical model with initial boundaries and conditions, attention is turned to the solution model using numerical methods 2.1. Time-stepping techniques [33] [11] progress the solution in intervals, while convergence and stability [57] ensure precision. Solvers, error analysis, and parallel computing enhance computational efficiency [17] [7], crucial for large-scale simulations.

**Data-Driven methods** Data-driven PDE solutions are a significant advance from traditional methods, incorporating advanced ML algorithms to enhance and accelerate simulations [63] [36] [34]. These models excel at processing large data sets, identifying subtle patterns, and underlying physics that is ignored by numerical methods. A key ML achievement is the detection and correction of discretization errors in PDE simulations, the improvement of accuracy, and the reduction of computational load.

This method improves error correction and accurately predicts complex fluid dynamics, such as turbulence [69] and varying aerodynamic motions [36]. Integrating machine learning into PDE has led to neural network methods, such as deep learning [9] and [50], to better represent fluid flow. This advancement integrates data directly into simulations, bridging theory and reality, enabling real-time adaptation, and significantly improving PDE model predictions.

Data-driven methods excel at handling complex geometries and flow conditions challenging traditional PDE-solving techniques. ML models interpret fluid dynamics effectively, avoiding the simplifications or high costs of physics-based models. Integrating these data-driven models with PDE approaches improves computational fluid dynamics, resulting in faster, more cost-effective, and more accurate simulations. This has significant implications for engineering and environmental research, promising simulations that align closely with real-world dynamics.

### 2.3 Reminders on relevant neural networks for solving PDE

**Introduction** Neural networks have successfully solved PDEs in fields like biology and physics. Function learning techniques, including Physics-Informed Neural Networks (PINNs), aim to approximate PDE solutions by learning the solution function. Despite their success with ill-posed problems, optimizing neural architecture remains challenging. Methods like operator learning address PINN limitations. Figure 1 categorizes the existing neural network architectures for PDE solving, which will be analyzed subsequently.

**Physics Informed Neural Networks** Physics-Informed Neural Networks (PINNs) [55, 52, 68, 44, 32] effectively solve PDE problems by integrating physical laws into learning [44], [32], [52]. The PINN process involves formulating the PDE, solving equations, and interpreting results, with physical laws encoded in the network.

This process involves the architecture and activation functions of the neural network. The network must be sophisticated and deep enough to solve the problem and capture fluid motion effects. After training with the provided data, the

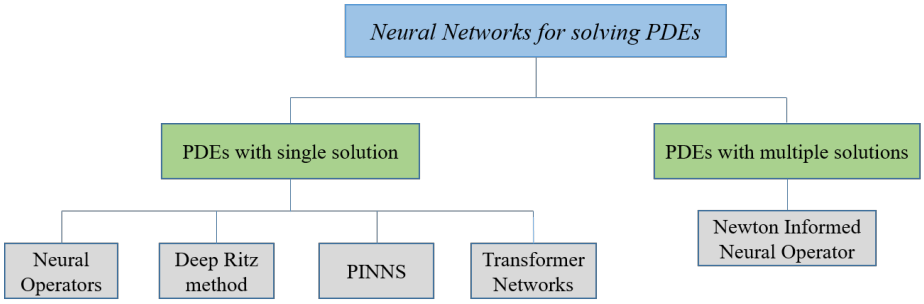


**Fig. 1.** A classification diagram of Neural Network architectures for solving PDE

results should be interpreted to understand the fluid dynamics simulation and ensure its correctness.

PINNs are ideal for solving PDEs due to their ability to mimic physical laws, making them more accurate than other numerical methods. They are especially useful when flow information is scarce or costly, as they use such data with neural network predictions and fluid motion laws to maintain physical consistency. Consequently, PINNs enhance the accuracy and efficiency of PDE solutions and are an effective tool as data-driven, physics-informed neural networks evolve.

PINNs, unlike traditional numerical methods such as FEM, can solve problems by substituting or augmenting traditional requirements with empirical data. PINNs problems are typically formulated as follows [55]:

$$\begin{aligned} \boldsymbol{u}_t + \mathcal{N}[\boldsymbol{u}] = 0, \quad & t \in [0,T], \boldsymbol{x} \in \Omega, \\ \boldsymbol{u}(0,\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{x}), \quad & \boldsymbol{x} \in \Omega, \\ \mathcal{B}[\boldsymbol{u}] = 0, \quad & t \in [0,T], \boldsymbol{x} \in \partial\Omega, \end{aligned} \tag{1}$$

where $\boldsymbol{u}(t,\boldsymbol{x}) \in \mathbb{R}^{d_u}$ is the solution to the PDE, $t$ denotes time, $\boldsymbol{x}$ is a vector of spatial coordinates in the domain $\Omega$, and $\mathcal{N}[\cdot]$ is a linear or nonlinear differential operator. The function $g$ describes the initial condition (IC) of the PDE, and $\mathcal{B}[\cdot]$ is a boundary operator corresponding to Dirichlet, Neumann, Robin, or periodic boundary conditions.

