# Artificial intelligence for partial differential equations in computational mechanics: A review [1]

Yizheng Wang[a,b], Jinshuai Bai[a,e,f], Zhongya Lin[a], Qimin Wang[c], Cosmin Anitescu[b], Jia Sun[d], Mohammad Sadegh Eshaghi[c], Yuantong Gu[e,f], Xi-Qiao Feng[a], Xiaoying Zhuang[c], Timon Rabczuk[b], Yinghua Liu[a,*]

[a] *Department of Engineering Mechanics, Tsinghua University, Beijing 100084, China*
[b] *Institute of Structural Mechanics, Bauhaus-Universität Weimar, Marienstr. 15, D-99423 Weimar, Germany*
[c] *Chair of Computational Science and Simulation Technology, Institute of Photonics, Leibniz University Hannover, Hannover 30167, Germany*
[d] *Drilling Mechanical Department, CNPC Engineering Technology RD Company Limited, Beijing 102206, China*
[e] *School of Mechanical, Medical and Process Engineering, Queensland University of Technology, Brisbane, QLD 4000, Australia*
[f] *ARC Industrial Transformation Training Centre—Joint Biomechanics, Queensland University of Technology, Brisbane, QLD 4000, Australia*

**Abstract**

In recent years, Artificial intelligence (AI) has become ubiquitous, empowering various fields, especially integrating artificial intelligence and traditional science (AI for Science: Artificial intelligence for science), which has attracted widespread attention. In AI for Science, using artificial intelligence algorithms to solve partial differential equations (AI for PDEs: Artificial intelligence for partial differential equations) has become a focal point in computational mechanics. The core of AI for PDEs is the fusion of data and partial differential equations (PDEs), which can solve almost any PDEs. Therefore, this article provides a comprehensive review of the research on AI for PDEs, summarizing the existing algorithms and theories. The article discusses the applications of AI for PDEs in computational mechanics, including solid mechanics, fluid mechanics, and biomechanics. The existing AI for PDEs algorithms include those based on Physics-Informed Neural Networks (PINNs), Deep Energy Methods (DEM), Operator Learning, and Physics-Informed Neural Operator (PINO). AI for PDEs represents a new method of scientific simulation that provides approximate solutions to specific problems using large amounts of data, then fine-tuning according to specific physical laws, avoiding the need to compute from scratch like traditional algorithms. Thus, AI for PDEs is the prototype for future foundation models in computational mechanics, capable of significantly accelerating traditional numerical algorithms.

*Keywords:* Artificial intelligence, Artificial intelligence for science, Artificial intelligence for partial differential equations, Computational mechanics, Physics-informed neural networks, Operator learning

## 1. Introduction

Over the past decade, artificial intelligence has had a profound impact on multiple fields. Currently, machine learning is widely regarded as a key means to achieve artificial intelligence. An apt metaphor describes machine

[1] We note that we have published an article in a Chinese journal: Wang Y Z, Zhuang X Y, Timon R, Liu Y H. AI for PDEs in solid mechanics: A review. Advances in Mechanics, 2024 [1]. Given that the methodology of AI for PDEs is similar, the methodologies and theories in this review are consistent with those in [1]. We have obtained permission from the Chinese journal *Advances in Mechanics* to publish this work in English-language version. The primary difference between this review and [1] is that we have expanded the discussion to include the application of AI for PDEs to both forward and inverse problems in fluid mechanics and biomechanics. Considering the significant attention AI for PDEs has garnered in computational mechanics, this review is valuable and should be shared with the global community through an English version.

*Corresponding author
*Email addresses:* wang-yz19@tsinghua.org.cn (Yizheng Wang), timon.rabczuk@uni-weimar.de (Timon Rabczuk), yhliu@mail.tsinghua.edu.cn (Yinghua Liu)

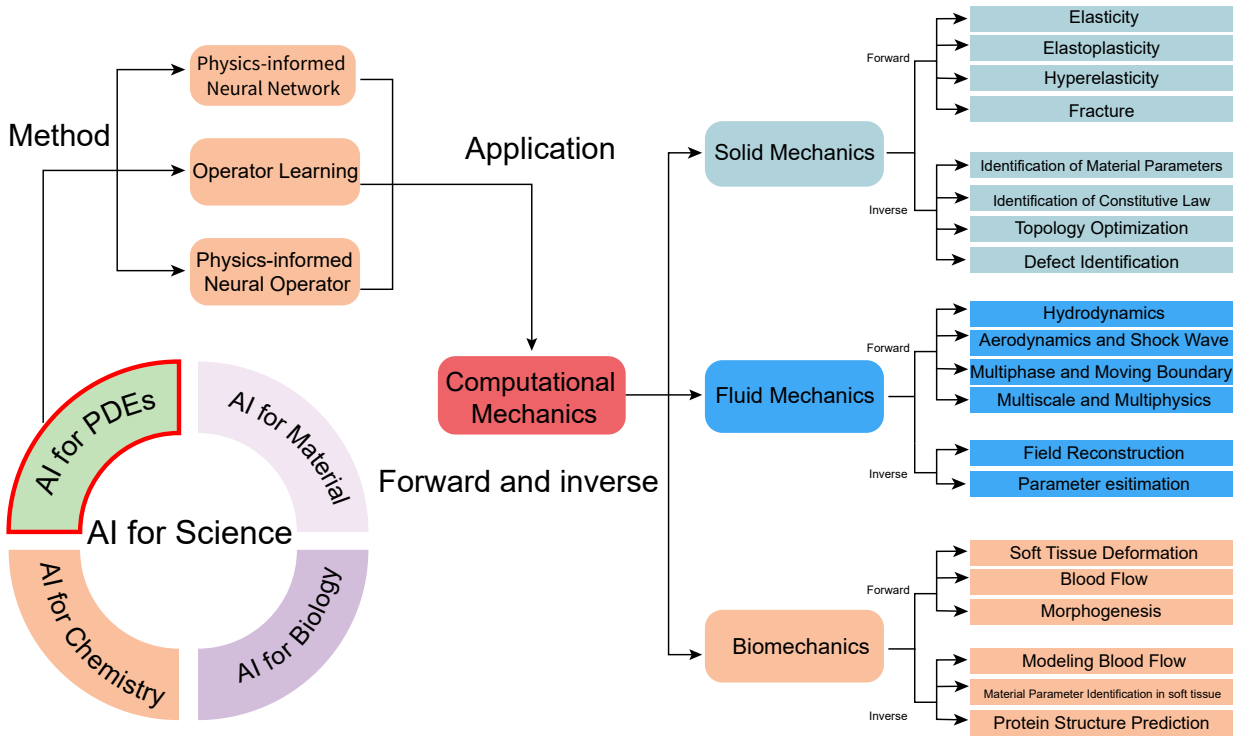


**Fig. 1.** The role of AI4PDEs in AI4Science, along with an introduction to AI4PDEs in computational mechanics, including solid, fluid, and biomechanics..

learning as "data-driven optimization algorithms," which is due to the fact that machine learning often requires a large amount of data for optimization. Additionally, since the physical laws behind some phenomena are not very clear, statistical optimization methods become particularly important for model construction. This is especially apparent in dealing with natural perception data, such as images and speech, which are easily understood by humans but difficult for computers to comprehend. Broadly speaking, current artificial intelligence is roughly equivalent to machine learning, with deep learning being one of its algorithms. Deep learning has been widely used in numerous fields and has achieved revolutionary success, particularly in areas such as computer vision [2], speech recognition [3], natural language processing [4], strategy games [5, 6], and drug development [7]. It can be said that deep learning is everywhere, empowering various fields. Recently, the integration of AI and traditional science (AI4Science: AI for Science) has drawn broad attention [8]. Due to many physical computing problems being closely related to the solution of partial differential equations (PDEs), using artificial intelligence algorithms to solve PDEs [9], a key research area known as AI4PDEs, has gained substantial interest among researchers in computational physics and mathematics [10]. This area is recognized as one of the most important research directions within the broader field of AI4Science [11]. Fig. 1 displays the relationship between AI4PDEs and AI4Science.

The field of computational mechanics has also been impacted by AI4Science. The integration of AI4Science and computational mechanics mainly occurs in two aspects. One is through deep learning methods, utilizing real experimental data or reliable numerical results [12] (such as those obtained from finite element methods) and then using neural networks to construct surrogate models. This is a type of implicit programming [13], where "implicit programming" refers to inputting data and utilizing the strong fitting ability of AI algorithms to output "programs," which consist of neural network parameters, thus eliminating the need for humans to write programs to solve particular problems specifically. This traditional method of explicitly writing programs to solve specific problems often requires designing programs to transform abstract principles into computer-readable code. Artificial intelligence algorithms represent a form of implicit programming that uses data to

build surrogate models. The core idea is to use the fitting power of neural networks to model the abstract relationships between data [12]. If training is successful, and the test set and training set are consistent in data distribution, surrogate models often exhibit very high computational efficiency and accuracy on the test set. However, this method has some unavoidable problems, such as requiring large amounts of data, and the quality and quantity of data determining the effectiveness of the model. This integration method relies entirely on data, necessitating the use of existing methods to obtain relevant data. Therefore, it faces challenges like the curse of dimensionality [14, 15]. Additionally, due to the lack of physical constraints, the interpretability is poor, leading to consequences that make it difficult to improve the model's effectiveness. This disadvantage causes a significant amount of manual, trial and error based parameter tuning, so using neural networks to establish surrogate models is essentially a black-box approach. The choice of surrogate models often benefits from computer vision algorithms to better integrate with current physical problems. For instance, computer vision algorithms like the Swin Transformer were used to predict weather in the PanGu foundation model [16], or to predict the equivalent modulus of non-uniform materials [12]. Overall, the first aspect is a completely data-driven modeling method. The limitation of this type of algorithm is that it only has the potential to be faster, but it is difficult to have the potential to be more accurate, because the data-driven modeling method is based on fitting existing high-precision data, and the accuracy can only be infinitely close to the existing high-precision data.

The other important aspect of the integration between AI4Science and mechanics is the incorporation of physical laws into the loss function of neural networks, which is a core part of AI4PDEs. Using physical information, by which the need for high-quality data is reduced [17], often reflected in solving PDEs, i.e., using AI4PDEs algorithms to solve PDEs of computational mechanics. The introduction of physical equations reduces the data requirement because it incorporates inductive biases, which refers to the assumptions on which algorithm models are based, helping the model to make predictions and reducing the amount of data needed. PDEs represent a special form of inductive bias, as physical equations are often derived using more basic assumptions (such as assumptions of linearity, small deformations, and continuity in elasticity mechanics) and then derived with mathematical tools to describe the laws of material motion. Thus, PDEs are a form of advanced inductive bias [18].

The use of deep learning to solve PDEs (AI4PDEs) first appeared with the Physics-Informed Neural Networks (PINNs), a term coined by Raissi et al. in 2019 [9]. However, the concept of using neural networks to solve PDEs dates back to 1990 [19], 1994 [20] and 1998 [21]. Because **deep** neural networks (DNNs), powerful GPU computing, and associated software tools were not available at that time, these ideas did not receive sufficient attention. Recently, with the rapid development of computer hardware, advancements in machine learning algorithms, and the convenient implementation of automatic differentiation algorithms within artificial intelligence frameworks like PyTorch and TensorFlow, the PINNs algorithm has garnered considerable attention, leading to an increase in the complexity of the problems that can be solved [22]. Compared to traditional numerical methods, PINNs has two main advantages: firstly, utilizing the powerful fitting capabilities of neural networks for solving challenging PDEs problems, such as those involving significant nonlinearity, convection-dominated, or shock problems, provides a new method of scientific computation [22]. For forward problems, since PINNs are a type of mesh-free method where the approximate function is a neural network and the solution is obtained for the entire domain at once, they share similar advantages with mesh-free methods, such as handling large deformation problems with significant mesh distortion. For inverse problems, the code format can be easily adapted, allowing for straightforward application with automatic differentiation (AD: Automatic Differentiation) [23]. The second advantage is its suitability for solving high-dimensional problems, mainly due to the construction of the PINNs loss function using the Monte Carlo method. The Monte Carlo method is a powerful tool for high-dimensional integral calculations. Therefore, the advantages of PINNs in solving high-dimensional problems do not originate from neural networks but from the Monte Carlo method used for the construction of loss function. In theory, if traditional methods also adopt the Monte Carlo approach for integral calculations, they would possess the same capabilities as PINNs in solving high-dimensional problems. PINNs can solve almost all PDE problems because the core of PINNs involves setting the approximate function as a neural network, so the computational approach is similar to traditional methods. Thus, any PDEs problem that traditional methods can solve, PINNs can also address, such as linear PDEs [24, 25], non-linear PDEs [26, 27], stochastic PDEs [28, 29], and integral differential equations (IDEs) [30]. PINNs also have drawbacks,

including a lack of robustness, accuracy, and computational efficiency [31] in many applications. For instance, even when the forward problem is well-posed and linear, the PINN formulation can lead to ill-posed (nonlinear) optimization problems [32].

Mathematically, PINNs are divided into two formats: the original strong form of PINNs [9] and the energy form of PINNs [25]. The strong form and energy form of PINNs are explained in detail in Section 2.1.1 and Section 2.1.2, respectively. The energy form is the deep Ritz method introduced by Yu in 2018 [25], which was later applied to the field of computational mechanics by Samaniego et al. (2020) [33], proposing the Deep Energy Method (DEM). The core idea of DEM is minimizing the potential energy of the system. In contrast to the strong form of PINNs, which utilizes PDEs, DEM approaches the problem through the system's total energy using the variational principle. Notably, although DEM mainly follows the energy form, it also incorporates the idea of the strong form of PINNs in the original review [33], called the deep collocation method (DCM), which is the strong form of PINNs. It can be said that DEM is the first major summary and large-scale application of PINNs in computational mechanics. The main difference between the strong form and energy form of PINNs lies in the composition of the loss function. The loss function of the strong form combines PDEs directly through weighted residuals, while the loss function of DEM is based on the principle of least action, such as the principle of minimum potential energy in mechanics. Both the strong form and energy form of PINNs are shown in the upper left corner of Fig. 2. A significant advantage of PINNs is that they can combine data and physical equations, making them a semi-supervised algorithm particularly suitable for problems involving both data and equations. Additionally, PINNs can be easily adapted for inverse problems, with the programming code being nearly identical to that for forward problems. It only requires setting the variables to be solved in the inverse problem as trainable optimization variables and incorporating them into the loss function. Thus, compared to traditional inverse problem algorithms, PINNs is very easy to program for solving inverse problems [10].

Overall, PINNs leverage existing physical laws, significantly reducing the need for datasets and offering better interpretability than purely data-driven approaches. However, assigning physical meaning to the hyper-parameters in PINNs remains challenging, especially in comparison to the finite element method (FEM). The accuracy of PINNs in solving forward problems was difficulties in surpassing traditional methods. The main reason is the non-convex nature of neural networks. Although the non-convexity of neural networks provides powerful fitting capabilities, it also introduces difficulties in optimization. PINNs are suitable for problems with clear physical laws and small data volumes. Of course, if the data volume is large enough, PINNs can be used, but in such cases, operator learning would be more appropriate (please note that the original PINNs does not require data for solving; data is optional, not mandatory).

In recent years, operator learning has become a hot research topic, as shown in the upper right corner of Fig. 2. Representative works include DeepONet [34] and FNO (Fourier Neural Operator) [35]. The initial proposals for operator learning were purely data-driven, making them highly suitable for large data volume problems (even if the physical processes are unclear, operator learning can still rely on learning the abstract complex relationships within the data). The difference from traditional data-driven methods lies in the reliable mathematical theory supporting operator learning and the advantage of discretization-invariance in operator learning. In terms of theoretical support, for example, DeepONet is based on the theory proposed by Chen et al. (1995) [39] that neural networks can fit any continuous operator, which led to the design of the algorithm. FNO is based on Fourier transforms; discretization-invariance roughly refers to the ability to train and test on any grid resolution, and after the mesh size changes, retraining is not required. This will be explained in detail in Section 2.2. It is important to note that some neural operator algorithms actually learn the mapping from a discrete input function to a discrete output function space, rather than the differential operator itself. To address this, Bartolucci et al. [40] proposed RENOs (Representation Equivalent Neural Operators) to learn the operator mapping of PDEs.

However, many complex engineering problems face the challenge of unclear physical processes. Even if PDEs are used to describe them, they are not completely accurate, and the data volume is also limited. The latest development of physics-informed neural operators (PINO: Physics-informed neural operator [37]) can handle these issues of vague physics and insufficient data to some extent, as shown in the lower left corner of Fig. 2. Initially, an approximate solution is obtained by relying on approximate physical equations and corresponding boundary initial conditions, which is then fine-tuned using limited data [38]. Additionally, for problems with large data volumes and clear physical processes, physics-informed neural operators can also be effectively applied

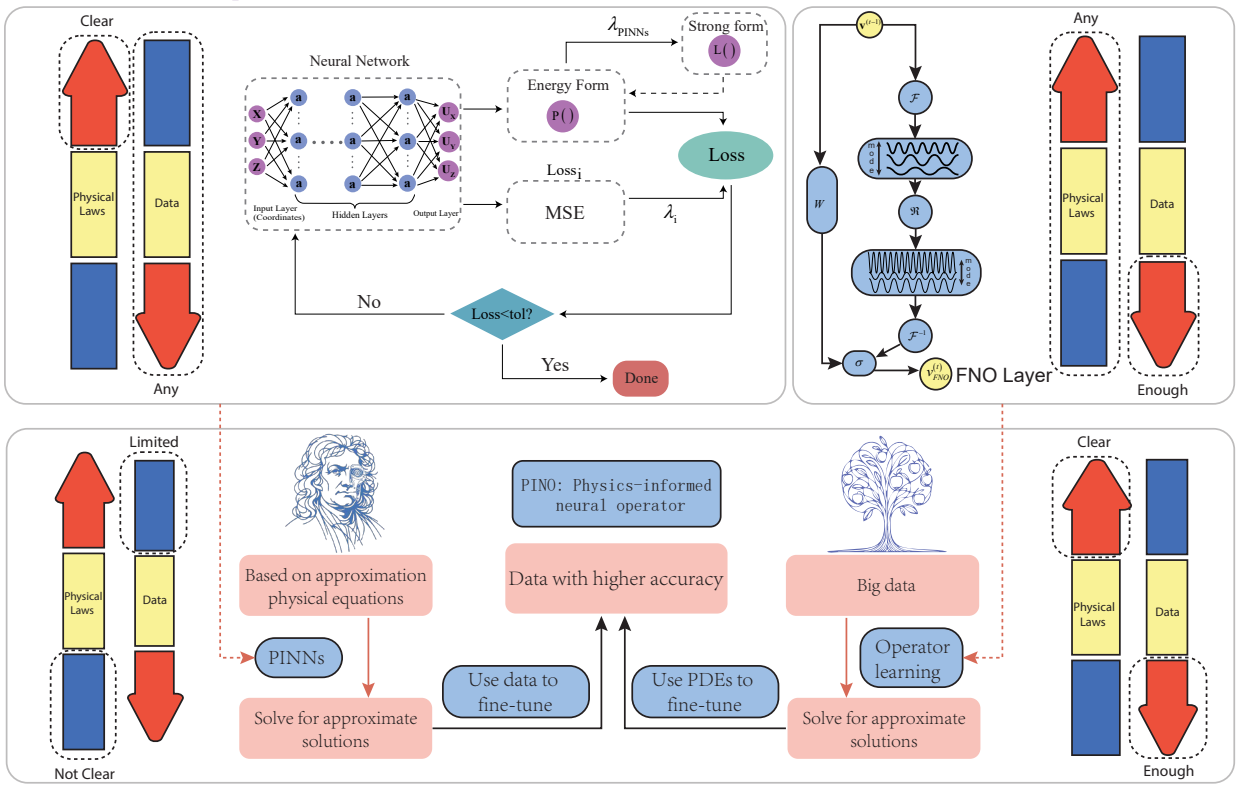


**Fig. 2.** Main methods of AI4PDEs: Physics-informed neural networks [9, 33, 25], operator learning [34, 35, 36], and physics-informed neural operators [37, 38].

to these issues [41, 37, 42, 32], as shown in the lower right corner of Fig. 2. The approach is to first obtain a good approximate solution using operator learning and then fine-tune it based on clear physical processes. Since solving PDEs relies on operator learning partly, it can significantly accelerate PDEs computations, thousands to tens of thousands of times faster than traditional numerical methods [43]. At the same time, iterating the approximate solution by operator learning according to the physical equation does not require much time because the initial solution (approximate solution) is not far from the true solution. This simultaneously possesses the speed of operator learning and the accuracy of physical equations. Therefore, it can reduce the dependence on supercomputers for solving complex PDEs and solve larger problems with fixed computing power. Thus, the potential of PINO is substantial not only in academia but also in industry. A detailed explanation will be provided in Section 2.3. It is worth noting that PINO is unlike the initially proposed PINNs which can only solve specific problems, i.e., once boundary conditions, geometric shapes, or materials change, a new solution is required in PINNs. However, PINO learns a series of mappings of PDEs family, allowing for rapid attainment of target solutions even if the aforementioned conditions change. Therefore, this will be one of the most promising directions in the field of computational mechanics. The essence of PINO is the combination of operators and physical equations, incorporating the ideas of PINNs in the physical equations and operator learning using data-driven like DeepONet or FNO. Therefore, PINO simultaneously includes PINNs and operator learning. If the data volume is large enough, PINO can be reduced to operator learning. If the physical process is clear, it can be reduced to PINNs.