In PINNs, the solution $\boldsymbol{u}(t,x)$ of the PDE is represented by a neural network $\boldsymbol{u}_\theta(t,x)$ with parameters $\theta$. This model can then be trained by minimizing a loss with three components:

$$\mathcal{L}(\theta) = \lambda_i \mathcal{L}_i(\theta) + \lambda_b \mathcal{L}_b(\theta) + \lambda_f \mathcal{L}_f(\theta). \tag{2}$$

The first two terms consist of a supervised loss that guarantees that the function learned by the neural network satisfies the initial and boundary conditions of the problem. That is, given randomly sampled points $\left\{\boldsymbol{x}_i^i\right\}_{i=1}^{N_i}$ from $\Omega$ and $\left\{t_b^i, x_b^i\right\}_{i=1}^{N_b}$ from $[0,T] \times \partial\Omega$, we have:

$$\mathcal{L}_i(\theta) = \frac{1}{N_i}\sum_{i=1}^{N_i}\left\|\boldsymbol{u}_\theta\left(0,\boldsymbol{x}_i^i\right)-\boldsymbol{g}\left(\boldsymbol{x}_i^i\right)\right\|_2^2, \quad \mathcal{L}_b(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b}\left\|\mathcal{B}\left[\boldsymbol{u}_\theta\right]\left(t_b^i,\boldsymbol{x}_b^i\right)\right\|_2^2. \tag{3}$$

The third term is the objective informed by physics, ensuring that the learned function $\boldsymbol{u}_\theta(t,x)$ satisfies the PDE by minimizing:

$$\mathcal{L}_f(\theta) = \frac{1}{N_f}\sum_{i=1}^{N_f}\left\|\frac{\partial \boldsymbol{u}_\theta}{\partial t}(t,x)+\mathcal{N}\left[\boldsymbol{u}_\theta\right](t,\boldsymbol{x})\right\|_2^2 \tag{4}$$

where the derivatives of $\boldsymbol{u}_\theta$ are calculated by automatic differentiation. This penalty is enforced on a set $\left\{t_f^i, x_f^i\right\}_{i=1}^{N_f}$ of randomly sampled points from the domain. Furthermore, these losses are weighted by hyper-parameters $\{\lambda_i, \lambda_b, \lambda_f\}$, leading to more flexibility during training. Figure 2 shows a general PINN framework to solve this problem.

*PINNs variant:* FI-PINNs (failure-informed PINNs) [20] enhance prediction in challenging regions by targeting under-predicted areas using adaptive sampling. HPINNs (Hard-Constrained Physics-Informed Neural Networks) [42] address tightly constrained inverse design problems by integrating hard constraints into the PINN framework via methods like the penalty method or the augmented Lagrangian.

gPINNs [66] improve accuracy and training efficiency by incorporating PDE gradient information into the loss function, especially for steep gradients. Bilevel PINNs [23] address PDE-constrained optimization using a bilevel optimization framework, with an inner loop for PDE constraints using PINNs and an outer loop for optimizing objectives, often using techniques like Broyden's hypergradients. FPINNs [51] solve problems defined by fractional differential equations, suitable for modeling complex anomalous diffusion in materials science and geophysics.

XPINNs (eXtended Physics-Informed Neural Networks) [31] address multi-domain problems by dividing the computational domain into subdomains, each trained with its own neural network, enhancing parallelism and efficiency with complex geometries and variable boundaries. Bayesian PINNs [65] apply Bayesian inference for uncertainty quantification, providing point estimates and uncertainty estimations crucial for geological simulations and predictive maintenance.

*Deep Ritz method:* The Deep Ritz method [67], designed to solve variational problems from PDE, is non-linear, adaptive, and suited to high dimensions, working well with stochastic gradient descent. Challenges include local minima, saddle points, and difficulties with essential boundary conditions. To address saddle-point issues, [61] introduces a deep double Ritz method using two neural networks to approximate functions in a nested minimization strategy.

*Neural Operators:* Neural operators enhance neural networks by learning mappings in function spaces, making them suitable for modeling spatio-temporal processes and PDEs. The Fourier Neural Operator (FNO) [39] uses the Fourier space for the integral kernel, achieving accuracy and efficiency, especially in fluid dynamics. Wavelet Neural Operators (WNOs) [60] use wavelet transformations to handle nonlinear PDEs, complex geometry, and boundary constraints. Graph Neural Operators (GNOs) [40], like the Multipole Graph Neural Operator (MGNO) [41], use graph neural networks for spatial relationships, efficiently handling long-range interactions with a hierarchical graph structure inspired by the fast multipole method. MGNOs excel in problems like Darcy flow and Burgers' equation, providing high accuracy and efficiency.