The outline of this article is shown in Fig. 1. Section 2 systematically summarizes AI4PDEs algorithms, including the strong form of PINNs, energy form, operator learning, and physics-informed neural operators. Section 3 introduces some theoretical work on AI4PDEs. Sections 4 and 5 discuss some applications of AI4PDEs in computational mechanics, including forward and inverse problems in solid, fluid, and biomechanics[2]. Section 6 predicts the likely emergence of the foundation model in computational mechanics and the potential opportunities and challenges AI4PDEs may face in the future.

## 2. AI for PDEs: Methodology

A multitude of physical phenomena rely on PDEs for modeling. Once the boundary and initial conditions become complex, the analytical solution of PDEs is often difficult to obtain. At this point, various numerical methods are employed to achieve approximate solutions, such as the commonly used finite element methods [44, 45, 46, 47], mesh-free methods [48, 49, 50, 51, 52, 53], finite difference methods [54], finite volume methods [55], boundary element methods [56], and spectral methods [57]. Despite the great success of traditional numerical methods in computational mechanics over the past 50 years, we still struggle to integrate data into traditional algorithms, especially in industries where multimodal, multi-resolution, and high-error complex data are common [58]. Moreover, it is also a challenge that finding a PDEs can describe such complex systems [36, 38]. Therefore, integrating data into computational models is of significant importance to describe complex systems. Additionally, when solving inverse problems and large, complex nonlinear problems, traditional algorithms often involve large computational loads and complex formats, especially when dealing with highly complex nonlinear issues, which require vast amounts of code and are not friendly to updates [10], such as OpenFOAM [59], which has more than 100,000 lines of code. For low-dimensional and relatively simple geometric problems, traditional numerical methods can offer high computational accuracy and efficiency.

While these methods, such as the FEM, are quite effective in dealing with many complex problems, challenges can arise when dealing with high-dimensional problems and complex geometric shapes [15, 60]. However, the latest AlphaGeometry [61] may assist mesh generators in the future. Additionally, traditional FEM often require the selection of specific basis functions (shape functions).

AI for PDEs, unlike traditional numerical methods, refers to a class of algorithms that use deep learning to solve PDEs. From the perspective of deep learning, it can be seen as learning a mapping relationship, which

[2]Please note that most of the images in Section 4 and Section 5 are adapted from the most important papers closely related to this review, combined with text and formulas, to clearly explain their important ideas. Due to the limited length of this article, for details on the algorithms, please refer to the original papers.

maps the input field through a neural network to the interested physical field and this mapping is achieved through composite functions, which is different from the current the predominance of additive functions for approximating mappings in traditional PDEs solver. The reason why deep learning is effective in solving PDEs boils down to the following points:

- Powerful approximation capability of neural networks: Under the condition of monotonically bounded activation functions, with a certain distribution of hidden layer neurons (at least one hidden layer), any continuous function can be approximated [62, 63]. Therefore, for some field functions with drastic changes, AI4PDEs can theoretically select the best basis functions automatically through the powerful fitting capabilities of neural networks, thereby avoiding the need to design shape functions like finite elements, which undoubtedly reduces the intellectual cost [31]. Additionally, neural networks can also fit any continuous operator [39], with DeepONet proposed by Lu et al. [34] utilizing this theory. This theory [39] provides a good theoretical prospect for the current hot research topic of operator learning.

- Direct integration of data and physical laws: AI4PDEs can utilize the powerful fitting capabilities of neural networks to learn the abstract rules behind data and physical equations, especially for complex physical processes where the physical laws are not very clear, and where the data volume is not large. An approximate solution can be obtained based on the physical equation and then adjusted with a small amount of data to achieve a solution closer to the real laws compared to traditional numerical methods [38]. Additionally, for domains with large data volumes and clear physical laws, operator learning can be used to pre-train the data and then fine-tune it according to the physical equation, significantly improving the speed of traditional numerical methods because the initial solution obtained by operator learning needs fewer iteration steps compared to a random initial solution by traditional numerical methods. This direction is currently one of the most cutting-edge in AI4PDEs [42, 41, 37].

- Precision and convenience of derivatives by automatic differentiation: This is because backpropagation uses the method of analytic gradient flow to calculate gradients, rather than using numerical methods, thereby obtaining precise gradient values. The automatic differentiation has been efficiently implemented in deep learning frameworks, making backpropagation very easy, thus AI4PDEs are simple in computation format and conducive to updates and iterations [58]. Although the mathematical format of AI4PDEs is not simple, thanks to the current optimization frameworks, such as Pytorch and Tensorflow, these frameworks facilitate the programming of AI4PDEs [64, 65], and the core solver usually does not exceed a hundred lines of code. Additionally, due to the wide applicability and adaptability of these frameworks, it reduces the need for people to be deeply familiar with traditional algorithms, especially inverse problem [10].

Due to the powerful fitting capabilities of neural networks as trial functions, the new paradigm of integrating data and equations, coupled with convenient code implementation, these unique advantages make AI4PDEs important and highly promising in solving mechanical PDEs, i.e., computational mechanics. AI4PDEs represent a class of numerical algorithms independent of finite elements, providing another possibility for computational mechanics. AI4PDEs have two important modules, the physical and data-driven modules, which we will introduce separately.

AI4PDEs' physical module is based on the idea of PINNs, with the two mainstream methods being the strong form and energy form of PINNs, both of which essentially use neural networks to replace the approximate functions in the weighted residual method. The strong form of PINNs often writes the domain equation, boundary conditions, and initial conditions into the loss function through hyperparameter-weighted residuals for optimization, different choices of approximate functions (trial functions), and weight functions (test functions) result in different numerical methods of PINNs strong form, similar to the research idea of different numerical methods arising from different trial and test functions in finite elements. PINNs' strong form often does not rely on numerical integration, and since all PDEs have a weighted residual form, the strong form of PINNs is general, almost any PDEs can be solved. However, PINNs' strong form has numerous hyperparameters, often requiring tuning, and when facing specific problems, the optimal hyperparameters are often unknown. Another important method of PINNs is the energy form, which essentially uses the variational principle, converting the strong form into an integral functional. The advantage of PINNs energy form is the dramatic reduction in

hyperparameters, often higher accuracy and efficiency (the variational principle has fewer derivative orders). The drawback of PINNs energy form is also very obvious: it is highly dependent on the choice of numerical integration scheme, also, not all PDEs have a well-behaved extremum variational principle, so its generality is not as good as PINNs strong form.

AI4PDEs' data-driven module is based on the recently developed operator learning, which is actually a special kind of pure data-driven algorithm. Using operator learning to learn the mapping of PDEs families, this mapping can be understood as the same type of PDEs under different parameters. The reason why it is possible to use neural networks to fit PDEs solutions is because we can understand PDEs as a kind of implicit mapping, which maps boundary conditions, geometric shapes, material fields to the solution fields. However, the mapping is very complex and difficult to write out in an analytic explicit form, therefore, it might be possible to use neural networks to learn the mapping. If successful, then inputting different boundary conditions, geometric shapes, and material fields can quickly obtain the needed field variables (such as displacement field), and the high-precision data obtained using traditional methods is the premise for learning PDEs' implicit mapping. Some recent works hope to combine physical equations into the operator [42, 41, 37, 66], which can use the physical equation to fine-tune the initial solution given by operator learning, thereby achieving lower computational cost compared to purely physics-based models. The current direction is still explored and has great research prospects, but the key is how to better combine prior knowledge and data together to better train the model.

Considering that the physical and data modules are the two most important parts of AI4PDEs, this chapter will review the AI4PDEs physical module (PINNs: Physics-informed neural networks), including the strong form and energy form. In addition, the AI4PDEs data module will be reviewed, including operator learning and the integration of data and physical modules (PINO: Physics-informed neural operators).

### *2.1. PINNs: Physics-informed neural networks*

#### *2.1.1. Strong form*

PINNs consist of two main aspects:

- The first is the use of neural networks to replace the approximate functions. The core of finite element in computational mechanics involves the selection of trial functions and test functions, and different selections form various types of finite element methods. PINNs mainly change shape function of FEM to neural networks in the trial function. The solution procedures of FEM and PINNs are fundamentally different. PINNs solve a non-convex optimization problem, whereas FEM addresses a linearized system of equations.

- The second aspect is the adoption of convenient automatic differentiation technology from artificial intelligence algorithm frameworks in the optimization algorithm to implement the physical equation loss function formed by PDEs. Using PDEs to construct the loss function essentially restricts the optimization space of neural networks, thereby obtaining an approximate solution space [22].

The idea of PINNs is very simple and straightforward, fundamentally introducing pre-known knowledge into the neural network, reducing the optimization space, and thereby improving accuracy and efficiency. The idea of combining traditional knowledge into algorithms has been very common in traditional machine learning, for example, Lauer et al. (2008) [67] embedded the prior knowledge of functions and their derivatives as constraints into support vector regression (SVR: Support Vector Regression) to reduce approximation errors.

PINNs were first introduced in the strong form, as shown in Fig. 3. The strong form of PINNs is an approximation function using neural networks in the weighted residual method. Considering the specific PDE, the original PDEs is:

$$\begin{cases} \text{Domain PDEs: } \boldsymbol{L}(\boldsymbol{u}(\boldsymbol{x},t)) = \boldsymbol{f}(\boldsymbol{x},t) & \forall(\boldsymbol{x},t) \in \Omega \bigtimes (0,T] \\ \text{Boundary condition: } \boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{h}(\boldsymbol{x},t) & \forall(\boldsymbol{x},t) \in \partial\Omega \bigtimes (0,T] \\ \text{Initial condition: } \boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{x}) & \forall(\boldsymbol{x},t) \in \Omega \bigtimes \{t=0\} \end{cases} . \tag{1}$$

We change the original equation into residual form, divided into three parts:

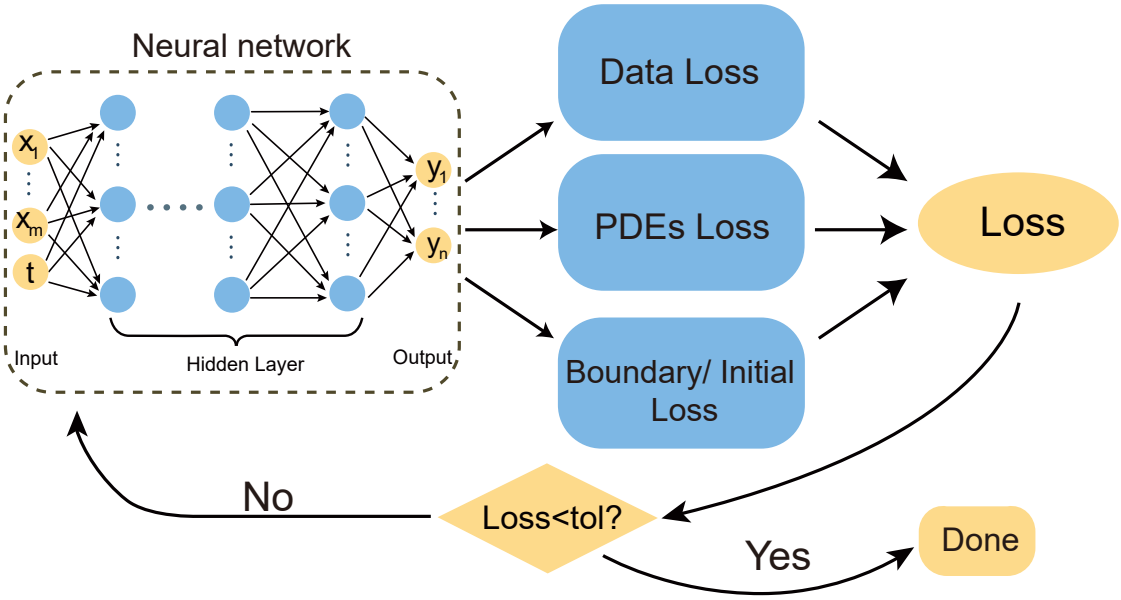


**Fig. 3.** AI for PDEs method: Schematic of PINNs strong form [9].

$$\begin{cases} \text{Domain PDEs residual: } \boldsymbol{r}_d(\tilde{\boldsymbol{u}}(\boldsymbol{x},t)) = \boldsymbol{L}(\tilde{\boldsymbol{u}}(\boldsymbol{x},t)) - \boldsymbol{f}(\boldsymbol{x},t) & \forall(\boldsymbol{x},t) \in \Omega \bigtimes (0,T] \\ \text{Boundary condition residual: } \boldsymbol{r}_b(\boldsymbol{x},t) = \tilde{\boldsymbol{u}}(\boldsymbol{x},t) - \boldsymbol{h}(\boldsymbol{x},t) & \forall(\boldsymbol{x},t) \in \partial\Omega \bigtimes (0,T] \\ \text{Initial condition residual: } \boldsymbol{r}_{ini}(\boldsymbol{x}) = \tilde{\boldsymbol{u}}(\boldsymbol{x},t=0) - \boldsymbol{g}(\boldsymbol{x}) & \forall(\boldsymbol{x},t) \in \Omega \bigtimes \{t=0\} \end{cases}, \tag{2}$$

where $\boldsymbol{L}$ is the differential operator, $\boldsymbol{f}$ is the non-homogeneous term of the PDEs, $\Omega$ is the spatial domain, $\partial\Omega$ is the boundary, $\boldsymbol{x}$ is the spatial coordinate, $t$ is time. $\boldsymbol{u}$ is the field of interest approximated by the neural network, and controlled by the parameters of the neural network. The neural network is denoted as $\boldsymbol{u}(\boldsymbol{x},t;\boldsymbol{W})$, where $\boldsymbol{W}$ represents the parameters of the neural network. $\boldsymbol{h}$ and $\boldsymbol{g}$ are the conditions that must be satisfied by the boundary conditions and initial conditions, respectively, which can often be analytically represented as functions $\boldsymbol{h}(\boldsymbol{x},t)$ and $\boldsymbol{g}(\boldsymbol{x})$, respectively. $\boldsymbol{r}_d$, $\boldsymbol{r}_b$, and $\boldsymbol{r}_{ini}$ are the domain residual, boundary condition residual, and initial condition residual, respectively.

The above residuals $\boldsymbol{r}_d$, $\boldsymbol{r}_b$, and $\boldsymbol{r}_{ini}$ are numerically summed using weight functions:

$$\begin{cases} \text{Domain PDEs residual weighted integral: } \boldsymbol{R}_i^d = \sum_{I=1}^{N_d} \boldsymbol{v}_i^d(\boldsymbol{x}_I,t_I) \odot \boldsymbol{r}_d(\tilde{\boldsymbol{u}}(\boldsymbol{x}_I,t_I)) & \forall(\boldsymbol{x},t) \in \Omega \bigtimes (0,T] \\ \text{Boundary condition residual weighted integral: } \boldsymbol{R}_i^b = \sum_{I=1}^{N_b} \boldsymbol{v}_i^b(\boldsymbol{x}_I,t_I) \odot \boldsymbol{r}_b(\tilde{\boldsymbol{u}}(\boldsymbol{x}_I,t_I)) & \forall(\boldsymbol{x},t) \in \partial\Omega \bigtimes (0,T] \\ \text{Initial condition residual weighted integral: } \boldsymbol{R}_i^{ini} = \sum_{I=1}^{N_{ini}} \boldsymbol{v}_i^{ini}(\boldsymbol{x}_I) \odot \boldsymbol{r}_{ini}(\tilde{\boldsymbol{u}}(\boldsymbol{x}_I)) & \forall(\boldsymbol{x},t) \in \Omega \bigtimes \{t=0\} \end{cases}, \tag{3}$$

where $\boldsymbol{v}_i^d$, $\boldsymbol{v}_i^b$, and $\boldsymbol{v}_i^{ini}$ are the domain, boundary, and initial weight functions, respectively, whose dimensions correspond one-to-one with $\boldsymbol{r}_d$, $\boldsymbol{r}_b$, and $\boldsymbol{r}_{ini}$. $\boldsymbol{R}_i^d$, $\boldsymbol{R}_i^b$, and $\boldsymbol{R}_i^{ini}$ are the domain residual integral, boundary condition residual integral, and initial condition residual integral, respectively. $\odot$ is the element-wise operation, meaning corresponding elements are multiplied without changing the tensor's shape. $N_d$, $N_b$, and $N_{ini}$ are the total number of numerical summation points in the domain, boundary, and initial areas, respectively. All of the above integral forms of residuals are weighted summed to form the loss function of PINNs in the strong form:

**Table 1**
Current status of PINNs strong form methods

| Methods | Trial function: subdomains | Test function: subdomains | Trial function: type | Test function: type | Method of Imposing Essential Boundary |
|---|---|---|---|---|---|
| PINNs [9] | No (Global) | No (Global) | Fully Connected | Dirac delta | Penalty Function |
| DGM [68] | No (Global) | No (Global) | Fully Connected | Least Squares | Penalty Function |
| VPINNs [69] | No (Global) | No (Global) | Fully Connected | Mixed Weight Functions | Penalty Function |
| PIELM [70] | Yes (Subdomains) | No (Global) | ELM[71] | Dirac delta | Penalty Function |
| cPINN [72] | Yes (Subdomains) | No (Global) | Fully Connected | Dirac delta | Penalty Function |
| XPINN [73] | Yes (Subdomains) | No (Global) | Fully Connected | Dirac delta | Penalty Function |
| hp-VPINNs [17] | No (Global) | Yes (Subdomains) | Fully Connected | Orthogonal Polynomials | Penalty Function |
| PhyGeoNet [74] | No (Global) | No (Global) | Convolutional Network (CNN) | Dirac delta | Strictly Enforced |
| PIGCN [75] | No (Global) | No (Global) | Graph Convolution (GCN) | Trial Function Space | Strictly Enforced |
| SPINN [76] | No (Global) | No (Global) | Radial Basis Function | Trial Function Space | Penalty Function |
| BINN [77] | No (Global) | No (Global) | Fully Connected | Fundamental Solution of PDEs | Boundary Integral Terms |
| KINN [78] | No (Global) | No (Global) | Kolmogorov Arnold Network [79] | Dirac delta | Penalty Function |

$$
\begin{aligned}
\mathcal{L} &= \lambda_{pdes}\mathcal{L}_{pdes} + \lambda_b\mathcal{L}_b + \lambda_{data}\mathcal{L}_{data} \\
\mathcal{L}_{pdes} &= \sum_{j=1}^{S^d} \boldsymbol{\beta}_j^d \cdot (\boldsymbol{R}_j^d)^2 \\
\mathcal{L}_b &= [\sum_{j=1}^{S^b} \boldsymbol{\beta}_j^b \cdot (\boldsymbol{R}_j^b)^2 + \sum_{j=1}^{S^{ini}} \boldsymbol{\beta}_j^{ini} \cdot (\boldsymbol{R}_j^i)^2] \\
\mathcal{L}_{data} &= \sum_{I=1}^{N_{data}} [\tilde{\boldsymbol{u}}(\boldsymbol{x}_I, t_I) - \bar{\boldsymbol{u}}(\boldsymbol{x}_I, t_I)]^2,
\end{aligned}
\tag{4}
$$

where $S^d$, $S^b$, and $S^{ini}$ are the number of weight functions in the domain, boundary, and initial conditions, respectively; $\boldsymbol{\beta}_j^d$, $\boldsymbol{\beta}_j^b$, and $\boldsymbol{\beta}_j^{ini}$ are the weights of the domain, boundary, and initial condition residual integrals, respectively. $\lambda_{pdes}$, $\lambda_b$, and $\lambda_{data}$ are the weights of the PDEs, boundary (initial), and data loss, respectively. $N_{data}$ is the number of existing high-precision data points. The training process involves adjusting the trainable weights of the neural network to minimize the loss function.

All important methods of PINNs in the strong form are almost always developed around Eq. (3) and Eq. (4). This article summarizes the existing important methods of PINNs in strong form from the perspectives of whether the trial functions and test functions are partitioned, types of trial functions and test functions, and the method of imposing essential boundary conditions, as shown in Table 1, where partitioning refers to dividing different areas into different neural networks.

The strong form of PINNs fundamentally uses a collocation method, where the approximate function is replaced with a neural network. PINNs in the strong form can be applied not only to solve forward problems of PDEs but also to determine unknown coefficients in inverse problems. In inverse problems, some data combined with a data-driven approach is often required, and the loss function includes a norm of the error between predictions and true values, as shown in Eq. (4) for $\mathcal{L}_{data}$. Unknown parameters of the inverse problem, such as elastic modulus and Poisson's ratio, are treated as trainable variables to be optimized. By solving for the gradients of the optimization variables in inverse problems, the parameters needed for the inverse problems are inferred to solve the inverse problem. It is noteworthy that in PINNs, the optimization variables of inverse problems are optimized using $\mathcal{L}_{pdes}$. The optimization of inverse problems in PINNs can also be split into two parts: initially fitting the data with a neural network without considering the physical equations, and after the fit, optimizing the variables needed for the inverse problem using the PDEs loss. However, in traditional PINNs, the loss from data and PDEs is optimized together through hyperparameters. It is important to note that whether these are split or not, the mathematical optimization goals are the same, essentially solving the same problem but with different optimization strategies. Moreover, traditional PINNs in strong form solving inverse problems (optimizing PDEs loss and data loss together) generally do not manage to make both $\mathcal{L}_{pdes}$ and $\mathcal{L}_{data}$ zero simultaneously; the optimization process is a trade-off between $\mathcal{L}_{pdes}$ and $\mathcal{L}_{data}$, often resulting in $\mathcal{L}_{data}$ reaching zero faster, while $\mathcal{L}_{pdes}$ tends to optimize more slowly. Lu et al. (2021) [80] proposed the HPINN algorithm, which handles constraints using the augmented Lagrangian method [81], thereby enhancing

the accuracy of solving inverse problems.

The strong form of PINNs does not require labeled data when addressing forward problems, although having labeled data can increase the efficiency and accuracy of the algorithm. The strong form of PINNs only requires spatial coordinates to use the powerful approximation capabilities of neural networks to map spacetime fields to the solution space [9]. From the perspective of computational mechanics, it is a type of trial function represented by a fully connected neural network and a Dirac delta mesh-free method for test functions. In their original paper [9], the boundary initial conditions were added by a soft constraint method using penalty functions, which involves adjusting extra hyperparameters. Around the same time, Sirignano et al. (2018) [68] introduced the DGM, which is very similar to PINNs, except that DGM does not address inverse problems. Additionally, DGM proposed using Monte Carlo algorithms instead of AD algorithms for derivative approximation, a method that can reduce the computational demands of high-dimensional problems [82], and DGM mathematically proved the convergence of PINNs in solving quasi-linear parabolic equations. Subsequently, Kharazmi et al. (2019) [69] drew on traditional finite element and numerical analysis ideas to propose VPINNs, a variational form of the weighted residual method that changes the test functions in PINNs from Dirac delta to polynomials and trigonometric functions, thus extending the weighted residual method of PINNs strong formulation. However, VPINNs has limitations; if the test functions use Legendre orthogonal polynomials, higher-order orthogonal polynomials that satisfy the loss function to zero must exist. Given the strong fitting capabilities of neural networks, there's no reason why the network would not fit higher-order orthogonal polynomials, thereby causing non-uniqueness in the solution [83]. Later, the same authors [17], improved VPINNs and further introduced hp-VPINNs, which are based on VPINNs but involve partitioning, meaning different areas have different test functions. This method found that orthogonal polynomials have high accuracy in this form, and it also compared different orders of variational weak forms. Jagtap et al. (2020) proposed cPINN [72] and XPINNs [73], which partition the global domain like finite elements into different areas, each using a different neural network as the trial function approximation. However, special handling is required at the interfaces, where continuity conditions for the interface and gradient continuity conditions (related to the highest derivative of the PDEs) must be added to the loss function. For second-order mechanical PDEs, continuity conditions for displacement and force must be added as interface loss function terms. cPINN is suitable for handling problems with drastic changes, using deeper neural networks where changes are large and shallow networks where changes are mild. Shukla et al. [84] integrated parallel algorithms into cPINN and XPINNs, enhancing the efficiency of the partitioned PINNs algorithms.

Most of the work discussed revolves around whether trial functions and test functions are partitioned and the types of test functions. Additionally, some work focuses on changing the type of network used for the trial functions, essentially modifying the trial function's approximation. Dwivedi et. al (2020) [70] used Extreme Learning Machines (ELM) for PINNs in strong form. ELMs are a type of single hidden layer fully connected neural network, where the single hidden layer is obtained through traditional nonlinear mapping followed by a linear transformation to produce the final output. Notably, unlike traditional FNNs, the nonlinear transformation of the single hidden layer in ELM is randomly initialized and requires no training; ELM only trains the final linear transformation. This brings many benefits, as the optimization in ELM is not iterative but can be accomplished through direct methods to obtain the weights of the linear transformation, significantly reducing the computational load. Therefore, Dwivedi et. al (2020) [70] proposed using ELM to replace the approximation function in the strong form of PINNs, calling this method PIELM. Since ELM sacrifices the expressive power of FNNs for efficiency, PIELM often does not perform well in cases of complex exact solutions. In such cases, partitioning ideas are needed to enhance the approximation capability of PIELM, thus improving its accuracy. Undoubtedly, since PIELM determines the weights of the neural network through direct solution methods, it is more efficient than most works in PINNs based on iterative algorithms. Additionally, some researchers have used new neural network structures to replace fully connected neural networks, such as Gao et al. (2021) [74], who initially proposed PhyGeoNet, which essentially replaces fully connected neural networks with convolutional neural networks. The backward derivation in PhyGeoNet is achieved using fixed-weight convolutional kernels, and PhyGeoNet uses the idea of parametric transformation to solve problems in irregular domains that are challenging for physically based convolutional neural networks. The boundary conditions in this algorithm are strongly enforced. Later, Gao et al. (2022) [75] continued to improve PhyGeoNet and introduced PIGCN, which uses graph neural networks as approximation functions, and the loss function is constructed in weak form.

Ramabathiran et al. (2021) [76] proposed SPINN, which uses radial basis functions (RBF) as approximation functions, aiming to reduce the complexity of the neural network. Bai et al. (2023) [85] further validated the use of RBF networks for solving nonlinear equations through systematic verification via NTK (Neural Tangent Kernel) theory [86]. Sun et al. (2023) [77] first combined boundary elements with PINNs methods, using the residual of the boundary integral equation as the loss function. Overall, the experience of changing the network structure of PINNs strong form is that smaller neural networks often lack expressive power, but more complex neural networks are typically difficult to train, so choosing the best neural network structure requires considering the complexity of the problem [22], similar to choosing element types in finite elements, as different element types in finite elements essentially choose different approximation functions, and PINNs choosing network structures essentially also choose different approximation functions. Wang et al. (2024) proposed KINN [78], which uses KAN proposed by Liu et al. (2024) [79] to replace MLP in PINNs, utilizing the strong form, energy form, and inverse form for the first time. Compared with traditional PINNs, the convergence speed and the accuracy of KINN were greatly improved.

In time-dependent PINNs strong form algorithms, there are two ways to solve time-dependent problems: expanding by one dimension or using a time-marching scheme (commonly FDM) [87]. Mattey et al. (2022) [14] proposed bc-PINN, which enhances the computational efficiency and accuracy of PINNs in solving strong nonlinear and high-order dynamic equations. Meng et al. (2020) [88] proposed PPINN, which uses a small network to enhance the efficiency of PINNs algorithms for long-term PDEs problems. Additionally, for data of different accuracies, Meng et al. (2020) [89] proposed MPINN, which combines high-precision data with low-precision data. In the area of stochastic PDEs, Yang et al. (2020) [28] proposed PIGAN, which combines physical equations with GANs (Generative Adversarial Networks) to solve the direct and inverse problems of stochastic PDEs (SPDEs).

There are several PINN libraries available, such as DeepXDE based on TensorFlow [64], SciANN based on Keras [90], NeuralPDE based on Julia [91], and SimNet [92], which facilitate the application of PINNs in solving PDEs and are suitable for both academia and industry. These libraries facilitate the application of PINNs in solving PDEs.

Table 2 provides some AI for PDEs codes in computational mechanics:

*2.1.2. Energy form*

DEM is a deep learning method for solving PDEs, designed based on the principle of least action in physics. When applied to a mechanical system, the principle of least action is manifested as Hamilton's principle:

$$\mathcal{H} = \int_{t_0}^{t_1} L dt \tag{5}$$

where $\mathcal{H}$ represents the Hamiltonian action, which is the spacetime functional of the entire system. $t_0$ and $t_1$ represent the initial and final times respectively. $L$ is the Lagrangian function:

$$\begin{aligned} L &= T - V \\ T &= \int_\Omega \frac{1}{2}\rho \boldsymbol{v} \cdot \boldsymbol{v} d\Omega \\ V &= \int_\Omega \Psi d\Omega - \int_\Omega \boldsymbol{f} \cdot \boldsymbol{u} d\Omega - \int_{\Gamma^t} \bar{\boldsymbol{t}} \cdot \boldsymbol{u} d\Gamma \end{aligned} \tag{6}$$

where $T$ and $V$ represent the kinetic energy and potential energy of the system, respectively. For an elastic system, the expressions for kinetic energy $T$ and potential energy $V$ are given as:

$$\begin{aligned} T &= \int_\Omega \frac{1}{2}\rho \boldsymbol{v} \cdot \boldsymbol{v} d\Omega \\ V &= \int_\Omega \Psi d\Omega - \int_\Omega \boldsymbol{f} \cdot \boldsymbol{u} d\Omega - \int_{\Gamma^t} \bar{\boldsymbol{t}} \cdot \boldsymbol{u} d\Gamma \end{aligned} \tag{7}$$

where $\rho$, $\boldsymbol{v}$, and $\Psi$ is the density, velocity and the strain energy density respectively. $\mathbf{f}$ and $\bar{\mathbf{t}}$ are the body force and the surface force vector at the traction boundary condition $\Gamma^t$, respectively. In DEM, the entire functional

**Table 2**
The codes of AI for PDEs in computational mechanics

| | | Reference | The link of code | Brief Description of Method |
|---|---|---|---|---|
| PINNs | The strong form of PINNs | [9] | https://github.com/maziarraissi/PINNs | The original code |
| | | [17] | https://github.com/ehsankharazmi/hp-VPINNs | Variational PINNs |
| | | [72] | https://github.com/AmeyaJagtap/Conservative_PINNs | PINNs with subdomains |
| | | [78] | https://github.com/yizheng-wang | KINN |
| | | [80] | https://github.com/lululxvi/hpinn | Inverse design |
| | | [88] | https://github.com/XuhuiM/PPINN | Time-dependent PDEs |
| | | [26] | https://github.com/sciann/sciann-applications/tree/master/SciANN-SolidMechanics-BCs | Solid mechanics |
| | | [93] | https://github.com/Raocp/PINN-elastodynamics | Elastodynamics |
| | | [94] | https://github.com/imcs-compsim/pinns_for_comp_mech | Contact mechanics |
| | | [95] | https://github.com/Jianxun-Wang/LabelFree-DNN-Surrogate | Incompressible NS |
| | | [96] | https://github.com/PredictiveIntelligenceLab/DeepStefan | Multiphase and moving boundary problems |
| | | [97] | https://github.com/Raocp/PINN-laminar-flow | Incompressible laminar flows |
| | | [98] | https://github.com/shengzesnail/AIV_MAOAC | Blood flow |
| | DEM: Deep Energy Method | [33] | https://github.com/ISM-Weimar/DeepEnergyMethods | The original code |
| | | [32] | https://github.com/yizheng-wang | DCEM |
| | | [31] | https://github.com/yizheng-wang | DEM with subdomains |
| | | [99, 100] | https://github.com/weili101/DeepPlates | Kirchhoff plate |
| | | [101, 102] | https://github.com/somdattagoswami/IGAPack-PhaseField | Phase field for fracture mechanics |
| | | [103] | https://github.com/MinhNguyenIKM/dem_hyperelasticity | Hyperelasticity |
| | | [104] | https://github.com/JinshuaiBai/RPIM_NNS | Hyperelasticity |
| | | [105] | https://github.com/Jasiuk-Research-Group/DEM_for_J2_plasticity | J2 elastoplasticity |
| | | [106] | https://github.com/MinhNguyenIKM/parametric-deep-energy-method | Parametric DEM |
| | | [107] | https://github.com/Jasiuk-Research-Group/DeepEnergy-TopOpt | Topology optimization |
| | | [108] | https://github.com/sbuoso/Cardio-PINN/ | Heart |
| | | [109] | https://github.com/dodaltuin/soft-tissue-pignn | Soft tissue |
| Operator learning | DeepONet | [34] | https://github.com/lululxvi/deeponet | The original code |
| | | [110] | https://github.com/lu-group/deeponet-extrapolation | DeepONet with physics or sparse observations |
| | | [111] | https://github.com/Jasiuk-Research-Group/ResUNet-DeepONet-Plasticity | DeepONet for elastoplastic |
| | FNO | [35] | https://github.com/neuraloperator/neuraloperator | The original code |
| | | [112] | https://github.com/eshaghi-ms/DeepNetBeam | Functionally Graded Porous Beams |
| | | [113] | https://github.com/gegewen/ufno | Multiphase flow |
| | | [114] | https://github.com/neuraloperator/Geo-FNO | FNO for general geometries |
| PINO | | [37] | https://github.com/neuraloperator/physics_informed | PINNs with FNO |
| | | [42] | https://github.com/PredictiveIntelligenceLab/Physics-informed-DeepONets | PINNs with DeepONet |
| | | [41] | https://github.com/somdattagoswami/IGAPack-PhaseField | DEM with DeepONet |
| | | [32] | https://github.com/yizheng-wang | DCEM with DeepONet |

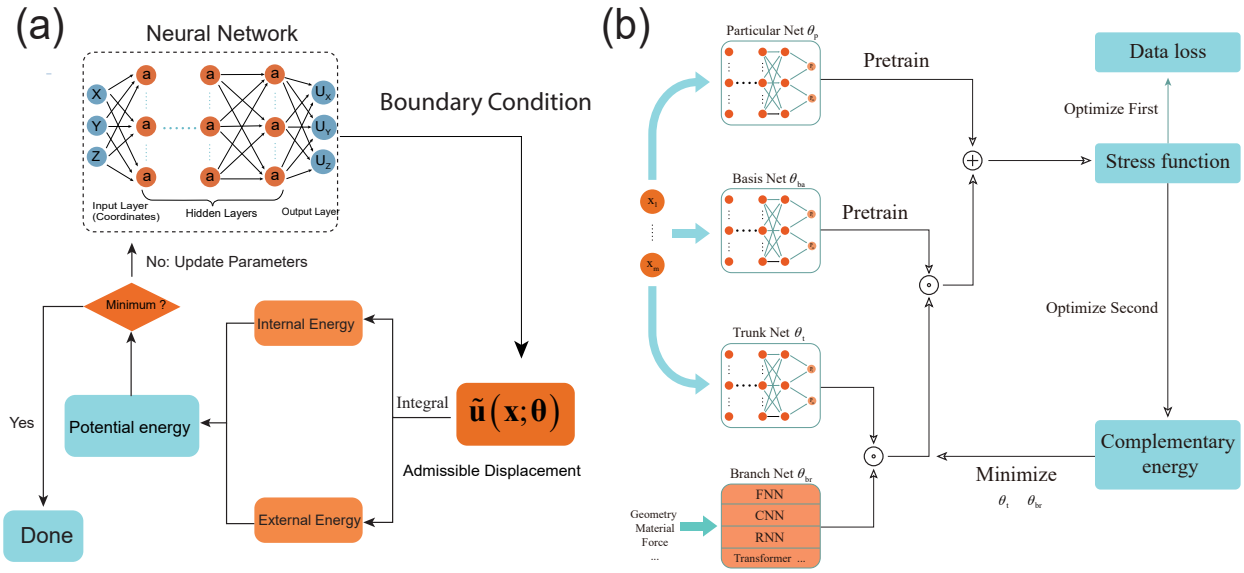


**Fig. 4.** AI for PDEs method: Schematic of (a) DEM based on the principle of minimum potential energy [33], (b) DCEM based on the principle of minimum complementary energy [32].