*Physics-Informed Transformer Networks:* Physics-Informed Transformer Networks (PITs) [16] blend transformer designs with physics-based learning. They employ attention processes to capture dependencies in input space, making them ideal for irregular grids and large-scale problems. PITs leverage data and physics-based constraints for robust solutions to time-varying PDE and complex issues, enhancing generalization and efficient dynamic system modeling.

*Newton Informed Neural Operator:* Classical neural network methods such as PINN, Deep Ritz, and DeepONet struggle with non-linear PDEs having multiple solutions due to their ill-posed nature. The Newton-informed neural operator [22] integrates classical Newton methods, reducing the need for supervised data points compared to other neural network approaches.

### 2.4 Examples of Partial Differential Equations

PDEs are equations with functions and their partial derivatives across multiple variables, essential for modeling physical phenomena like fluid flow, heat transfer, and wave propagation. Here are key PDE examples in science and engineering:

*Navier-Stokes Equations* The Navier-Stokes equations govern fluid motion, encapsulating momentum and mass conservation, crucial in fluid dynamics. For incompressible flow, these equations are:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{5}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{6}$$

where $\mathbf{u}$ is the velocity field, $p$ is the pressure field, $\nu$ is the kinematic viscosity, and $\mathbf{f}$ represents external forces.

*Burgers' Equation* Burgers' equation is a fundamental partial differential equation from fluid mechanics. It is used to model various types of wave phenomena and is given by:

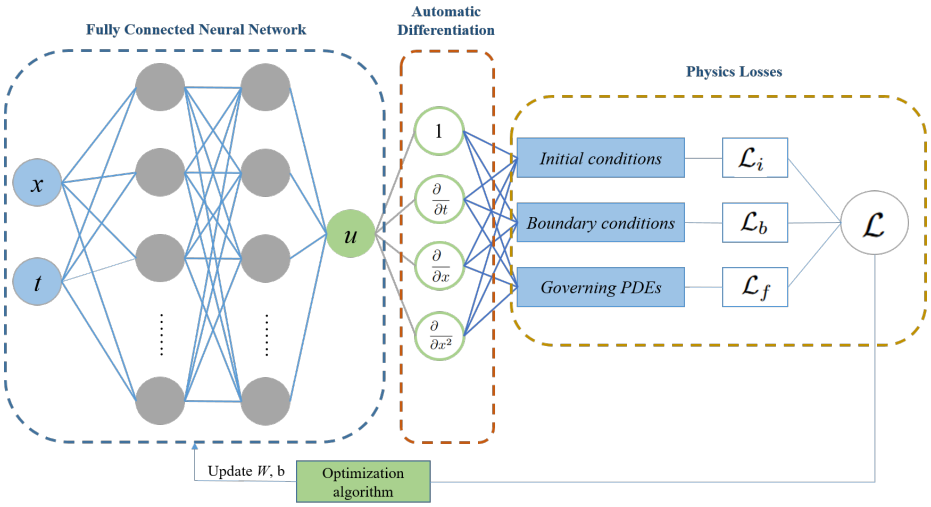


**Fig. 2.** A general framework to solve the forward problem of nonlinear PDE [68]

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \tag{7}$$

where $u$ is the velocity field and $\nu$ is the viscosity.

*Laplace's Equation* Laplace's equation, a second-order partial differential equation, describes scalar fields like electric potential and fluid velocity potential in fields such as fluid dynamics and electromagnetism. It is given by:

$$\nabla^2 \phi = 0 \tag{8}$$

where $\phi$ is the scalar potential function. In Cartesian coordinates, for a function $\phi(x, y, z)$, the equation expands to:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = 0 \tag{9}$$

*Inverse Problems Involving Nonlinear PDE* Inverse problems aim to find unknown parameters or inputs of a PDE from observed outputs. In fluid dynamics, this involves identifying the source term $\mathbf{f}$ in the Navier-Stokes equations from velocity and pressure observations. These are typically framed as optimization problems to minimize the discrepancy between observed data and model predictions, under PDE constraints.

# 3 Challenges associated with Neural Networks for solving PDE

## 3.1 Limitations of Neural Networks for solving PDE

Neural networks excel at challenging PDE tasks but have notable limitations, among them:

- Divergence behavior: Neural networks require consideration of their convergence behavior, which can be challenging due to potential entrapment in local minima during optimization. The weights in these networks, the unknowns in solving PDE, are derived using a less robust iterative solver compared to traditional numerical methods.
- Scalability: Neural networks are challenging to study in high-dimensional or complex domain spaces. As problem complexity increases, computational and memory demands grow, making it costly to simulate real-world problems requiring precise spatial solutions.
- Optimal Neural Network Architecture: Designing a neural network is crucial before exploring a task. Performance improves with the right number of neurons per layer and correct activation functions. Developing an optimal architecture is time-consuming and computationally intensive.
- Data-Gathering Issues: The performance of the neural network depends on the quality and volume of the data. Collecting sufficient and high-quality data is challenging and crucial for accurately describing a problem.
- Neural networks often need retraining whenever they face a new problem, a transformed problem, or a modified domain.
- Relying solely on physical laws in neural network training does not guarantee a perfect application. The physical correctness of a network's solution depends on the approximation accuracy. Proper problem solving requires precisely defined geometry, initial and boundary conditions.