$\mathcal{H}$ is optimized as the loss function. If we consider a static system where $T = 0$, DEM uses the potential energy $V$ as the loss function as illustrated in Fig. 4. The mathematical form of the DEM loss function is given by:

$$\mathcal{L} = \mathcal{H} = \int_{t_0}^{t_1} L dt = \int_{\Omega} \Psi d\Omega - \int_{\Omega} \boldsymbol{f} \cdot \boldsymbol{u} d\Omega - \int_{\Gamma^t} \bar{\boldsymbol{t}} \cdot \boldsymbol{u} d\Gamma \tag{8}$$

The principle of DEM is to use the minimum potential energy principle under the condition that the displacement field satisfies the essential displacement conditions, i.e., to select the displacement field that minimizes the total potential energy $\mathcal{L}$ among all admissible displacements. DEM is similar to the weak form energy variational method of finite elements, but DEM directly optimizes energy without needing to express virtual work in the corresponding weak form like finite elements. The admissible displacement field is constructed through a neural network, but this admissible displacement field often needs to be pre-built to naturally satisfy the enforced displacement boundary conditions. The construction method for the admissible displacement field $\tilde{\boldsymbol{u}}(\boldsymbol{x})$ is commonly adopted as:

$$\tilde{\boldsymbol{u}}(\boldsymbol{x}) = \boldsymbol{P}(\boldsymbol{x}) + \boldsymbol{D}(\boldsymbol{x}) * \boldsymbol{u}(\boldsymbol{x}; \boldsymbol{\theta}), \tag{9}$$

where $\boldsymbol{P}(\mathbf{x})$ is a boundary network that satisfies the PDEs displacement boundary conditions, i.e., if the coordinate point falls exactly on the essential boundary condition, it outputs the value of the essential boundary condition; $\boldsymbol{D}(\mathbf{x})$ is a distance network, indicating the shortest distance from the current coordinate point to the nearest essential boundary condition location; $\boldsymbol{u}(\mathbf{x}; \boldsymbol{\theta})$ is a generalized network that needs to be trained by substituting it into the minimum potential energy loss function. The idea of the admissible displacement field to satisfy the essential boundary conditions also exists in meshless methods [115]. Note that boundary and distance networks do not necessarily need to be neural networks; choosing any other fitting function, such as Radial Basis Function (RBF) [31] is also possible. We discuss the construction method of the admissible displacement field in detail in Section 3.4.

In the study of PINNs energy form, we provide a review of the status of PINNs energy form from four perspectives: whether subdomains are partitioned (whether trial functions are partitioned), types of trial functions, how essential boundary conditions are applied, and what kind of energy principle is used, as shown in Table 3 on the current state of research on PINNs energy form. Sheng et al. (2021) [116] extended Deep Ritz and proposed PFNN, using distance, boundary, and generalized networks to construct admissible displacement

**Table 3**
Current status of PINNs energy form methods

| Methods | Trial function: subdomains | Trial function: type | Method of Imposing Essential Boundary | Energy Principle |
|---|---|---|---|---|
| Deep Ritz [25] | No (Global) | Fully Connected | Penalty Function | Variational Principle |
| DEM [33] | No (Global) | Fully Connected | distance, boundary, and generalized networks | Minimum Potential Energy |
| PFNN [116] | No (Global) | Fully Connected | distance, boundary, and generalized networks | Variational Principle |
| mDEM [117] | No (Global) | Fully Connected | distance, boundary, and generalized networks | Mixed Form |
| P-DEM [106] | No (Global) | Fully Connected | Penalty Function | Minimum Potential Energy |
| CENN [31] | Yes (Subdomain) | Fully Connected | distance, boundary, and generalized networks | Minimum Potential Energy |
| GCN-DEM [118] | No (Global) | Graph Convolutional (GCN) | distance, boundary, and generalized networks | Minimum Potential Energy |
| PIRBN [85] | No (Global) | Radial Basis Function | distance, boundary, and generalized networks | Minimum Potential Energy |
| DCEM [32] | No (Global) | Fully Connected | distance, boundary, and generalized networks | Minimum Complementary Energy |
| KINN [78] | No (Global) | Kolmogorov Arnold Network [79] | distance, boundary, and generalized networks | Minimum Potential Energy |

fields. Fuhg et al. (2022) [117] proposed a classical mixed formulation mDEM, which integrates strong form, energy form, and constitutive equations into the loss function for optimization. Nguyen-Thanh et al. (2021) [106] expanded DEM with P-DEM, borrowing the idea of parametric elements from finite elements to address complex geometric boundary problems. Wang et al. (2022) [31] expanded DEM and proposed CENN, proposing a subdomain form of the deep energy method. In terms of neural network structures, He et al. (2023) [118] introduced GCN-DEM, replacing the original fully connected neural network with a graph convolutional neural network. As current PINNs energy forms are based on the minimum potential energy principle, Wang et al. (2023) [32] first used the minimum complementary energy principle to propose DCEM, which parallels the deep energy method DEM, and combined operator learning with physical equations, making it an important complementary form to DEM. Wang et al. (2024) proposed KINN [78], which utilizes KAN, as proposed by Liu et al. (2024) [79], to replace MLP in PINNs, and employed strong form, energy form, and inverse form for the first time.

### *2.2. Operator learning*

Neural operators learn the mapping between different functions, which has extensive applications in science and engineering [36]. For instance, the input function could be the boundary conditions of a PDEs, and the output is the function field of interest. Although functions are infinite-dimensional in mathematical terms, in practice, we often input discretized data on a finite mesh to approximate continuous functions.

Unlike previous data-driven methods based on neural networks, operator learning is discretization-invariance, characterized by:

- The ability to input at any resolution.
- The capability to output at any location.
- Convergence of results as the mesh is refined.

Traditional neural network-based data-driven algorithms are often related to the degree of discretization; once the resolution of the input-output functions is changed, the model usually needs to be retrained. However, operator learning does not need to restart training for data of different resolutions due to the characteristic of the discretization-invariance. Additionally, not all fitting algorithms can be termed as neural operators. Neural operators not only need to satisfy discretization-invariance but also meet the requirements of the generalized approximation [39]. It can be said that neural operators are a superior class of algorithms within operator learning, such as the Fourier Neural Operator (FNO) [35]. Operator learning can be applied not only in deterministic models but also in probabilistic models, like the diffusion model [119], to learn the probability density in function spaces, such as solving stochastic PDEs.

The design philosophy of operator learning algorithms is shown in Fig. 5a, with the core being the kernel integral method in the neural operator layers:

$$\boldsymbol{H}^{(t)}(\boldsymbol{v}^{(t)};\boldsymbol{\theta}_k^{(t)})(\boldsymbol{x}) = \int_{D^{(t)}} \boldsymbol{k}^{(t)}(\boldsymbol{x},\boldsymbol{y};\boldsymbol{\theta}_k^{(t)})\boldsymbol{v}^{(t)}(\boldsymbol{y})d\boldsymbol{y}, \tag{10}$$

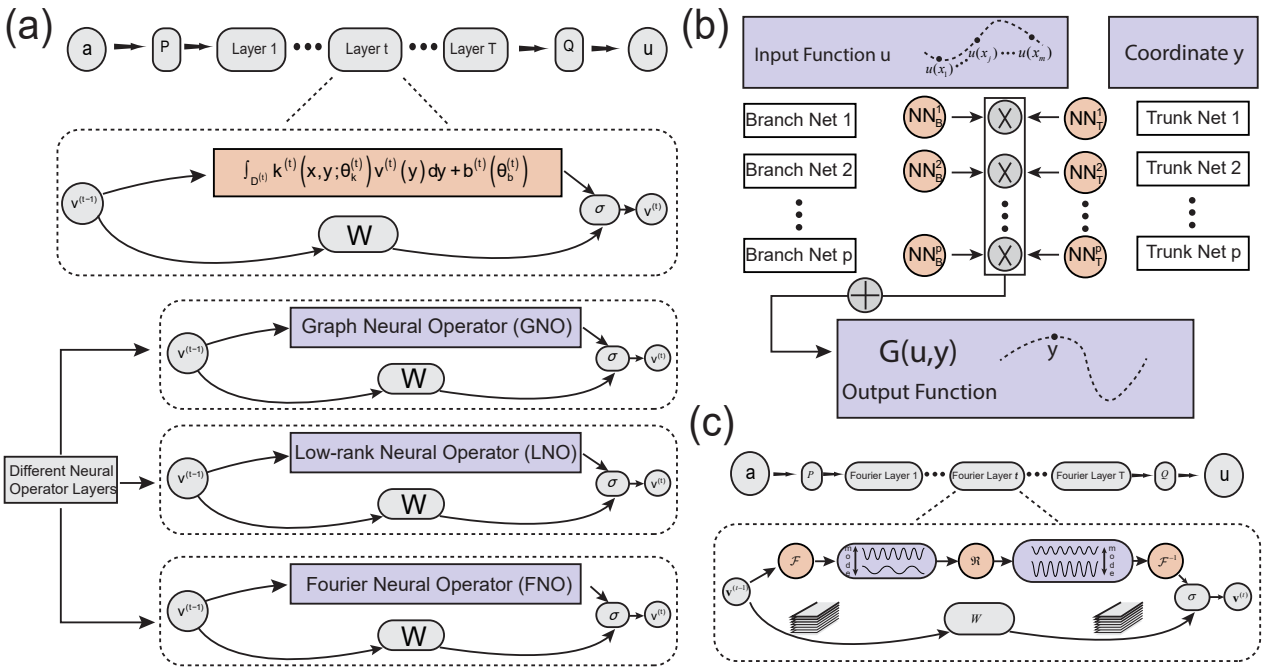


**Fig. 5.** AI for PDEs Method: Schematic of Operator Learning. (a) Neural Operator Layer Structure: Graph Neural Operator (GNO) [120], Local Neural Operator (LNO), and Fourier Neural Operator (FNO) [35] can serve as the core of the neural operator architecture [36]. (b) Details of DeepONet [34]. (c) Details of FNO [35].

where $\boldsymbol{v}^{(t)}(\boldsymbol{y})$ is the input to the neural operator layer; $\boldsymbol{k}^{(t)}(\boldsymbol{x},\boldsymbol{y})$ is the kernel function, similar to the Green's function in linear PDEs, and $\boldsymbol{k}^{(t)}(\boldsymbol{x},\boldsymbol{y})$ is approximated using neural networks, where $\boldsymbol{\theta}_k^{(t)}$ are the parameters for approximating the kernel function. The reason it's a kernel integral method is that the kernel operator $\boldsymbol{H}^{(t)}$ acts on the function $\boldsymbol{v}^{(t)}$, and through integration, $\boldsymbol{H}^{(t)}$ transforms $\boldsymbol{v}^{(t)}$ into another class of functions $\boldsymbol{H}^{(t)}(\boldsymbol{v}^{(t)})(\boldsymbol{x})$. $\boldsymbol{H}^{(t)}$ is similar to integral transformations in mathematical equations. In terms of methods of mathematical physics, $\boldsymbol{H}^{(t)}(\boldsymbol{v}^{(t)})(\boldsymbol{x})$ is referred to the image function, $\boldsymbol{v}^{(t)}(\boldsymbol{y})$ is the original function, and $\boldsymbol{k}^{(t)}(\boldsymbol{x},\boldsymbol{y})$ is the kernel function. Note that the $t$ index indicates the $t$-th neural operator layer. Hence, the overall computation process of the operator learning is:

$$\begin{aligned}\boldsymbol{G}_\theta(\boldsymbol{a}(\boldsymbol{x});\boldsymbol{\theta}) =& \boldsymbol{Q}(\boldsymbol{v}^{(T+1)};\boldsymbol{\theta}_Q)\circ\sigma[\boldsymbol{H}^{(T)}(\boldsymbol{v}^{(T)};\boldsymbol{\theta}_k^{(T)})+\boldsymbol{W}^{(T)}(\boldsymbol{v}^{(T)};\boldsymbol{\theta}_w^{(T)})+\boldsymbol{b}^{(T)}(\boldsymbol{\theta}_b^{(T)})]\circ\cdots \\ &\circ\sigma[\boldsymbol{H}^{(1)}(\boldsymbol{v}^{(1)};\boldsymbol{\theta}_k^{(1)})+\boldsymbol{W}^{(1)}(\boldsymbol{v}^{(1)};\boldsymbol{\theta}_w^{(1)})+\boldsymbol{b}^{(1)}(\boldsymbol{\theta}_b^{(1)})]\circ\boldsymbol{P}(\boldsymbol{a}(\boldsymbol{x});\boldsymbol{\theta}_P)\end{aligned}, \tag{11}$$

where $\boldsymbol{P}$ and $\boldsymbol{Q}$ are respectively for elevating and reducing dimensions of the original data, with corresponding learnable parameters $\boldsymbol{\theta}_P$ and $\boldsymbol{\theta}_Q$. $\boldsymbol{H}^{(t)}$, $\boldsymbol{W}^{(t)}$, and $\boldsymbol{b}^{(t)}$ are respectively the kernel integral, linear transformation, and bias, with the corresponding learnable parameters being $\boldsymbol{\theta}_k^{(t)}$, $\boldsymbol{\theta}_w^{(t)}$, and $\boldsymbol{\theta}_b^{(t)}$. Note that these learnable parameters are generally approximated using neural networks.

Different choices of kernel functions constitute various operator learning algorithms, hence designing the kernel function is central to operator learning. When specifically forming a Fourier transform, it results in the Fourier Neural Operator (FNO). There is theoretical evidence that Eq. (11) meets the general approximation requirements [121, 122], hence operator learning algorithms designed according to Eq. (11) possess excellent convergence properties. DeepONet [34] also meets the general approximation requirements [39], essentially designed through the theory proposed by [39]. Although the FNO and DeepONet are designed differently, they both exhibit excellent convergence properties. The two most typical algorithms in operator learning are DeepONet and FNO, thus this paper will focus on these two algorithms below.

*2.2.1. DeepONet*

DeepONet is a neural operator learning algorithm based on neural networks. Its mathematical structure is analogous to a Taylor series:

$$f(\boldsymbol{u})(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i(\boldsymbol{u})\phi_i(\boldsymbol{x}). \tag{12}$$

As illustrated in Fig. 5b, the Trunk net employs neural networks to approximate the basis functions $\phi_i(\boldsymbol{x})$. Trunk net shares a common idea with PINNs, where neural networks fit from the coordinate space to the target function. However, in DeepONet, it is the basis functions $\phi_i(\boldsymbol{x})$ that are fitted by the Trunk net, rather than the target functions. The coefficients before the basis functions $\alpha_i(\boldsymbol{u})$ are fitted by the Branch net. It is noteworthy that inputting the same function into the Branch net produces fixed weights $\alpha_i(\boldsymbol{u})$, aligning with the concept of function approximation in numerical analysis. Unlike traditional function approximation algorithms that pre-select basis functions, DeepONet uses neural networks to adaptively select suitable basis functions and weights based on data.

Since the Trunk network is similar to PINNs, we can apply automatic differentiation algorithms in PINNs to construct partial differential operators and obtain new loss functions (strong or energy form). If the Branch network is fixed, DeepONet can be reduced to PINNs.

DeepONet's network structure is constructed according to the theory of universal operator approximation from [39]. The mathematical form of this approximation theory in [39] is:

$$\mid G(\boldsymbol{u},\boldsymbol{y}) - \sum_{k=1}^{p}\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} w_{ij}^k \boldsymbol{u}(\boldsymbol{x}_j) + b_i^k\right)\sigma(\boldsymbol{W}^k\cdot\boldsymbol{y} + \boldsymbol{B}^k) \mid < \epsilon. \tag{13}$$

DeepONet replaces the terms in the above equation with neural networks:

$$\begin{aligned} NN_B^k(\boldsymbol{u}(\boldsymbol{x});\boldsymbol{\theta}_{NN}^B) &= \sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} w_{ij}^k \boldsymbol{u}(\boldsymbol{x}_j) + b_i^k\right) \\ NN_T^k(\boldsymbol{y};\boldsymbol{\theta}_{NN}^T) &= \sigma(\boldsymbol{W}^k\cdot\boldsymbol{y} + \boldsymbol{B}^k) \\ NN_O(\boldsymbol{y};\boldsymbol{u}) &= \sum_{k=1}^{p} NN_B^k(\boldsymbol{u}(\boldsymbol{x});\boldsymbol{\theta}_{NN}^B)\cdot NN_T^k(\boldsymbol{y};\boldsymbol{\theta}_{NN}^T) \end{aligned}, \tag{14}$$

where $NN_B^k$, $NN_T^k$, and $NN_O$ are respectively the Branch network, Trunk network, and the output of DeepONet. $\boldsymbol{u}(\boldsymbol{x})$ can be approximated from sensors $\boldsymbol{x}$ (discrete points approximating a continuous function). For instance, if the input $\boldsymbol{u}(\boldsymbol{x})$ of $NN_B$ is the gravitational potential energy of a three-dimensional cubic material over $[0,1]^3$, then the gravitational potential energy field within $[0,1]^3$ might be sampled at 0.01 intervals taking $101^3$ points. Data structure of $\boldsymbol{u}(\boldsymbol{x})$ can vary, such as data structures similar to images. Therefore, we can use network structures for the Branch network that are appropriate for the type of data, such as CNNs for local features similar to image data, and RNNs for time-related features. It is important that the network structure of the Branch network must consider the particularities of the problem to choose the most appropriate network structure. Thus, $\boldsymbol{u}(\boldsymbol{x})$ determines the output of the Branch network in DeepONet, which is related to the weights $\alpha_i(\boldsymbol{u})$ in Eq. (12) and is independent of the coordinates of interest. Similarly, the coordinates fully determine the output of the Trunk network, precisely defining the basis functions $\phi_i(\boldsymbol{x})$ in Eq. (12). Therefore, DeepONet shares many similarities with traditional function approximation. It can be said that DeepONet can adaptively learn basis functions and weights based on data. For more details and extensions on DeepONet, see [34].

*2.2.2. FNO: Fourier neural operator*

The Fourier Neural Operator (FNO) is an operator learning algorithm introduced by Li et al. (2020) [35], which has received considerable attention. The initially proposed FNO is a completely data-driven model.

Even earlier than FNO, the same first author, Li et al. [120], used graph neural networks to learn relationships between field variables (GNO), but FNO demonstrated better performance than GNO. The essence of FNO involves integrating Fourier transforms into operator learning, representing Eq. (10) with a Fourier transform:

$$\boldsymbol{H}^{(t)}(\boldsymbol{v}^{(t)};\boldsymbol{\theta}_k^{(t)})(\boldsymbol{x}) = \int_{D^{(t)}} \boldsymbol{k}^{(t)}(\boldsymbol{x},\boldsymbol{y};\boldsymbol{\theta}_k^{(t)})\boldsymbol{v}^{(t)}(\boldsymbol{y})d\boldsymbol{y} = \mathcal{F}^{-1}\circ\Re\circ\mathcal{F}(\boldsymbol{v}^{(t)}(\boldsymbol{x})), \tag{15}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transform and its inverse, respectively, $\Re$ is a linear transformation. To clarify the process of the FNO algorithm, we combine the framework and examples of the FNO algorithm. The framework of the FNO algorithm, as shown in Fig. 5c, considers the one-dimensional Burgers equation, commonly used as a benchmark due to its space-time and nonlinear characteristics:

$$\begin{cases} \text{Domain control equation}: \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \quad x \in [-1,1],\ t \geq 0 \\ \text{Boundary conditions}: u(-1,t) = u(1,t) = 0 \\ \text{Initial condition:}\ u(x,0) = u_0(x) \end{cases}, \tag{16}$$

where $u$ is the field variable to be solved, $\nu$ is the viscosity coefficient, and the boundary conditions are fixed. Once the initial condition $u_0$ is determined, the solution $u$ can be uniquely determined. Thus, in this problem, the input is the initial condition $u_0 \in \mathbb{R}^{d_i}$ (i.e., $a(x)$ in [35]), and the output is $u(x,t_1)$ at a future time $t_1$. The FNO algorithm process first uses a fully connected neural network $\boldsymbol{P}$ to elevate the dimension of the input $a(x)$ to $d_c$ (i.e., $P:\mathbb{R}\to\mathbb{R}^{d_c}$, $d_c$ similar to channels in computer vision):

$$\boldsymbol{v}^{(0)}(x) = \boldsymbol{P}(a(x)) \in \mathbb{R}^{d_i * d_c}. \tag{17}$$

Next, $v^{(0)}(x)$ is sent into Fourier layer, where each channel of $v^{(0)}(x)$ undergoes a Fourier transform, transforming into:

$$\boldsymbol{f}^{(1)}(x) = \mathcal{F}(\boldsymbol{v}^{(0)}(x)) \in \mathbb{C}^{d_i * d_c}. \tag{18}$$

High frequencies are filtered out, retaining low frequencies, and cutting off the frequency at $d_m (1 \leq d_m \leq d_i)$, thus the data structure transforms into $f^{(1)}(x) \in \mathbb{C}^{d_m \times d_c}$. A linear transformation $\Re$ is applied to each frequency of all channels:

$$\boldsymbol{f}_L^{(1)}(x) = \Re(\boldsymbol{f}^{(1)}(x)) \in \mathbb{C}^{d_m * d_c}. \tag{19}$$

Since different frequencies use different linear transformations, there are $d_m$ linear transformation matrices, and the linear transformation is from $d_c$ to $d_c$, so the linear transformation matrix is $\mathbb{R}^{d_c \times d_c \times d_m}$. After the linear transformation $\Re$, the data's dimensional structure is not changed. Since the next step involves Fourier inverse transformation, high frequencies removed by cutting are replaced by zeros. Fourier inverse transformation is performed on $\boldsymbol{f}_L^{(1)}(x)$:

$$\boldsymbol{v}_f^{(1)} = \mathcal{F}^{-1}(\boldsymbol{f}_L^{(1)}(x)) \in \mathbb{R}^{d_i * d_c}. \tag{20}$$

The advantage of the linear transformation $\Re$ is that it learns the mapping relationships between data in the Fourier frequency space while filtering out high frequencies. $\Re$ is a special form of regularization that reduces overfitting and enhances the model's generalization capability.

Furthermore, using the idea of residual networks from [123], another linear transformation is directly applied to the channels of $v^{(0)}(x)$, the same linear transformation for all different frequencies, only one linear transformation, i.e., $\boldsymbol{W} \in \mathbb{R}^{d_c \times d_c}$:

$$\boldsymbol{v}_L^{(1)} = \boldsymbol{W}(\boldsymbol{v}^{(0)}(x)) \in \mathbb{R}^{d_i * d_c}. \tag{21}$$

$\boldsymbol{W}$ uses the idea of residual networks to reintroduce high-frequency components filtered out by the Fourier transform back into the network for learning, preventing a decrease in network prediction performance due to the high frequencies filtered out by the Fourier transform. The results of the Fourier transform and the linear transformation are added together to obtain:

$$\boldsymbol{v}^{(1)}(x) = \sigma(\boldsymbol{v}_f^{(1)} + \boldsymbol{v}_L^{(1)} + \boldsymbol{b}^{(1)}) \in \mathbb{R}^{d_i * d_c}, \tag{22}$$

where $\boldsymbol{b}^{(1)}$ is the bias. Adding a nonlinear transformation $\sigma$ not only increases the network's expressive capability but also further enhances the high-frequency fitting capability. The Fourier layer is the computational core of FNO, and the above operations are repeated $T$ times to obtain $v^{(T)}(x)$. Finally, the dimension $d_c$ is reduced to the target dimension by $\boldsymbol{Q} : \mathbb{R}^{d_c} \to \mathbb{R}$.

The two major advantages of FNO are, first, since the Fourier transform filters out high-frequency modes, it can enhance the model's speed and generalization ability, reducing overfitting; second, discretization-invariance, since all operations in FNO are unrelated to the data's mesh and are based on point-wise operations, so it has the advantage of discretization-invariance. The initially proposed FNO has two major flaws. First, the Fourier Neural Operator (FNO) employs the fast Fourier transform, which requires the input data to be defined over a regular domain. Although it is feasible to enclose an irregular domain within a larger regular one, this approach often yields suboptimal results. To address this, Li et al. [114] introduced Geo-FNO, specifically designed to handle irregular domains effectively. Second, while FNO requires data points to be positioned on a uniform mesh, points on an uneven mesh must be converted into a uniform mesh using interpolation. Thus, the error of interpolation decreases the accuracy of FNO.

For the comparison between FNO and DeepONet, Lu et al. [18] systematically analyzed both methods. They introduced enhancements to FNO to handle mappings of different dimensionalities and complex geometries, while also improving DeepONet to accelerate training and enhance accuracy, supplemented by theoretical comparisons.

### *2.3. PINO: Physics-informed neural operator*

The concept of the Physics-Informed Neural Operator (PINO) was proposed by Z. Li et al. (2021) [37]. As shown by the black lines in Fig. 6, the idea is very concise: first, operator learning is trained using big data, then a good initial solution is provided on the test set using operator learning, which is then fine-tuned based on physical equations. Since the initial solution is not arbitrarily given but inferred from historical data, it only needs some adjustment to be fine-tuned to the true solution. Therefore, theoretically, the PINO algorithm not only possesses very high precision (since it is controlled by physical equations) but also leverages the efficiency advantage of operator learning. The above process involves using operator learning to provide an initial solution, followed by fine-tuning with PDEs. For example, [32] isolates the training of physical equations and data. The approach first uses purely data-driven training for operator learning and then fine-tunes with PDEs for specific problems. Additionally, PDEs and data can be combined together to train an operator constrained by physical equations, similar to the training mode of PINNs. For example, Wang et al. (2021) [42] proposed combining DeepONet with physical equations, and combined the data loss with PDEs loss together. Later, Goswami et al. (2022) [41] built on the idea proposed by [42] to predict crack propagation (by combining DeepONet with the phase field method of predicting crack path). Currently, there is no conclusion as to which method is better; both are still being explored. The advantage of isolating the training of PDEs and data is that it can avoid catastrophic forgetting and reduce hyperparameters, specifically those used to balance data and PDE losses. The downside is that it requires additional fine-tuning on new PDEs, thereby increasing the testing time. On the other hand, integrating PDEs and data during training has the advantage of high efficiency during the testing period, as PDEs are already incorporated into the operator during training. However, this undoubtedly increases the training time. Additionally, it introduces extra hyperparameters to balance the data and PDEs, and faces the risk of catastrophic forgetting [124]. This occurs because the PDEs under the current parameters can significantly lose the knowledge and performance of the PDEs under previous parameters, thereby increasing the training cost. Although incorporating PDEs to train neural operators increases computational cost, it remains an effective approach in scenarios where data is scarce but PDEs are available. Eshaghi et al. [125] proposed the Variational Physics-Informed Neural Operator (VINO). Unlike PINO, VINO employs the variational form of PDEs to train a fully Fourier Neural Operator. Additionally, VINO uses the shape functions in FEM to replace the automatic differentiation (AD) algorithm for computing differential operators, enhancing the stability of derivative computations. Since the variational form requires integration, the derivative computation using the shape functions in VINO further improves integration accuracy.

Another way of combining data and physical equations is for some complex physical processes [38], which refers to those that cannot be accurately described based on current understanding and can only be approximated by some physical equations. We can use PINNs to solve for an approximate solution (low-fidelity) based on

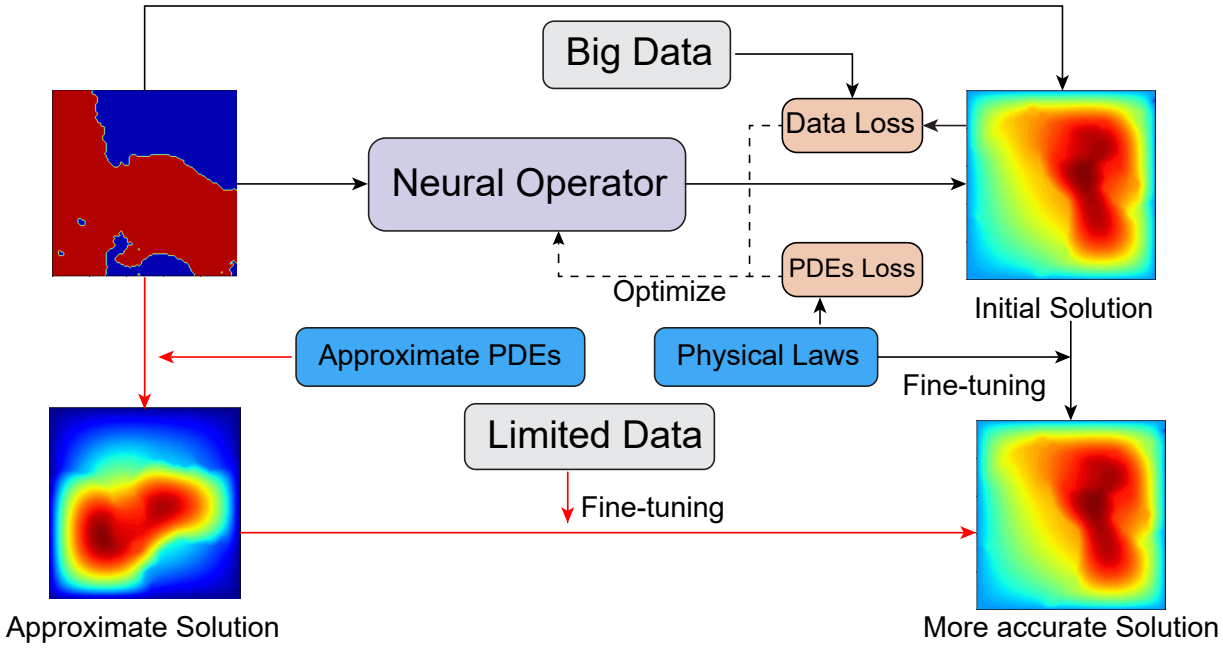


**Fig. 6.** AI for PDEs method: Schematic of Physics-Informed Neural Operator (PINO). There are two implementation approaches. The first, depicted by the black lines, is suitable for scenarios with precise physical equations. Initially, the neural operator is trained using extensive data and physical equations. The method involves using the neural operator to provide an initial solution and establishing a data loss with the data labels, followed by constructing a PDEs loss based on the physical equations. The combination of data loss and PDEs loss allows for more effective training of the neural operator compared to purely data-driven methods [37]. After training, the neural operator is fine-tuned with the physical equations of specific problems to provide highly accurate solutions. The second approach, depicted by the red lines, is suited for cases with only approximate physical equations, such as complex phenomena not well understood by humans. This method starts with an approximate solution derived from the approximate PDEs, which is then fine-tuned using limited data [38].

approximate physical equations and boundary conditions, and then adjust the deep parameters of the neural network according to the available data, thereby obtaining a data-based solution (high-fidelity) that does not require exact knowledge of the physical process. This allows us to fuse experimental data with approximate physical equations, as shown by red lines in Fig. 6. Although this method of combination currently does not use operator learning, it can be easily modified using operator learning algorithms when obtaining high-fidelity results in the future, which is a promising direction, specifically targeting those complex physical processes and combining operators with physical equations.

Given that the concept of Physical Neural Operator (PINO) was only recently introduced, numerous research challenges remain to be explored. Key questions include how to more effectively integrate physical equations into operator learning, how to handle noisy data, and how to perform learning with limited samples. In this paper, we focus on the first issue, as we have encountered significant difficulties in integrating physical equations into operator learning. Specifically, after training the Fourier Neural Operator (FNO) with large datasets, we observed a sudden increase in the loss function when incorporating physical equations via finite difference methods. This increase in loss reduces efficiency. In some extreme cases, the combination of data and PDEs may even be worse than using physical equations directly (without data). We hypothesize that this issue arises due to a non-overlapping optimization space between data loss and physical equation loss, which leads to difficulties in their integration. Despite these challenges, we believe that PINO holds substantial promise for academic research. Its theoretical foundation is both robust and promising, providing ample opportunities for exploration and innovation.

Since the PINO inherently combines operator learning with physical equations, it can be seen as encompassing both Physics-Informed Neural Networks (PINNs) and operator learning. If only physical equations are utilized, PINO reduces to PINNs; if solely data-driven methods are applied, PINO reduces to operator learning.

### *2.4. Summary*

This chapter introduced the methodologies of AI for PDEs, including PINNs, operator learning, and physics-informed neural operators. Specific algorithms have been reviewed in their respective sections. Here, we discuss notable points and potential directions for future research:

The advantage of the strong form of PINNs in solving high-dimensional problems arises not from the neural network itself but from the way the loss function is computed, specifically through Monte Carlo integration. Monte Carlo integration is a well-established area in applied statistics, and many concepts from this field can be borrowed to enhance PINNs, such as variance reduction techniques. These techniques involve altering the probability density of sampling points to reduce the variance of the Monte Carlo integral, which is crucial for the convergence speed of PINNs. The smaller the variance of the loss function, the faster the convergence of PINNs. The selection of collocation points has always been a core issue for PINNs. Therefore, combining Monte Carlo integration techniques from applied statistics with PINNs is essential for optimizing point selection and reducing variance, thereby accelerating convergence.

Although the energy form of PINNs generally has higher precision and efficiency than the strong form of PINNs algorithms and has fewer hyperparameters [100], not all PDEs have a corresponding energy form for PINNs, and the energy form has higher demands on energy integration. The energy form of PINNs requires a loss function that finds an extremum; if it is a stationary saddle point problem, it is difficult to compute the stationary saddle point with existing optimization algorithms, although some existing algorithms for solving saddle points, such as the Alternating Direction Method of Multipliers (ADMM), can be used. However, due to the highly non-convex nature of neural networks, there are many saddle points in the loss function space, so even if optimized to a saddle point, it cannot be guaranteed to be the true solution. This poses a great challenge to the saddle point problem of the energy form of PINNs. For example, if the famous generalized variational principle is used as the loss function (HW three-field variational principle and HR mixed two-field variational principle), the optimization is very difficult because the optimization of the generalized variational principle is a saddle point problem, and there are too many neural network saddle points. Thus, we don't know if the saddle point is the true solution of the generalized variational principle. Therefore, the limitations of the energy algorithm of PINNs are very obvious; it must be PDEs with very good properties ($\delta^2\mathcal{L} > 0$, where $\mathcal{L}$ is the functional) to have an energy algorithm for PINNs. Hence, the universality of the energy algorithm of PINNs is not as good as that of the strong form of PINNs. The future of PINNs will not only depend on researchers

who combine artificial intelligence and computational mechanics but will also be inseparable from experts in the field of artificial intelligence, especially optimization algorithms. Although there are already libraries for the strong form of PINNs, such as DeepXDE, there is currently a lack of libraries for the energy form of PINNs like DEM, and DEM programming is much more complicated than the strong form of PINNs, mainly involving the selection of integration schemes and the derivation of variational formats. Therefore, it is very necessary to write DEM libraries for AI for Mechanics.

Operator learning algorithms essentially learn the implicit mapping of PDE families through big data. The main operator learning algorithms currently are FNO and DeepONet, both supported by corresponding theories. The core of these algorithms is the kernel integral method, and different kernel integral schemes can give rise to various operator learning algorithms. While Fourier transformation is currently in use, other transformations such as Laplace transform and Z-transform can be considered special cases of kernel integration, leading to diverse neural operator algorithms. Additionally, DeepONet is based on the mathematical proof of approximating continuous operators by neural networks from 1995 [39], substituting this mathematical proof's convergence criteria with neural networks. We believe that other theories have proven the same thing, that neural networks can approximate any continuous operator. As a result, other theories can be exploited to create various algorithms similar to DeepONet in the future.

The algorithm based on the physics-informed neural operator is particularly interesting and has significant research and industrial application prospects. This is because this method can continually self-learn. The algorithm can become faster over time, and with gradually increasing data, it can train increasingly accurate operators. Essentially, the connection is universal; although the problems we are currently solving have not precisely appeared in the training set, they share similarities, and we can leverage existing data for unknown tests, and then correct them based on physical equations. The larger the dataset, the more accurate the initial solution predicted by operator learning will be, and thus the physical equation correction time will be shorter. Another combination method is for unknown phenomena, where an approximate equation can provide an approximate solution, which is then fine-tuned based on data, making it very suitable for complex phenomena, such as biomechanics.

In this chapter, we primarily introduced some methodologies of AI for PDEs. In the next chapter, we will introduce some theoretical research on AI for PDEs, focusing on PINNs.