## 3.2 Understanding of the convergence of neural networks

Neural network convergence occurs when the values of the training loss function consistently decrease, assuming that minimization is the goal.

Studying the convergence of neural networks in solving partial differential equations (PDEs) is crucial. Convergence means the neural network increasingly approximates the true PDE solution during training. Understanding this is vital for neural network-based methods.

**Convergence in Physics-Informed Neural Networks (PINNs)** The convergence of PINNs depends on the architecture of the neural network, the quality and quantity of data, and the optimization algorithms. The model minimizes a loss function by combining data with physical insights, often through addition or averaging, including terms for boundary conditions and governing equations.

- Optimization Landscape: The optimization landscape of PINNs is complex, with multiple local minima that can trap the optimization process. Adaptive learning rates, advanced gradient descent methods, and regularization techniques help navigate this landscape and encourage convergence towards a global or near-global minimum.
- Loss Function Composition: The composition of the loss function is crucial. PINNs employ a composite loss that includes data-driven and physics-based terms. The balance between these terms, controlled by hyperparameters, affects convergence. Improper weighting can lead to slow convergence and suboptimal solutions.
- Gradient pathologies, such as vanishing or exploding gradients, can hinder the convergence of PINNs. Using advanced activation functions such as ReLU, SELU, and batch normalization can help alleviate these issues and enhance training stability.

**Convergence in Neural Operators** Fourier and Wavelet Neural Operators learn mappings across function spaces, presenting convergence challenges. They require precise design and training to capture complex functional relationships accurately.

- Spectral methods like Fourier and Wavelet Neural Operators enhance convergence by efficiently capturing dominant solution modes. They require extensive training to perform well across various problem domains and scales.
- Transfer Learning: Convergence improves by pre-training neural operator models on related tasks and then retraining with specific datasets, speeding up convergence and enhancing generalization.

**Convergence in Transformer Networks** Transformer networks in numerical simulations use attention mechanisms to manage input space dependencies, with their complexity and input scale impacting convergence.

- Attention Mechanisms: Long-range dependencies in the data are effectively modeled through the self-attention mechanisms of transformer networks, which may lead to a more robust convergence. However, this is not without challenges, as it could be expensive to compute and require large datasets for training.
- Scalability and Efficiency: Scalability in transformer networks is a double-edged sword. They're effective for large problems, but convergence in high-dimensional spaces requires efficient methods like gradient clipping and distributed training.

**Convergence in Newton Informed Neural Operators** Newton Informed Neural Operators combine Newton methods with neural networks to improve convergence for nonlinear PDEs with multiple solutions.

- Hybrid Optimization: Integrating Newton's method provides a robust framework for managing solutions with better convergence rates, requiring fewer supervised data points and leveraging strengths from classical numerical methods and neural networks.
- Iterative Refinement: Newton-informed techniques iteratively enhance approximations for accurate solutions, effectively handling complex solution landscapes.

The convergence of neural network simulations is based on network architecture, loss function design, optimization algorithms, and domain characteristics of the problem. Key approaches like neural operators, PINNs, Transformer Networks, and Newton Informed N-operators require further study to optimize convergence. Addressing these challenges will make artificial neural networks more reliable and relevant for complex scientific and engineering computations.

## 4 Discussion

### 4.1 Performance metrics

The evaluation of numerical simulation programs focuses on the execution time of a set problem size. Performance is tested by varying the number of cores (e.g. 2, 4, 8, etc.) and measuring execution time for a fixed data size. This helps assess how execution time changes with different core counts, noting that doubling cores does not necessarily halve the time.

In addition, scalability testing can also be combined with measurements of :

- response time;
- requests per second, transactions per second;
- performance related to the number of users on the system;
- CPU and RAM usage during tests;
- network utilization - number of data items sent and received.

In PDE modeling, mesh generation is a complex yet vital process in numerical computation. It subdivides a 3D model into smaller flow domains for discretizing and solving governing equations. Mesh quality is crucial for accurate and stable simulation results.

Meshing is a simulation bottleneck; thus, ensuring high-quality meshes is crucial for fast and accurate analysis. Understanding mesh quality assessment criteria is essential, and PDE solvers assist engineers in evaluating mesh quality and estimating errors.

In fluid system simulation, the challenge begins with system geometry. Mesh discretization captures the unique flow features of each cell, but this is difficult because of the complexity of the domain, the complexity of the model, and the mesh type.

Regarding the good properties we may expect for a simulation result, we refer to [49]. This tutorial explains, for instance, the notions of errors and uncertainty.

In machine learning, loss functions measure model performance and are differentiable, aiding training via optimization methods like Gradient Descent. In contrast, performance metrics evaluate model performance during training and testing and need not be differentiable. Regression problems require metrics that calculate the distance between predictions and ground truth.

To evaluate regression models, we use metrics such as mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and R-squared $R^2$. Classification models are evaluated using Accuracy, Confusion Matrix, Precision and Recall, F1-score, and AU-ROC. Definitions of these metrics are available online.

The diverse performance metrics complicate the comparison of the results from classical PDE simulations and PINNs. Clarifying this issue remains an open question. The main concern is determining which technique provides better quality results and understanding the associated risks.

### 4.2 Frugal methods and algorithms

Sustainability is increasingly crucial due to the climate crisis, which includes energy efficiency, resource preservation, and flexible energy use in computing. The paper [13] examines AI's role in this crisis, urging a paradigm shift to foster a resilient society. Enhancing AI algorithm design is feasible for better memory efficiency. For example, in [14], Tri Dao developed a framework for efficient attention mechanisms in transformers.

To build on the previous section, we should consider whether there are performance metrics and tools to assess the environmental impact of techniques. Researchers must ensure that their algorithms and library components are efficient.

Python libraries like CodeCarbon[7], CarbonTracker[8], and ImpactTracker[3] help measure environmental indicators. CodeCarbon tracks and analyzes carbon emissions from compute engines. CarbonTracker predicts energy use and carbon footprint in deep learning model training. ImpactTracker promotes energy-efficient reinforcement learning through a leaderboard.

### 4.3 A system view of the ecosystem

The algorithms and tools are part of a larger neural network ecosystem for numerical simulations, which forms an organized framework and computing platform. In [15], authors collected papers on advances in High Performance Computing for AI, focusing on load-balancing schedulers for CPU–GPU systems, hardware accelerators, and rapid neural network training on GPU clusters for high-accuracy recognition.

In [48], the authors proposed a new direct-time Graph Neural Networks (GNN) architecture for irregular meshes, featuring increasing graph sizes with

[7] https://github.com/mlco2/codecarbon
[8] https://github.com/lfwa/carbontracker

spline convolutions. In [56], Melissa was introduced as a system for large-scale ensemble runs with thousands of simulations across varying inputs, which aids in sensitivity analysis, surrogate training, reinforcement learning, and data assimilation while handling large data volumes without storage. Melissa is a file-avoiding, fault-tolerant, and elastic framework for such ensemble runs on supercomputers.

INRIA France initiated significant projects in 2019. The "HPC - AI - Big-Data Convergence Days (Conv'2019)"[9] enabled the exploration of synergies between AI, Big Data, and HPC. It set the stage for 2024 discussions. A recent event in 2024, "GAP: Grenoble Artificial Intelligence for Physical Sciences"[10], included noteworthy presentations such as "Inferring effective state variables and Dynamics from Data", "Simulation-based Inference for the Physical Sciences", "An Overview of Operator Learning", and "Efficient Training of Deep Learning Models" by Julia Gusak, focusing on "Neural Ordinary Differential Equations (Neural ODEs)".

INRIA introduced DeepPhysX[11], a platform to bridge digital technology and AI algorithms such as neural networks.

## 5 Use cases and available resources on the Web

This section demonstrates web programming use cases to show the practical application of the concepts discussed.

### 5.1 Solving the problem with Neural Networks

1. Physics-Informed Neural Networks (PINNs):
   - PINNs[12]: Offers Tensorflow and Pytorch examples and tutorials on using PINNs for examples for solving the Burgers' and Helmholtz equations. The companion tutorial of the original PINN paper [52] is also available online[13].
2. Neural operators:
   - FNO: [14]: Implements neural operators that learn mappings between function spaces, useful for high-dimensional PDE and spatiotemporal processes. Includes examples of solving the Darcy equation.
   - WNO: [15]: Uses wavelet transformations and neural networks to solve highly nonlinear PDE, handling complex geometries and boundary conditions. It includes examples for 1-D Burger's equation, 2-D Allen-Cahn equation, 2-D Darcy equation, and 2-D Navier-Stokes equation.

[9] https://project.inria.fr/conv2019/
[10] https://discord.gg/PNUhG88SNC
[11] https://www.inria.fr/en/combining-numerical-simulation-artificial-intelligence
[12] https://github.com/omniscientoctopus/Physics-Informed-Neural-Networks
[13] https://maziarraissi.github.io/PINNs/
[14] https://github.com/neuraloperator/neuraloperator
[15] https://github.com/simon596/Wavelet-Neural-Operator

3. Physics-Informed Transformer Networks (PITNs):
   – PITNs [16]: Combine transformer architectures with physics-informed learning to solve PDE, including examples for 1D reaction, 1D wave, convection, and Navier-Stokes equations.