## 3. AI for PDEs: Theoretical Research

### *3.1. Activation functions*

The choice of activation function affects the function representation of PINNs. Essentially, the activation function determines the degree of nonlinearity, thereby altering the neural network's fitting ability. The *tanh* function is the most commonly selected activation function for PINNs, though other non-linear activation functions are also viable. PINNS must avoid activation functions with discontinuous higher-order derivatives or those whose higher-order derivatives are identically equal to zero, because PDEs often contain higher-order derivatives. The chain rule results in an ineffective optimization when the derivative of the activation function is not satisfied with the above condition. Therefore, activation functions like ReLu, which have a second-order derivative of zero, should not be chosen in higher-order derivative PDEs, especially when the highest order of the derivatives in the PDEs is greater than or equal to 2. The above is achieved by specifically changing the type of activation function to alter the accuracy and efficiency of PINNs. However, Jagtap et al. [126] proposed a strategy applicable to any activation function, involving the multiplication of a trainable parameter $a$ after each neural network linear transformation layer, and the idea behind the strategy is similar to KAN [79]. This parameter alters the activation function shape to achieve faster convergence, expressed mathematically as:

$$u(x) = (L_k \circ \sigma \circ naL_{k-1} \circ \sigma \circ naL_{k-2} \cdots \circ \sigma \circ naL_1)(x), \tag{23}$$

where $n$ is a non-trainable parameter set to determine the basic shape of the activation function, $L_i$ (where i ranges from 1 to k) represents the linear mapping layers of the network, and $\sigma$ is the non-linear activation function. Later, Jagtap et al. [127] further refined the algorithm by independently adjusting each linear layer's $a$ and added a loss function term for $a$, aiming to increase $a$ to prevent gradient vanishing and thus further

speeding up convergence. Essentially, the concept behind adjusting the activation function shape resonates with regularization in machine learning.

### *3.2. Errors*

The errors in AI for PDEs primarily consist of four components: approximation error, optimization error, generalization error [34], and integral error.

Although neural networks have the capability to approximate any continuous function, a specific neural network structure must still be selected. Therefore, there will inevitably be some discrepancy between the solution space of the neural network and the actual solution space, which constitutes the approximation error. Optimization error refers to the error arising from the local optima due to the highly non-convex nature of neural networks. Generalization error is generally not specific to PINNs but is raised concerning operator learning because PINNs solve specific problems described by physical equations, whereas operator learning studies families of PDEs. Thus, operator learning deals with physical problems across different geometries, constitutive models, and boundary conditions, making generalization error a measure of how well operator learning approximates a family of PDEs. PINNs involve an integral error, which is determined by the number and distribution of collocation points. In the strong form of PINNs, integral error refers to the discrepancy between the integral of the least squares loss function and the exact integral. In the energy form of PINNs, integral error refers to the discrepancy between the integral over the energy functional and the exact energy functional. Most loss functions in PINNs typically use Monte Carlo integration for approximation. The expectation of Monte Carlo integration generally matches the exact value, and the variance measures the convergence efficiency of the integration. The expectation and variance of Monte Carlo integration are determined by the way collocation points are arranged. Therefore, researching how to reduce errors and enhance accuracy in PINNs by studying collocation methods is crucial, as these methods can reduce integral error.

Fig. 7 shows a visual representation for an intuitive understanding of errors in PINNs. For better comprehension, we compare PINNs with traditional computational mechanics methods. In traditional computational mechanics, such as finite element methods, the size of the function space for approximation is determined by the type and number of elements. In contrast, PINNs use neural networks as approximation functions, so the function space is significantly larger due to the universal approximation [128]. However, due to the optimization methods and the highly non-convex nature of neural networks, the full potential of PINNs' powerful fitting capabilities is not realized, leading to optimization error. Additionally, the integral error introduced by the collocation method determines the extent to which the PINNs' loss function approximates the true loss function, thus also limiting the approximation capability of PINNs. Finally, because the structure of the neural network is specifically given, the mathematical operations behind PINNs are very clear, meaning the approximation function space is already determined and the gap between the function space of NN and the exact solution introduces approximation error. From this perspective, the structure of the neural network plays a role similar to the element type in finite elements, as both determine the approximation function space. Displacement finite elements obtain nodal displacement values which is used to acquire the physical field of the entire domain through interpolation with shape functions, while PINNs simulate field variables by optimizing neural network parameters. Thus, the parameters of the neural network in PINNs are analogous to the nodal displacements in finite element methods.

To better understand the integral error in PINNs, we provide the following example to illustrate. Consider a simple ordinary differential equation example [83]:

$$\begin{cases} u^{''}(x) = 0 & x \in (-1,1) \\ u(-1) = 0 & ,u(1) = 1 \end{cases}. \tag{24}$$

If we select $N_u + 2$ points between -1 (inclusive) and 1 (inclusive), excluding $x = 0$, to construct the loss function for the strong form of PINNs, we observe an interesting phenomenon. The loss function is:

$$Loss = \frac{1}{N_u}\sum_{i=1}^{N_u} |u^{''}(x_i)|^2 + u(-1)^2 + [u(1) - 1]^2. \tag{25}$$

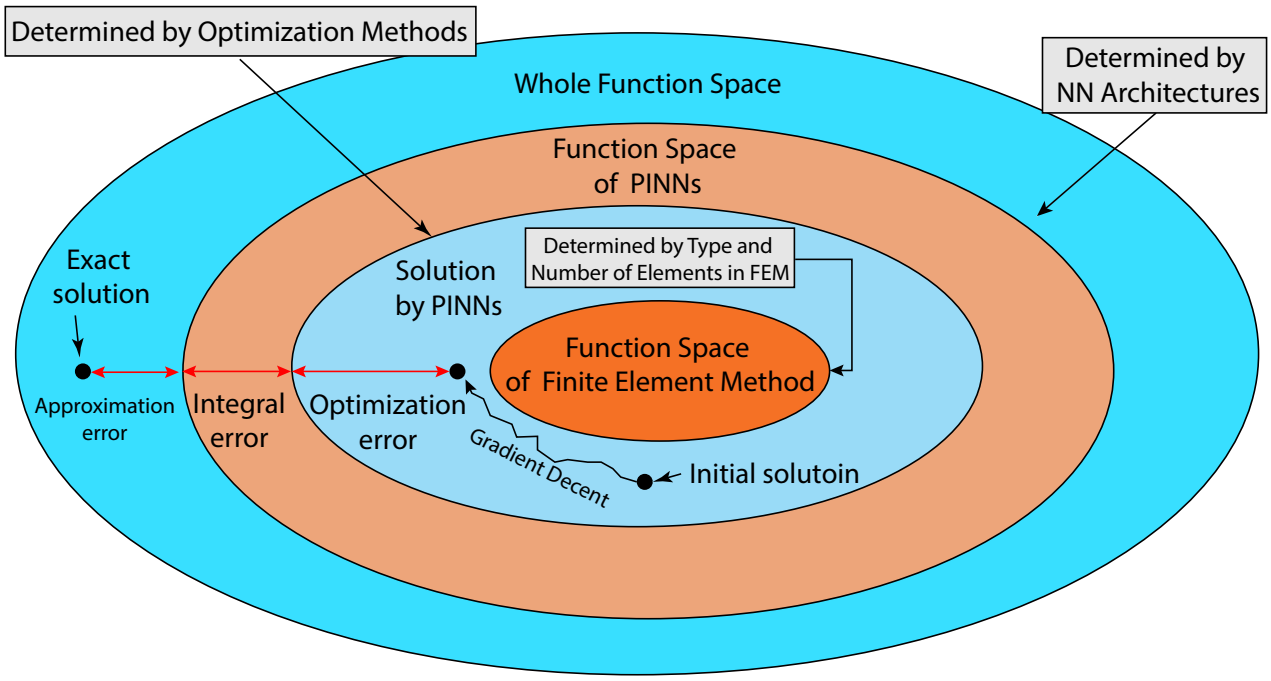


**Fig. 7.** Error comparison between PINNs and FEM.

As $N_u$ approaches infinity, it is easy to verify that the piece function in Eq. (26) can make the loss function in Eq. (25) zero.

$$u(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{26}$$

However, the exact solution to Eq. (24) is $u(x) = x + 1/2$. Due to the powerful fitting capabilities of neural networks, theoretically, exact solution and Eq. (26) both could potentially be optimized (in fact, many other solutions that satisfy the zero loss function also exist). The phenomenon of non-unique solutions is not only due to the powerful fitting capabilities of neural networks but also due to the integral error in PINNs. The neural network brings theoretical advantages of a large approximation function space but also introduces the drawback of non-uniqueness in numerical solutions (traditional methods are very robust and stable due to the limitations of approximation function space). Because the approximation function space of the neural network is very large, regularization techniques are crucial for AI for PDEs [22].

### *3.3. Weight Selection*

The optimization process of PINNs is a type of multi-task learning in machine learning, so the selection of weights among different loss functions is particularly important. Different choices of weights will result in different optimization outcomes, thus affecting the efficiency and accuracy of the PINNs algorithm. From an optimization perspective, since most optimization algorithms are based on gradient descent, the algorithm will prioritize components with high optimization value in the loss function (optimization value refers to the trend in the magnitude change of the loss function), potentially neglecting other components [23]. The multi-task optimization of the loss function can be expressed mathematically as:

$$\mathcal{L} = \lambda_r \mathcal{L}_r + \lambda_i \mathcal{L}_i + \lambda_b \mathcal{L}_b + \lambda_d \mathcal{L}_d, \tag{27}$$

where $\{\lambda_r, L_r\}$, $\{\lambda_i, L_i\}$, $\{\lambda_b, L_b\}$, and $\{\lambda_d, L_d\}$ respectively represent the weights and loss functions for internal PDEs, initial conditions, boundary conditions, and data. Our task is to optimize $L_r$, $L_i$, $L_b$, and $L_d$ to their

**Table 4**
Current research on PINNs weight selection

| Reference | Brief Description of Method |
|---|---|
| [13] | Selecting weights by comparing the gradients of different components of the loss function |
| [129] | Using NTK theory to select weights |
| [86] | Identifying the frequency tendencies of neural networks using NTK theory |
| [130] | Modifying the PINNs loss function into a saddle point problem |
| [131] | Modifying the loss function using maximum likelihood estimation |

ideal states, hence research on weight selection in PINNs mainly explores how to adjust the magnitudes of $\{\lambda_r, \lambda_i, \lambda_b, \lambda_d\}$, i.e., the different methods of weight selection.

Here we summarize representative studies on PINNs weight selection: Wang et al. (2021) [13] proposed selecting the weights for other loss function terms based on the internal PDEs loss function, where the method of selecting weights is as follows:

$$\hat{\lambda}_o = \frac{max\{|\nabla_{\boldsymbol{\theta}} \mathcal{L}_r(\boldsymbol{\theta})|\}}{\overline{|\nabla_{\boldsymbol{\theta}} \mathcal{L}_o(\boldsymbol{\theta})|}}. \tag{28}$$

The gradient of the internal loss function with respect to the optimizable parameters is calculated, and we select the maximum value. Then, the average value of the gradients for other components (subscript $o$, which can be chosen as initial conditions, boundary conditions, and data loss functions) is calculated. The gradients of $\mathcal{L}_r$ and $\mathcal{L}_o$ are then divided to serve as the weight for that loss function component. This is followed by a moving average, i.e., the previously obtained weight $\lambda_o$ and the newly obtained weight $\hat{\lambda}_o$ are averaged:

$$\lambda_o = (1-\alpha)\lambda_o + \alpha\hat{\lambda}_o. \tag{29}$$

This moving average idea is similar to the well-known optimization algorithms RMSprop and Adam in machine learning. RMSprop and Adam consider historical information to aid optimization, as a part of the randomness of gradient descent can be mitigated through moving averages. Finally, this weight in Eq. (29) is used for optimization. After a certain number of optimizations, the weight is recalculated, and this process is repeated.

Subsequently, Wang et al. (2022) [129] incorporated the Neural Tangent Kernel (NTK) theory proposed by Jacot et al. (2018) [132] into PINNs. NTK theory is used to analyze the optimization and generalization properties of deep neural networks during training. Naturally, this theory is applicable to PINNs, because PINNs' approximation function is a deep neural network. As a result, Wang et al. (2022) [129] proposed adjusting weights based on NTK theory:

$$\begin{aligned} \lambda_b &= \frac{\sum_{i=1}^{N_r+N_b} \lambda_i^{rb}(n)}{\sum_{i=1}^{N_b} \lambda_i^{b}(n)} = \frac{Tr(K^{rb})}{Tr(K^b)} \\ \lambda_r &= \frac{\sum_{i=1}^{N_r+N_b} \lambda_i^{rb}(n)}{\sum_{i=1}^{N_r} \lambda_i^{r}(n)} = \frac{Tr(K^{rb})}{Tr(K^r)} \end{aligned}, \tag{30}$$

where $\lambda_i^{rb}(n)$, $\lambda_i^{b}(n)$, and $\lambda_i^{r}(n)$ are the eigenvalues of the NTK matrices $K^{rb}$, $K^b$, and $K^r$ respectively at the $n$-th iteration. $\lambda_b$ and $\lambda_r$ are the hyperparameters corresponding to the loss functions. $N_r$ and $N_b$ are the total numbers of collocation points within the domain and on the boundary, respectively. The NTK matrix is specifically represented as:

$$\begin{aligned} \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x'}) &= \lim_{\eta \to 0} \frac{f(\boldsymbol{x}; \boldsymbol{\theta} + \eta \frac{\partial f}{\partial \boldsymbol{\theta}}|_{\boldsymbol{x}'}) - f(\boldsymbol{x}; \boldsymbol{\theta})}{\eta} \\ &= \lim_{\eta \to 0} \frac{f(\boldsymbol{x}; \boldsymbol{\theta}) + \eta(\frac{\partial f}{\partial \boldsymbol{\theta}}|_{\boldsymbol{x}'}) \circ (\frac{\partial f}{\partial \boldsymbol{\theta}}|_{\boldsymbol{x}}) + O(\eta \frac{\partial f}{\partial \boldsymbol{\theta}}|_{\boldsymbol{x}'}) - f(\boldsymbol{x}; \boldsymbol{\theta})}{\eta} = (\frac{\partial f}{\partial \boldsymbol{\theta}}|_{\boldsymbol{x}}) \circ (\frac{\partial f}{\partial \boldsymbol{\theta}}|_{\boldsymbol{x}'}) \end{aligned}, \tag{31}$$

where $f$ is the function to be approximated, such as the displacement field in mechanics. The elements of the NTK matrix are calculated by cross-computing for all training points; if there are $N$ points, the NTK matrix will be an $N \times N$ positive semi-definite matrix. The specific calculation of the NTK matrix can be done using the last term in Eq. (31), i.e., calculating the gradients at points $x^{'}$ and then performing a dot product operation between gradients of $x$ and $x^{'}$. The limit form of Eq. (31) is the algorithmic significance of the NTK matrix, i.e., performing gradient descent at point $x^{'}$, and due to the change in parameters, calculating the change at another point $x$. Since the parameters of a neural network are interconnected, the NTK matrix can be used to measure the convergence of the neural network algorithm (here, the greater the change in the function, the faster the algorithm converges).

In summary, the NTK matrix is related to convergence speed, so the larger the entries of this matrix, the more attention the optimizer will pay to this part, and the faster the convergence will be. Therefore, balancing different loss function weights, using the NTK matrix is very reasonable. It is also possible to measure the rate of convergence using the norm of the matrix instead of its trace. The advantage of using the trace is that only the diagonal elements of the matrix need to be calculated, so the computational load is smaller (calculating the NTK matrix is computationally intensive). Subsequently, Wang et al. (2021) [86] used NTK to identify the spectral tendencies of neural networks, i.e., fully connected neural networks approximate first the low frequencies and ignore high frequencies [133], for example, for the target function:

$$u(x,t) = \sin(\pi x)\cos(10\pi t) + \sin(2\pi x)\cos(20\pi t). \tag{32}$$

When using fully connected networks to fit the above target function, it is found that it cannot be completely fitted, and the error is very large. Therefore, the same problem occurs in PINNs, considering that the true solution of PDEs is this type of low-frequency carrying high-frequency function (common in multi-scale PDEs). The solution is to introduce sin and cos transformations on the input coordinates, and the specific transformations of sin and cos are determined based on NTK theory [86]. It is worth noting that instead of directly adjusting the hyperparameters in front of the loss function, the input spatiotemporal coordinates are transformed into the frequency domain.

Additionally, Liu et al. (2021) [130] proposed modifying Eq. (27) into a saddle surface form of the loss function using the constrained Lagrange multiplier method:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha}) = \lambda_r(\boldsymbol{\alpha})\mathcal{L}_r(\boldsymbol{\theta}) + \lambda_i(\boldsymbol{\alpha})\mathcal{L}_i(\boldsymbol{\theta}) + \lambda_b(\boldsymbol{\alpha})\mathcal{L}_b(\boldsymbol{\theta}) + \lambda_d(\boldsymbol{\alpha})\mathcal{L}_d(\boldsymbol{\theta}), \tag{33}$$

where $\boldsymbol{\theta}$ is the optimization parameter of the neural network, and $\boldsymbol{\alpha} = [\ \alpha_r \quad \alpha_i \quad \alpha_b \quad \alpha_d\ ]$ are trainable parameters which are combined with softmax to decide the weights

$$\lambda_j = \frac{exp(\alpha_j)}{exp(\alpha_r) + exp(\alpha_i) + exp(\alpha_b) + exp(\alpha_d)}, j \in \{r, i, b, d\} \tag{34}$$

Comparing Eq. (33) with Eq. (27), it is not difficult to see that Eq. (33) is a generalized form of Eq. (27), that is, Eq. (33) means optimizing the most difficult hyperparameter $\lambda$ group. So, by optimizing Eq. (33), theoretically, better results than the traditional PINNs loss function of Eq. (27) can be achieved. Another interesting approach is that Xu et al. (2023) [131] proposed constructing a loss function using the maximum likelihood function. This idea was initially proposed by Kendall et al. [134] to use uncertainty estimation to solve the problem of setting weights for multi-task optimization. Because the loss function in PINNs consists of many terms, it is a multi-objective optimization problem in machine learning. We make a basic assumption that the model prediction $\hat{y}$ and the label $\bar{y}$ follow a Gaussian distribution, i.e.:

$$p(\bar{y}|\hat{y}) = N(u = \hat{y}, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(\bar{y} - \hat{y})^2}{2\sigma^2}). \tag{35}$$

This formula means that when we know the output of the model, we believe that the true solution is distributed around the model output in a Gaussian distribution centered at the model output, but we do not know the

variance $\sigma^2$ in advance; it needs to be learned from the data. Then we consider the outputs of multiple models, because it is multi-task learning:

$$p(\bar{y}_1, \bar{y}_2, \cdots, \bar{y}_k|\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_k) = \prod_{i=1}^{k} p(\bar{y}_i|\hat{y}_i) = \prod_{i=1}^{k} N(u = \hat{y}_i, \sigma_i^2) \tag{36}$$

.