### 5.2 Solving the problem with conventional methods

1. Finite Element Method (FEM)
   – FEniCS[17] [5]: An open-source computing platform for solving PDE using FEM.
   – FreeFEM++ [18] [24]: open-source PDE solver that uses FEM for numerics. Provides a high-level programming language for the description of PDE.
2. Finite Difference Method (FDM)
   – FiPy[19]: A Python library for solving PDE using finite-volume and finite-difference methods.
3. Finite Volume Method (FVM)
   – OpenFOAM[20] [10]: An open-source C++ toolbox for the development of customized numerical solvers, and pre-/post-processing utilities for solving continuum mechanics problems, including CFD.
   – Manapy [21] [35]: A Python library for solving PDE using finite-volume methods using JIT compilation. Includes 2D and 3D examples using hybrid meshes.
4. Multigrid Methods
   – PyAMG[22] [6]: A Python library for Algebraic Multigrid (AMG) solvers.

## 6 Conclusion

This paper provides a comprehensive review of recent neural network advancements in solving partial differential equations (PDE), focusing on methods like Physics-Informed Neural Networks (PINNs), Neural Operators, Transformers, and Newton Informed Neural Operators.

Physics-Informed Neural Networks (PINNs) have been developed to incorporate physical laws into neural network training, thus enhancing their ability to accurately solve partial differential equations (PDEs). PINNs have also evolved into various derivatives, such as Failure-Informed, Hard-Constrained, and Gradient-Enhanced PINNs, each addressing specific challenges to improve performance in different contexts.

[16] https://github.com/AdityaLab/pinnsformer
[17] https://github.com/FEniCS
[18] https://github.com/FreeFem
[19] https://github.com/usnistgov/fipy
[20] https://github.com/OpenFOAM
[21] https://github.com/imadki/manapy
[22] https://github.com/pyamg/pyamg

Neural operators are neural networks that map between function spaces, making Fourier and Wavelet Neural Operators effective for modeling spatio-temporal processes and solving complex PDEs. They are highly accurate and computationally efficient, particularly for high-dimensional fluid dynamics and problems.

Physics-Informed Transformers: Originally designed for natural language, Transformer Networks are now adapted through Physics-Informed Transformers for numerical simulations, leveraging attention mechanisms to capture dependencies across input spaces, ideal for irregular grids and large-scale problems.

The Newton Informed Neural Operator blends classical Newton methods with neural networks, introducing a novel approach. It not only differentiates between multiple solutions, but also uses less supervised data than current neural network techniques.

Despite progress, challenges remain, notably convergence and stability in high-dimensional spaces. More work is needed to scale methods to large problems and design efficient networks. The quality and quantity of training data significantly affect the performance of the model.

Future research should explore improved training algorithms, advanced hardware, and hybrid approaches that combine neural networks with numerical methods. Customizing neural network architectures for specific science problems is crucial for efficiency.

## References

1. Aboussi, W., Ziggaf, M., Kissami, I., Boubekeur, M.: A highly efficient finite volume method with a diffusion control parameter for hyperbolic problems. Mathematics and Computers in Simulation (2023)
2. Anderson, J.D., Wendt, J.: Computational fluid dynamics, vol. 206. Springer (1995)
3. Anthony, L.F.W., Kanding, B., Selvan, R.: Carbontracker: Tracking and predicting the carbon footprint of training deep learning models (2020). https://doi.org/10.48550/ARXIV.2007.03051, https://arxiv.org/abs/2007.03051
4. Baliga, B., Patankar, S.: A new finite-element formulation for convection-diffusion problems. Numerical Heat Transfer **3**(4), 393–409 (1980)
5. Baratta, I.A., Dean, J.P., Dokken, J.S., Habera, M., Hale, J.S., Richardson, C.N., Rognes, M.E., Scroggs, M.W., Sime, N., Wells, G.N.: DOLFINx: the next generation FEniCS problem solving environment. preprint (2023). https://doi.org/10.5281/zenodo.10447666
6. Bell, N., Olson, L.N., Schroder, J.: Pyamg: Algebraic multigrid solvers in python. Journal of Open Source Software **7**(72), 4142 (2022)
7. Benkhaldoun, F., Cérin, C., Kissami, I., Saad, W.: Challenges of translating hpc codes to workflows for heterogeneous and dynamic environments. In: 2017 International Conference on High Performance Computing & Simulation (HPCS). pp. 858–863. IEEE (2017)
8. Burnham, W.H.: Memory, historically and experimentally considered. i. an historical sketch of the older conceptions of memory. The American Journal of Psychology **2**(1), 39–90 (1888)