Here, the assumption of independent and identically distributed is used, then using the basic concepts of statistics, the maximum likelihood function is established:

$$\begin{aligned} L &= \ln p(\bar{y}_1, \bar{y}_2, \cdots, \bar{y}_k|\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_k) = \sum_{i=1}^{k} \ln N(u = \hat{y}_i, \sigma_i^2) \\ &= \sum_{i=1}^{k} \ln[\frac{1}{\sqrt{2\pi}\sigma_i} exp(-\frac{(\bar{y}_i - \hat{y}_i)^2}{2\sigma_i^2})] = \sum_{i=1}^{k} [-\frac{L_i}{2\sigma_i^2} - \ln \sigma_i - \ln(\sqrt{2\pi})] \end{aligned}, \tag{37}$$

where $L_i = (\bar{y}_i - \hat{y}_i)^2$. We ignore the constant term $\ln(\sqrt{2\pi})$, because the constant term does not affect the derivative of $L$ in Eq. (37). Since our goal is to make the likelihood function $L$ as large as possible, the loss function can be set to minimize the negative likelihood function:

$$\mathcal{L} = \sum_{i=1}^{k} [\frac{L_i}{2\sigma_i^2} + \frac{1}{2} \ln \sigma_i^2]. \tag{38}$$

This allows us to modify the loss function in PINNs to Eq. (38), where $1/(2\sigma_i^2)$ is corresponding the weight $\lambda_i$ in Eq. (27), and $L_i$ is different loss functions in PINNs (PDEs, boundary, initial conditions, data loss). It is worth noting that we need to optimize $\sigma_i$ as a training parameter along with the neural network to let the data choose the best weight. Results show that the loss function set by Eq. (38) converges faster and more accurately, because Eq. (38) is constructed using the statistical maximum likelihood estimation, by introducing an uncertainty parameter $\sigma$ for each task (loss function), allowing the model to adaptively adjust the relative importance of different task losses. Tasks with high uncertainty (mathematically meaning the variance $\sigma_i^2$ is large) are automatically given a lower weight, this is because high uncertainty means that the model's prediction for that task is less reliable, thus it should occupy a smaller proportion in the total loss. This mechanism allows the model to dynamically balance multiple tasks during training, rather than relying on statical weights in advance.

### *3.4. Distance Networks*

In addition to the constraints of the PDEs, there are also the constraints of boundary conditions. The simplest implementation is through penalty factors to satisfy the constraints of boundary conditions:

$$Loss_b = \beta \sum_{i=1}^{N} |NN(\boldsymbol{x}_i; \boldsymbol{\theta}) - \bar{u}(\boldsymbol{x}_i)|^2 \tag{39}$$

where $Loss_b$ is the boundary loss, $NN(\boldsymbol{x}_i; \boldsymbol{\theta})$ is the neural network of the variable of interest, $\boldsymbol{\theta}$ is the trainable parameter of the neural network, and $\bar{u}(\boldsymbol{x}_i)$ is the boundary condition at $\boldsymbol{x}_i$. $\beta$ is the penalty factor.

However, the penalty factors introduce additional hyperparameters and can compromise the uniqueness of the solution. The penalty factor is a method of satisfying boundary conditions as a soft constraint, which is an approximate way of meeting boundary conditions. Therefore, using a penalty factor to approximately satisfy boundary conditions can theoretically lead to non-unique solutions in PINNs.

Therefore, strictly enforcing boundary conditions is a very important direction for PINNs. One method of strict enforcement is through the use of distance networks. Consider the following mechanical equation:

$$\begin{cases} \text{Domain Control Equation: } \boldsymbol{L}(\boldsymbol{u}(\boldsymbol{x})) = \boldsymbol{f}(\boldsymbol{x}) & \forall \boldsymbol{x} \in \Omega \\ \text{Displacement Boundary Condition: } \boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x}) & \forall \boldsymbol{x} \in \Gamma^u \\ \text{Force Boundary Condition: } \boldsymbol{\sigma} \cdot \boldsymbol{n} = \boldsymbol{t}(\boldsymbol{x}) & \forall \boldsymbol{x} \in \Gamma^t \end{cases}, \tag{40}$$

where $\boldsymbol{L}$ is the differential operator, $\boldsymbol{u}(\boldsymbol{x})$ is the displacement at $\boldsymbol{x}$, $\boldsymbol{f}(\boldsymbol{x})$ is the body force, $\boldsymbol{h}(\boldsymbol{x})$ is the Dirichlet boundary ($\Gamma^u$) condition, $\boldsymbol{t}(\boldsymbol{x})$ is the Neumann boundary ($\Gamma^t$) condition, and $\boldsymbol{\sigma}$ is the stress tensor. To solve this PDEs, we construct the displacement as follows:

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{D}(\boldsymbol{x}) * \boldsymbol{u}_g(\boldsymbol{x}; \boldsymbol{\theta}), \tag{41}$$

where an approximation function fits $\boldsymbol{h}(\boldsymbol{x})$ as particular network, $\boldsymbol{D}(\boldsymbol{x})$ is the distance network, which outputs the shortest distance to the displacement boundary based on the coordinate $\boldsymbol{x}$:

$$\boldsymbol{D}(\boldsymbol{x}) = \min_{\boldsymbol{y} \in \Gamma^u} \sqrt{(\boldsymbol{x} - \boldsymbol{y}) \cdot (\boldsymbol{x} - \boldsymbol{y})}. \tag{42}$$

This distance can be pre-calculated for a finite set of points using nearest neighbor algorithms and then approximated by a fitter $D(\boldsymbol{x})$. Finally, $\boldsymbol{u}_g(\boldsymbol{x}; \boldsymbol{\theta})$ is a neural network that learns based on the PDEs and the force boundary condition. The advantage of the distance network is that if the coordinate point lies on the displacement boundary $\Gamma^u$, where the displacement network $D(\boldsymbol{x})$ equals zero, the output of the displacement field from Eq. (41) is exactly $\boldsymbol{h}(\boldsymbol{x})$. This means that during the training of PINNs in the strong form, there is no need to consider the displacement boundary condition. Especially in the energy form of PINNs, the energy principle allows force boundary conditions to be directly incorporated into the energy functional without prior consideration. Therefore, the energy form of PINNs does not need to explicitly handle boundary conditions, significantly reducing the number of hyperparameters.

Thus, distance networks are important for PINNs, both in the strong and energy forms. Research on distance networks primarily focuses on exploring admissible displacement field compositions to better train PINNs. The following is a review of distance networks: Lagaris et al. (1998) [21] initially proposed the concept of PINNs as well as the construction of admissible displacement fields that meet essential boundary conditions, but only provided a general mathematical form and specific construction methods in regular domains, i.e., converting the distance network into a multiplication by coordinates. Later, Lagaris et al. (2000) [135] proposed the construction of admissible displacement fields for irregular domains. McFall et al. (2009) [136, 137] discussed in detail the procedure for solving boundary value problems of arbitrary boundary shapes using neural networks. With PINNs receiving significant attention in recent years, this research area has been revitalized, and Berg et al. (2018) [60] proposed the form of admissible displacement fields in Eq. (41), where the particular solution network $\boldsymbol{h}(\boldsymbol{x})$, distance network $\boldsymbol{D}(\boldsymbol{x})$, and general network $\boldsymbol{u}_g(\boldsymbol{x}; \boldsymbol{\theta})$ are all approximated through fully connected neural networks, essentially similar to the method proposed by Lagaris et al. (2000) [135]. Later, Rao et al. (2021) [93] extended the form of admissible displacement fields to linear elastic dynamics; Sheng et al. (2021) [116] proposed using RBF (Radial Basis Function) instead of the fully connected form of the distance network to solve the energy form of PINNs. It is worth mentioning that Sukumar et al. (2022) [138] discussed PINNs' distance networks in great detail, even satisfying force boundary conditions in addition to displacement boundary conditions.

In summary, research on distance networks seems to have reached its developmental limit at the methodological level. In the future, the concept of distance networks is expected to be more widely integrated into various PINNs and operator learning algorithms, aiming to enhance the accuracy of these algorithms and reduce their reliance on hyperparameters.

### *3.5. Transfer Learning*

Transfer learning is a crucial concept and application in machine learning, and also applied in PINNs. The main advantage of using transfer learning in PINNs is due to iterative algorithms, which can inherit previously trained parameters, thus requiring fewer iterations for similar new tasks. The application approach of transfer learning in PINNs is generally the same, which involves directly inheriting the parameters from the previous task and then iterating in new tasks. There are two main methods of parameter iteration:

The first method is very simple; it involves completely inheriting the parameters and then re-iterating [26]. The second method involves freezing the first few layers of the neural network and only training the later layers to achieve the effect of transfer learning. For example, Goswami et al. (2020) [102] proposed using transfer learning to reduce the computational cost of phase-field modeling in fracture mechanics. In phase-field modeling, incremental step loading is required, so recalculating each step would be computationally expensive. By transferring the parameters learned by the PINNs energy method from the previous step to the next incremental step and training only the later layers, the iteration time for subsequent steps is reduced. Similarly, Chakraborty (2021) [38] proposed a transfer learning approach that incorporates approximate PDEs, and freezed the parameters of the initial layers, and uses high-precision experimental data to train the parameters of the later layers of the neural network that have been trained by PDEs. Xu et al. (2023) [131] suggested first training PINNs on a simple geometric model, and freezing the parameters of the shallow layers of the fully connected neural network while only training the deep layers in real tests with complex geometries. The rationale is that most geometric problems in structural mechanics are similar, and the basic features extracted by the shallow layers can be combined with the deep layers to form similar geometries, allowing the parameters to be directly transferred and reducing the number of iterations in PINNs.

In summary, these two methods in transfer learning are quite common in machine learning. This approach allows us to leverage models that have been pre-trained on large datasets and use transfer learning to address new tasks with smaller datasets or tasks that are slightly different from the original ones.

### *3.6. Gradient Approximation*

The core of AI for PDEs lies in the computation of differential operators for PDEs. Therefore, the calculation of gradients is particularly crucial for solving PDEs using neural networks. There are generally three methods to compute gradients in this field: numerical difference approximation, symbolic computation, and automatic differentiation [139]. Each of these methods is reviewed below.

1. Numerical Difference Approximation: This method essentially approximates the limit form of derivatives, ignoring higher-order terms to achieve an approximate differential. The approach to approximate gradients using the numerical difference in AI for PDEs aligns with traditional numerical analysis and doesn't fundamentally differ. Numerical differences include forward, backward, and central differences, which differ in terms of error order. The advantage of numerical difference lies in its simplicity and wide applicability; it does not require knowledge of the derivative's analytic form. However, its drawbacks include significant truncation and rounding errors. Therefore, despite its broad applicability, its error characteristics must be carefully considered. Recently, some methods have used CNNs to approximate gradients, essentially using fixed convolution kernel weights to simulate finite difference [83].

2. Symbolic Computation: This method obtains analytical solutions of gradients through derivative operations (e.g., product rule, chain rule), and then inserts specific values to achieve the final gradients. Typically implemented recursively, this method offers high precision but can become inefficient with complex functions due to expression swelling [140], which is challenging to simplify.

3. Automatic Differentiation: Automatic differentiation is similar to symbolic computation but avoids the problem of expression swelling by breaking down complex functions into combinations of simple functions and using known derivatives of basic functions through the chain rule. During derivative computation, node values from forward operations are used in the chain rule formulas, thus eliminating the need for a complete analytical expression of gradients, thereby addressing the issue of expression swelling seen in symbolic computation. There are two modes of implementation for automatic differentiation: forward mode and reverse mode. Forward mode computes function values and gradients simultaneously in one forward operation, which requires less memory and is suitable for problems with fewer parameters. However, reverse mode, which computes derivatives layer by layer from the output back inward, is more common in deep learning due to its suitability for problems with smaller output dimensions and larger parameter sets. Overall, automatic differentiation not only provides precise derivatives, similar to symbolic computation, but also solves the weaknesses of symbolic methods, such as expression swelling. Furthermore, the reverse mode is particularly well-suited to the small-output-dimensional context of deep learning. In AI for PDEs, it is common to compute the derivatives of a single scalar (such as the loss function) concerning a large number of weights, making the reverse mode very suitable for this scenario. Thus, the reverse mode is the predominant choice for derivative computation in the AI4PDEs domain.

In recent years, apart from automatic differentiation, another method of gradient approximation has emerged through statistical approaches [82, 68]. This method does not require backward gradient propagation for differentiation. For instance, assuming a function $f(x)$, where $f(x)$ can be expressed as:

$$u(x) = E_{\delta\sim N(0,\sigma^2)} f(x+\delta), \tag{43}$$

where $u(x)$ is the field quantity of interest. We consider a simple first-order derivative problem:

$$u^{'}(x) = g(x). \tag{44}$$

Eq. (43) can transform Eq. (44). The derivative of $u(x)$ can be converted to:

$$\begin{aligned} u(x) = E_{\delta\sim N(0,\sigma^2)} f(x+\delta) &= \int_{\mathbb{R}} f(x+\delta)\frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{\delta^2}{2\sigma^2})d\delta \\ &= \frac{1}{\sigma\sqrt{2\pi}}\int_{\mathbb{R}} f(t)\exp[-\frac{(t-x)^2}{2\sigma^2}]dt \end{aligned} \tag{45}$$

$$\begin{aligned} u(x) = E_{\delta\sim N(0,\sigma^2)} f(x+\delta) &= \int_{\mathbb{R}} f(x+\delta)\frac{1}{\sigma\sqrt{2\pi}}exp(-\frac{\delta^2}{2\sigma^2})d\delta \\ &= \frac{1}{\sigma\sqrt{2\pi}}\int_{\mathbb{R}} f(t)exp[-\frac{(t-x)^2}{2\sigma^2}]dt \\ u^{'}(x) &= \frac{1}{\sigma\sqrt{2\pi}}\int_{\mathbb{R}} f(t)exp[-\frac{(t-x)^2}{2\sigma^2}](\frac{(t-x)}{\sigma^2})dt \\ &= \frac{1}{\sigma\sqrt{2\pi}}\int_{\mathbb{R}} f(x+\delta)exp[-\frac{\delta^2}{2\sigma^2}](\frac{\delta}{\sigma^2})d\delta \\ &= E_{\delta\sim N(0,\sigma^2)}[\frac{\delta}{\sigma^2}f(x+\delta)] \end{aligned} \tag{46}$$

Eq. (46) turns the derivative computation into a process of calculating expectations, allowing the problem to be reformulated as solving for $f$ instead of computing derivatives through backward AD. As the expectation of forward computations converges to the true gradient, this method theoretically can replace AD algorithms, enhancing computational efficiency. The accuracy of the approximate derivatives is determined by the variance, a topic within the convergence study of Monte Carlo methods not further discussed here.

### *3.7. Convergence Proofs*

This section summarizes articles related to mathematical proofs of convergence for AI for PDEs. As shown in Table 5, the proofs are divided into three main parts: the first part concerns the convergence proofs for both direct and inverse problems in PINNs, along with proofs that neural networks can approximate any continuous function and articles on uncertainty estimates in PINNs; the second part relates to operator learning, which includes proofs that neural operators can approximate any continuous operator, specific proofs of convergence for the Fourier Neural Operator (FNO) within neural operators, and proofs that DeepONet can approximate any continuous operator, meaning it can approximate any family of continuous PDEs; the third part discusses how Physics-Informed Neural Operators (PINO) can approximate any continuous function or any continuous operator.

In summary, a single-layer feedforward network can represent any continuous function (universal approximation theory) [141]. Fig. 8 briefly illustrates the core concept of the universal approximation theory. To explain it briefly, we consider an arbitrary one-dimensional continuous function $f(x)$, where $x \in [a, b]$. We approximate it using a fully connected neural network $NN(x) = \sum_{i=1}^{N} s_i\sigma(w_i x + b_i)$, where the activation function is chosen as:

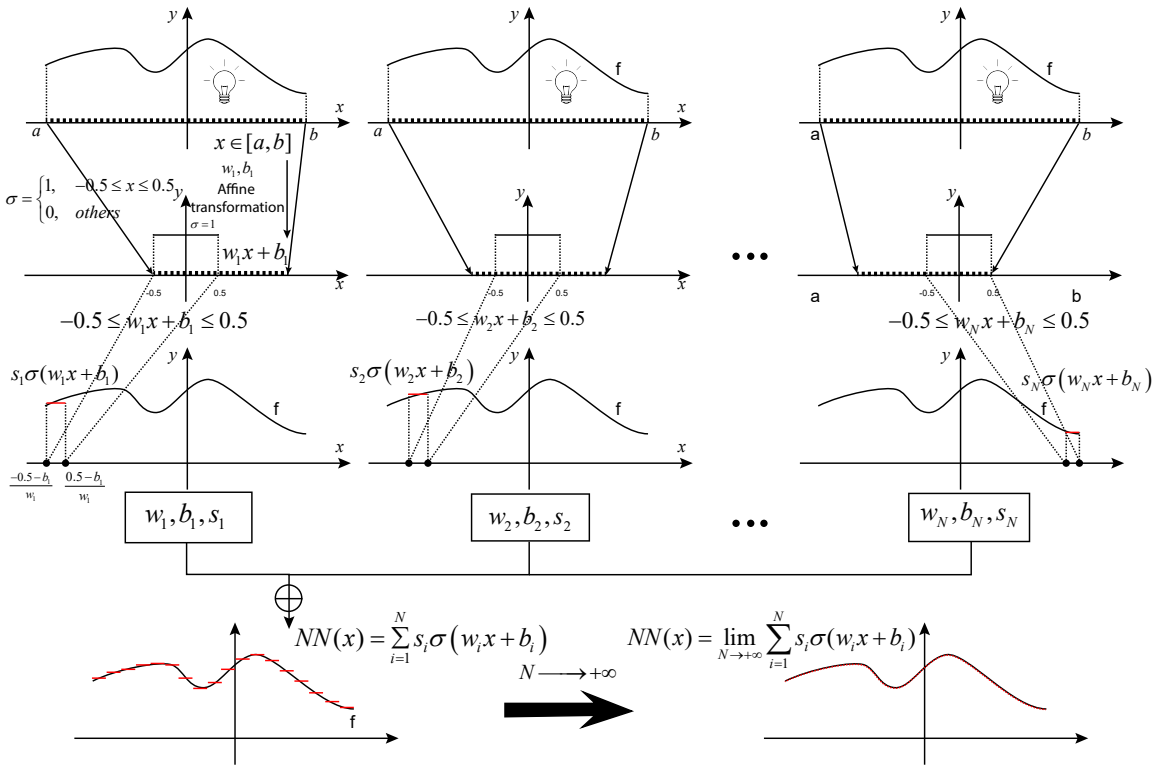


**Fig. 8.** A brief explanation of the universal approximation theorem for neural networks.

**Table 5**
AI for PDEs convergence proofs

| Literature | Brief Description of Proof |
|---|---|
| [63][62][128][143] | Proofs of the strong fitting capabilities of NNs |
| [144] | Convergence proofs for elliptic second-order linear and parabolic PDEs using PINNs |
| [145] | Proofs of approximation properties of inverse problems using PINNs |
| [146] | Estimations of uncertainty in PINNs |
| [36] | Proofs that neural operators can approximate any continuous operator |
| [122] | Proofs that the FNO in neural operators can approximate any continuous operator |
| [39] | Theoretical support for DeepONet: NNs can approximate any continuous operator |
| [147] | Proofs that DeepONet can approximate any continuous operator |
| [121] | PINO can approximate any continuous function or any continuous operator |

$$\sigma(x) = \begin{cases} 1 & \text{if } -0.5 \leq x \leq 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

It is easy to observe that $w_i x + b_i$ is non-zero only when it lies within the range $[-0.5, 0.5]$; outside this range, the result is zero. Therefore, after applying the activation function $\sigma$, the range of $x$ where $\sigma(w_i x + b_i)$ is non-zero can be expressed as:

$$\frac{-0.5 - b_i}{w_i} \leq x \leq \frac{0.5 - b_i}{w_i} \tag{48}$$

Next, we only need to adjust $s_i$ such that:

$$||f(x) - s_i \sigma(w_i x + b_i)||_{L^\infty} \leq \varepsilon_i, \text{ where } \frac{-0.5 - b_i}{w_i} \leq x \leq \frac{0.5 - b_i}{w_i} \tag{49}$$

where $||\cdot||_{L^\infty}$ represents the maximum value of the function. Clearly, by controlling the affine transformations $w_i$ and $b_i$, we can achieve a piecewise approximation, as shown in Fig. 8. Therefore, with a sufficient number of parameters $\{w_i, b_i, s_i\}_{i=1}^N$, we can approximate $f$ within any desired error $\varepsilon$. Mathematically, this can be expressed as follows: for any given error $\varepsilon$, there exists an $N$ such that:

$$||f(x) - \sum_{i=1}^{N} s_i \sigma(w_i x + b_i)||_{L^\infty} \leq \varepsilon = \max\{\varepsilon_i\}_{i=1}^N, \text{ where } a \leq x \leq b \tag{50}$$

The above explanation is an informal but intuitive interpretation of the universal approximation theorem. This proof only applies to single-layer networks with a one-dimensional target function, not to higher dimensions. A formal proof can be found in Table 5 under "Proofs of the strong fitting capabilities of NNs." Empirically, using deeper networks can achieve the same function with fewer units and improve generalization error [142]. However, neural networks may require an impractically large number of units and may struggle with optimization problems. Most of the convergence proofs demonstrate that a neural network exists which can approximate a function or operator within $\varepsilon$, but they do not provide a method for determining its parameters and hyperparameters. Therefore, future research could focus on developing theoretical frameworks that offer guidance on neural network architecture design to achieve better convergence.