9. Calzolari, G., Liu, W.: Deep learning to replace, improve, or aid cfd analysis in built environment applications: A review. Building and Environment **206**, 108315 (2021)
10. Chen, G., Xiong, Q., Morris, P.J., Paterson, E.G., Sergeev, A., Wang, Y.: Openfoam for computational fluid dynamics. Notices of the AMS **61**(4), 354–363 (2014)
11. Chen, J., Nakao, J., Qiu, J.M., Yang, Y.: A high-order eulerian-lagrangian runge-kutta finite volume (el-rk-fv) method for scalar conservation laws. arXiv preprint arXiv:2405.09835 (2024)
12. Constantin, P., Foiaş, C.: Navier-stokes equations. University of Chicago press (1988)
13. Couillet, R., Trystram, D., Ménissier, T.: The submerged part of the ai-ceberg [perspectives]. IEEE Signal Process. Mag. **39**(5), 10–17 (2022). https://doi.org/10.1109/MSP.2022.3182938, https://doi.org/10.1109/MSP.2022.3182938
14. Dao, T.: Flashattention-2: Faster attention with better parallelism and work partitioning (2023)
15. Dias de Assuncao, M., Rocha Rodrigues, E., Raffin, B.: Preface – special issue advances on high performance computing for artificial intelligence. Journal of Parallel and Distributed Computing **156**, 131 (2021). https://doi.org/https://doi.org/10.1016/j.jpdc.2021.06.002, https://www.sciencedirect.com/science/article/pii/S0743731521001313
16. Dos Santos, F., Akhound-Sadegh, T., Ravanbakhsh, S.: Physics-informed transformer networks. In: The Symbiosis of Deep Learning and Differential Equations III (2023)
17. Elmisaoui, S., Kissami, I., Ghidaglia, J.M.: High-performance computing to accelerate large-scale computational fluid dynamics simulations: A comprehensive study. In: International Conference on Advanced Intelligent Systems for Sustainable Development. pp. 352–360. Springer (2023)
18. ENKIEWICZ, O.C.Z.: Introductory lectures on the finite element method. MECHANICS OF SOLIDS (1972)
19. Euler, L.: Introductio in analysin infinitorum, volume 2 (1748)
20. Gao, Z., Yan, L., Zhou, T.: Failure-informed adaptive sampling for pinns. SIAM Journal on Scientific Computing **45**(4), A1971–A1994 (2023)
21. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al.: Recent advances in convolutional neural networks. Pattern recognition **77**, 354–377 (2018)
22. Hao, W., Liu, X., Yang, Y.: Newton informed neural operator for computing multiple solutions of nonlinear partials differential equations (2024)
23. Hao, Z., Ying, C., Su, H., Zhu, J., Song, J., Cheng, Z.: Bi-level physics-informed neural networks for pde constrained optimization using broyden's hypergradients. arXiv preprint arXiv:2209.07075 (2022)
24. Hecht, F., Pironneau, O., Le Hyaric, A., Ohtsuka, K.: Freefem++ manual. Laboratoire Jacques Louis Lions (2005)
25. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2016)
26. Hinton, G.E.: Deep belief networks. Scholarpedia **4**(5), 5947 (2009)
27. Hong, H.K., Chen, J.T.: Derivations of integral equations of elasticity. Journal of Engineering Mechanics **114**(6), 1028–1044 (1988)
28. Hua, Y., Guo, J., Zhao, H.: Deep belief networks and deep learning. In: Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things. pp. 1–4. IEEE (2015)

29. Huray, P.G.: Maxwell's equations. John Wiley & Sons (2009)
30. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37. p. 448–456. ICML'15, JMLR.org (2015)
31. Jagtap, A.D., Karniadakis, G.E.: Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. Communications in Computational Physics **28**(5) (2020)
32. Jin, X., Cai, S., Li, H., Karniadakis, G.E.: Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. Journal of Computational Physics **426**, 109951 (2021)
33. Kassam, A.K., Trefethen, L.N.: Fourth-order time-stepping for stiff pdes. SIAM Journal on Scientific Computing **26**(4), 1214–1233 (2005)
34. Kiener, A., Langer, S., Bekemeyer, P.: Data-driven correction of coarse grid cfd simulations. Computers & Fluids p. 105971 (2023)
35. Kissami, I.: Manapy: An mpi-based python framework for solving poisson's equation using finite volume on unstructured-grid. In: AIP Conference Proceedings. vol. 3034. AIP Publishing (2024). https://doi.org/https://doi.org/10.1063/5.0194750
36. Kou, J., Zhang, W.: Data-driven modeling for unsteady aerodynamics and aeroelasticity. Progress in Aerospace Sciences **125**, 100725 (2021)
37. Lagrange, J.L.: Théorie des fonctions analytiques (paris). Oeuvres de Lagrange IX (1797)
38. Li, Z., Liu, F., Yang, W., Peng, S., Zhou, J.: A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems (2021)
39. Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A.: Fourier neural operator for parametric partial differential equations (2020). arXiv preprint arXiv:2010.08895 (2010)
40. Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A.: Neural operator: Graph kernel network for partial differential equations. arXiv preprint arXiv:2003.03485 (2020)
41. Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Stuart, A., Bhattacharya, K., Anandkumar, A.: Multipole graph neural operator for parametric partial differential equations. Advances in Neural Information Processing Systems **33**, 6755–6766 (2020)
42. Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., Johnson, S.G.: Physics-informed neural networks with hard constraints for inverse design. SIAM Journal on Scientific Computing **43**(6), B1105–B1132 (2021)
43. Maazioui, S., Kissami, I., Benkhaldoun, F., Ouazar, D.: Numerical study of viscoplastic flows using a multigrid initialization algorithm. Algorithms **16**(1), 50 (2023)
44. Mao, Z., Jagtap, A.D., Karniadakis, G.E.: Physics-informed neural networks for high-speed flows. Computer Methods in Applied Mechanics and Engineering **360**, 112789 (2020)
45. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**, 115–133 (1943)
46. Medsker, L., Jain, L.C.: Recurrent neural networks: design and applications. CRC press (1999)

47. Medsker, L.R., Jain, L.: Recurrent neural networks. Design and Applications **5**(64-67), 2 (2001)
48. Meyer, L.T., Pottier, L., Ribés, A., Raffin, B.: Deep surrogate for direct time fluid dynamics. CoRR **abs/2112.10296** (2021), https://arxiv.org/abs/2112.10296
49. NASA: Tutorial on cfd verification and validation, https://www.grc.nasa.gov/www/wind/valid/tutorial/tutorial.html
50. Obiols-Sales, O., Vishnu, A., Malaya, N., Chandramowliswharan, A.: Cfdnet: A deep learning-based accelerator for fluid simulations. In: Proceedings of the 34th ACM international conference on supercomputing. pp. 1–12 (2020)
51. Pang, G., Lu, L., Karniadakis, G.E.: fpinns: Fractional physics-informed neural networks. SIAM Journal on Scientific Computing **41**(4), A2603–A2626 (2019)
52. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics **378**, 686–707 (2019)
53. Richardson, L.: The approximate solution of various boundary problems by surface integration combined with freehand graphs. Proceedings of the Physical Society of London **23**(1), 75 (1910)
54. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review **65**(6), 386 (1958)
55. Santos, F.D., Akhound-Sadegh, T., Ravanbakhsh, S.: Physics-informed transformer networks. In: The Symbiosis of Deep Learning and Differential Equations III (2023), https://openreview.net/forum?id=zu80h9YryU
56. Schouler, M., Caulk, R.A., Meyer, L.T., Terraz, T., Conrads, C., Friedemann, S., Agarwal, A., Baldonado, J.M., Pogodzinski, B., Sekula, A., Ribés, A., Raffin, B.: Melissa: coordinating large-scale ensemble runs for deep learning and sensitivity analyses. J. Open Source Softw. **8**(87), 5291 (2023). https://doi.org/10.21105/JOSS.05291, https://doi.org/10.21105/joss.05291
57. Smith, G.D.: Numerical solution of partial differential equations: finite difference methods. Oxford university press (1985)
58. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**(56), 1929–1958 (2014), http://jmlr.org/papers/v15/srivastava14a.html
59. Steinhauser, M.O.: Fundamentals of Numerical Simulation, pp. 185–225. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-98954-5_4, https://doi.org/10.1007/978-3-030-98954-5_4
60. Tripura, T., Chakraborty, S.: Wavelet neural operator: a neural operator for parametric partial differential equations. arXiv preprint arXiv:2205.02191 (2022)
61. Uriarte, C., Pardo, D., Muga, I., Muñoz-Matute, J.: A deep double ritz method (d2rm) for solving partial differential equations using neural networks. Computer Methods in Applied Mechanics and Engineering **405**, 115892 (2023). https://doi.org/https://doi.org/10.1016/j.cma.2023.115892, https://www.sciencedirect.com/science/article/pii/S0045782523000154
62. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 6000–6010. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)
63. Wang, J.X., Xiao, H.: Data-driven cfd modeling of turbulent flows through complex structures. International Journal of Heat and Fluid Flow **62**, 138–149 (2016)

64. Wilkes, A.L., Wade, N.J.: Bain on neural networks. Brain and cognition **33**(3), 295–305 (1997)
65. Yang, L., Meng, X., Karniadakis, G.E.: B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data. Journal of Computational Physics **425**, 109913 (2021)
66. Yu, J., Lu, L., Meng, X., Karniadakis, G.E.: Gradient-enhanced physics-informed neural networks for forward and inverse pde problems. Computer Methods in Applied Mechanics and Engineering **393**, 114823 (2022)
67. Yu, W.E..B.: The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. Communications in Mathematics and Statistics **6**, 1–12 (2018)
68. Yuan, L., Ni, Y.Q., Deng, X.Y., Hao, S.: A-pinn: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations. Journal of Computational Physics **462**, 111260 (2022). https://doi.org/https://doi.org/10.1016/j.jcp.2022.111260, https://www.sciencedirect.com/science/article/pii/S0021999122003229
69. Zhu, Y., Dinh, N.: A data-driven approach for turbulence modeling. arXiv preprint arXiv:2005.00426 (2020)