## 4. AI for PDEs in the application of forward problems of computational mechanics

In this chapter, we review the applications of AI for PDEs in computational mechanics. Specifically, we focus on their application in addressing forward problems across various domains such as solid mechanics, fluid dynamics, and biomechanics.

### *4.1. Solid mechanics*

We focus on discussing the latest research developments in the field of AI for PDEs within solid mechanics. Therefore, we detail various examples of applications of AI for PDEs in solid mechanics. For the strong form of PINNs, it can solve almost any forward problem in solid mechanics, but for the energy form, it is only applicable to solid mechanics problems that adhere to energy principles, such as linear elastic statics, hyperelastic problems, and phase field method simulations of fractures.

#### *4.1.1. Linear elasticity mechanics*

The domain PDEs in linear elasticity mechanics are composed of three sets of equations: the equilibrium equations, constitutive equations, and kinematic equations:

$$\begin{cases} \text{Equilibrium equations: } \sigma_{ij,j} + f_i = \rho \ddot{u}_i & \forall (\boldsymbol{x}, t) \in \Omega \times [0, T] \\ \text{Constitutive equations: } \sigma_{ij} = 2G\varepsilon_{ij} + \lambda \varepsilon_{kk} \delta_{ij} & \forall (\boldsymbol{x}, t) \in \Omega \times [0, T] \\ \text{Kinematic equations: } \varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) & \forall (\boldsymbol{x}, t) \in \Omega \times [0, T] \end{cases}, \tag{51}$$

where $\lambda$ and $G$ are the Lame parameters. $\boldsymbol{u}$, $\boldsymbol{\epsilon}$, and $\sigma$ are the displacement, strain, and stress respectively. Boundary and initial conditions include initial displacement conditions, initial velocity conditions, displacement boundary conditions, and force boundary conditions:

$$\begin{cases} \text{Initial displacement condition: } \boldsymbol{u}(\boldsymbol{x}, 0) = \boldsymbol{u}_0(\boldsymbol{x}) & \forall \boldsymbol{x} \in \Omega \\ \text{Initial velocity condition: } \dot{\boldsymbol{u}}(\boldsymbol{x}, 0) = \boldsymbol{v}_0(\boldsymbol{x}) & \forall \boldsymbol{x} \in \Omega \\ \text{Displacement boundary condition: } \boldsymbol{u}(\boldsymbol{x}, t) = \bar{\boldsymbol{u}}(\boldsymbol{x}, t) & \forall (\boldsymbol{x}, t) \in \Gamma^u \times [0, T] \\ \text{Force boundary condition: } \boldsymbol{t}(\boldsymbol{x}, t) = \bar{\boldsymbol{t}}(\boldsymbol{x}, t) & \forall (\boldsymbol{x}, t) \in \Gamma^t \times [0, T] \end{cases}. \tag{52}$$

The above set of equations represents all the control equations and boundary conditions in linear elasticity mechanics. If we consider the structural characteristics of the problem, these equations can be simplified, such as in plate and shell mechanics. However, the essence of the core solution remains solving PDEs in Eq. (51) with boundary conditions in Eq. (52). In the strong form of PINNs, Eq. (51) and Eq. (52) are directly coupled through different hyperparameters into one loss function for solving. In the energy form of PINNs, these equations are transformed into an energy functional, such as the principle of minimum potential energy or minimum complementary energy. However, only static problems can be transformed. Although dynamic problems also involve the Hamiltonian functional, the functional of dynamic problems is stationary problems. The requirement of minimum potential and complementary energies is an extremum problem. Due to the highly non-convex nature of neural networks, solving the Hamiltonian functional in dynamics faces immense optimization difficulties due to infinite saddle points, making it extremely challenging to obtain stationary points. Additionally, a significant difficulty with the Hamiltonian functional's admissible displacement field is that it requires the displacement field to satisfy the final moment displacement solution (a necessity in Hamiltonian theory). Unfortunately, we do not know what the final moment displacement field is. Thus, the high degree of non-convexity and the unknown nature of the final moment displacement field pose significant challenges for the energy method of PINNs in dynamic problems in linear elasticity. The above analysis is just an application of PINNs in linear elasticity problems, while AI for PDEs encompasses not only PINNs but also operator learning and PINO. Therefore, we will next provide an overview of AI for PDEs in the forward problems of linear elasticity in solid mechanics.

Rao et al. (2021) [93] used the strong form of PINNs algorithm and the construction of admissible displacement fields to solve linear elasticity dynamics problems, as shown in Fig. 9a. Guo et al. (2021) [148] used the strong form of PINNs algorithm for the first time in Kirchhoff plates. Subsequently, Zhuang et al. (2021) [99] applied the energy form of PINNs using the principle of minimum potential energy in Kirchhoff plates. Li et al. (2021) [100] compared the strong and energy forms of PINNs in Kirchhoff plates, as illustrated in Fig. 9b. Bai et al. [149] studied modified loss functions in strong form, solving elastic problems with geometric nonlinearity and constitutive linearity. Later, Sun et al. (2023) [77] combined PINNs with boundary element methods. Mathematically, the strong form of PDEs can be converted into the weak form through Gauss's divergence theorem. If

the differential equation involves a linear self-adjoint operator, the weak form can be further transformed into an energy form. If the weak form is subjected to Gauss's divergence theorem again until the highest derivatives in the test functions are achieved, thereby reducing the differential equation to its lowest order, it then transforms into the inverse form. Boundary elements leverage the inverse form to formulate boundary integral equations. Thus, the strong form, energy form (or weak form), and inverse form are all different representations of the same PDEs. However, due to limitations in the fundamental solutions of Green's functions in boundary elements, currently, only problems in linear elasticity can be solved, as shown in Fig. 9c.

The above literature review discusses using PINNs to replace traditional numerical methods, such as the finite element method, for solving linear elasticity problems. Additionally, operator learning has also been applied to linear elasticity problems. For instance, Wang et al. (2024) [150] proposed using the Fourier Neural Operator (FNO) to replace the calculation process of elastic tensors in mechanical homogenization. In the original problem, traditional methods required finite element calculations of displacement fields under six loading conditions for different geometries, followed by numerical homogenization formulas to obtain the fourth-order equivalent elastic tensor. Since the boundary conditions in the numerical homogenization process are fixed and only the geometry changes, a large amount of data can be calculated using finite elements for different geometries, then operator learning can be used to learn the mapping from geometry to displacement fields, and finally, it can be input into traditional numerical homogenization programs, significantly improving computational efficiency (about 1000 times faster compared to traditional finite element methods). Additionally, in multiscale calculations, the computation of Representative Volume Elements (RVEs) is a key limiting factor in multiscale algorithms. Liu and Wang et al. (2024) [151] proposed a multiscale framework based on operator learning aimed at improving material energy utilization rates, starting from molecular dynamics simulations based on material formulas to compute thermal conductivities, then using RVEs to calculate equivalent thermal conductivities under different geometries, and finally integrating the equivalent thermal conductivities into real architectural structures to predict energy utilization rates. Operator learning can improve the efficiency of computation in RVE. This framework, which spans multiple scales from nanoscale molecular dynamics simulations to microscale RVEs based on operator learning, and to macroscale numerical simulations of buildings, is a multiscale simulation framework that can predict the energy utilization rates of new materials. When faced with new materials, using the aforementioned framework with operator learning can significantly accelerate the multiscale simulation of new materials, thus better contributing to global climate efforts.

#### *4.1.2. Elastoplastic mechanics*

Elastoplastic theories encompass various models that describe the relationship between strain increments and stress during the plastic deformation phase. Abueidda et al. (2021) [27] utilized PINNs with J2 flow theory, kinematic hardening, and isotropic hardening models to address elastoplastic problems. Here, we illustrate how PINNs solve elastoplastic issues, starting by specifying parameters for elastoplastic problems: the Lame constants $\lambda$ and $G$, yield stress $\sigma_s$, parameters $K$ describing isotropic hardening of the yield surface, and parameter $H$ for back stress evolution. Using traditional PINNs, sufficient points are sampled on the domain displacement boundaries, and force boundaries. We initialize variables that will evolve iteratively. The variables to be initialized include initial plastic strain $\boldsymbol{e}_o^p$, initial back stress $\boldsymbol{q}_o$, and initial internal variable $\alpha_0$ for isotropic hardening. According to J2 flow theory, the yield condition with the kinematic hardening and the isotropic hardening models is:

$$\mathcal{F}(\boldsymbol{\sigma}, \boldsymbol{q}, \alpha) = \sqrt{\boldsymbol{\eta} : \boldsymbol{\eta}} - \sqrt{\frac{2}{3}}(\sigma_y + K\alpha), \tag{53}$$

where $\boldsymbol{\eta} = \boldsymbol{s} - \boldsymbol{q}$, with $\boldsymbol{s}$ representing the deviatoric stress tensor reduced by the hydrostatic stress, and $\boldsymbol{q}$ is the back stress. Given the path-dependent nature of plastic mechanics, we typically employ incremental loading for numerical simulation. Setting boundary conditions incrementally, we consider a pseudo-time $t = 1$ scenario, using the PINNs network (inputting coordinates, outputting displacement fields) to output displacement fields at points within the domain and on displacement and force boundaries. Using automatic differentiation, we derive the gradient of displacement. From the relationship between strain and displacement gradients, we compute the strain at $t = 1$, denoted $\boldsymbol{\varepsilon}_1$, and from this strain, compute the deviatoric strain $\boldsymbol{e}_1$.

According to the predefined variables, including initial plastic strain $\boldsymbol{e}_o^p$, initial back stress $\boldsymbol{q}_o$, and initial

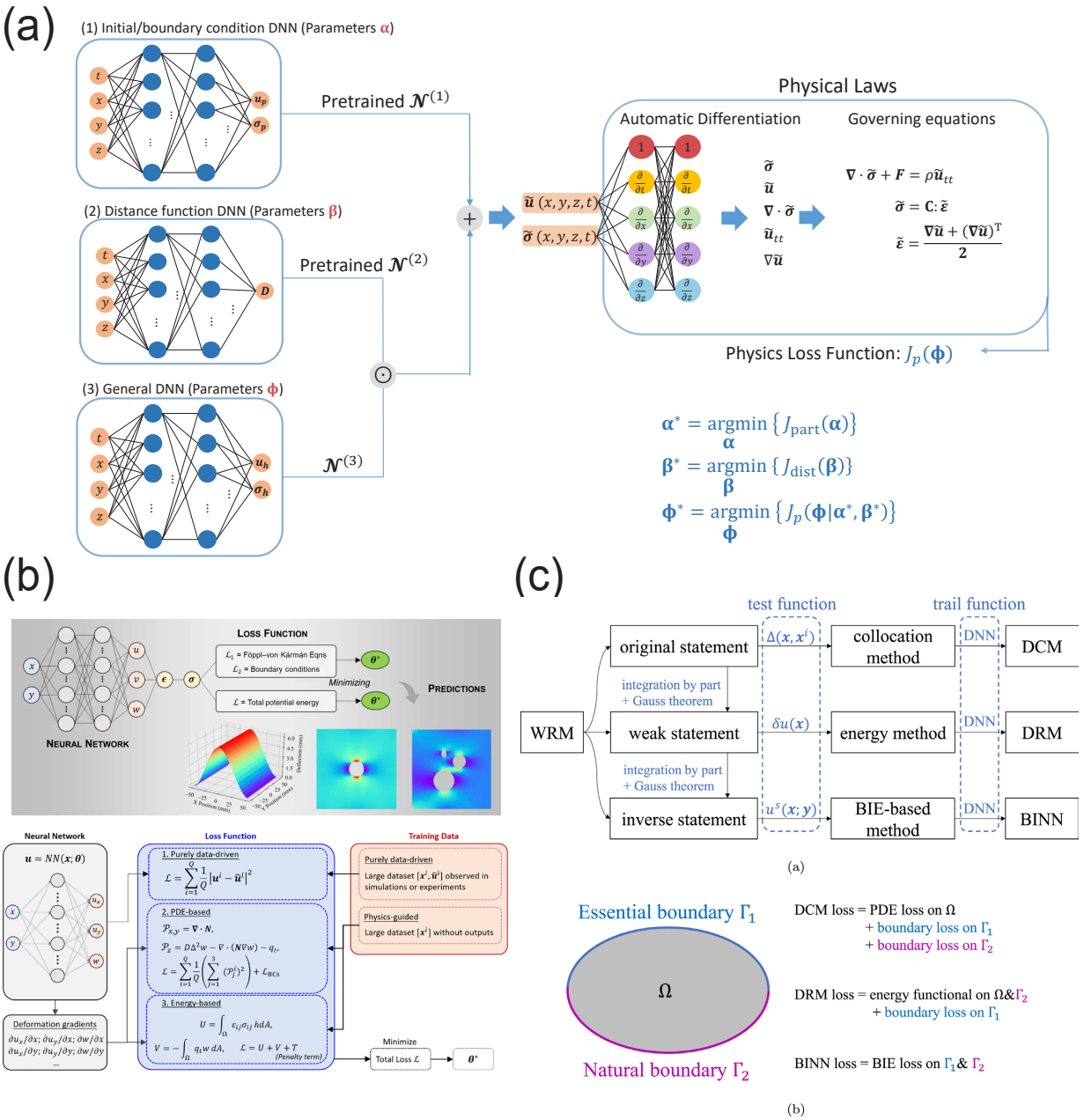


**Fig. 9.** Applications of PINNs in elastic mechanics. (a) Strong form: linear elastic dynamics [93] , (b) Strong and energy form: Kirchhoff plates [100], (c) Inverse form: elasticity problems [77]

isotropic hardening internal variable $\alpha_0$, we calculate the trial deviatoric stress $\boldsymbol{s}_1^{trial}$, trial relative stress $\boldsymbol{\eta}_1^{trial}$, and the trial state of the yield condition $\mathcal{F}_1^{trial}$:

$$\begin{aligned}\boldsymbol{s}_1^{trial} &= 2G(\boldsymbol{e}_1 - \boldsymbol{e}_0^p)\\ \boldsymbol{\eta}_1^{trial} &= \boldsymbol{s}_1^{trial} - \boldsymbol{q}_o\\ \mathcal{F}_1^{trial} &= \sqrt{\boldsymbol{\eta}_1^{trial} : \boldsymbol{\eta}_1^{trial}} - \sqrt{\frac{2}{3}}(\sigma_y + K\alpha_0)\end{aligned} . \tag{54}$$

We determine whether the material enters the plastic phase by checking if $\mathcal{F}_1^{trial}$ is less than or equal to zero. If $\mathcal{F}_1^{trial}$ is less than or equal to zero, the material is considered to be in the elastic phase, and stress is calculated directly using $\boldsymbol{s}_1^{trial}$:

$$\boldsymbol{\sigma}_1 = \lambda trace(\boldsymbol{\varepsilon}_1)\boldsymbol{I} + \boldsymbol{s}_1^{trial}. \tag{55}$$

Using this stress and displacement field, we compute the loss function for $t = 1$ using the balance equations, displacement boundary conditions, and force boundary conditions, and optimize the parameters of the PINNs network, verifying in each optimization cycle whether the material remains elastic. If $\mathcal{F}_1^{trial}$ exceeds zero, we must adjust the previous steps, notably changing how stress is computed. Using $\boldsymbol{\eta}_1^{trial}$ and $\mathcal{F}_1^{trial}$, we compute the flow direction $\boldsymbol{n}_1$, plastic flow increment $\bigtriangleup\gamma_1$, and updated isotropic hardening internal variable $\alpha_1$ at $t = 1$:

$$\begin{aligned}\boldsymbol{n}_1 &= \frac{\boldsymbol{\eta}_1^{trial}}{\sqrt{\boldsymbol{\eta}_1^{trial} : \boldsymbol{\eta}_1^{trial}}}\\ \bigtriangleup\gamma_1 &= \frac{\mathcal{F}_1^{trial}}{2(G + \frac{H}{3} + \frac{K}{3})}\\ \alpha_1 &= \alpha_0 + \sqrt{\frac{2}{3}}\bigtriangleup\gamma_1\end{aligned} . \tag{56}$$

The back stress $\boldsymbol{q}$, plastic strain $\boldsymbol{e}^p$, and stress $\boldsymbol{\sigma}$ are updated accordingly:

$$\begin{aligned}\boldsymbol{q}_1 &= \boldsymbol{q}_0 + \frac{2}{3}\bigtriangleup\gamma_1 H\boldsymbol{n}_1\\ \boldsymbol{e}_1^p &= \boldsymbol{e}_0^p + \bigtriangleup\gamma_1\boldsymbol{n}_1\\ \boldsymbol{\sigma}_1 &= \lambda trace(\boldsymbol{\varepsilon}_1)\boldsymbol{I} + \boldsymbol{s}_1^{trial} - 2G\bigtriangleup\gamma_1\boldsymbol{n}_1\end{aligned} . \tag{57}$$

The loss function is optimized in the same way as during the elastic phase, using the revised stress and displacement fields along with corresponding equations and boundary conditions. This process continues iteratively through incremental loading steps until the loss function converges, marking the end of the elastoplastic simulation in the strong form of PINNs. The simulation results are compared with FEM solutions, focusing on displacement fields as illustrated in Fig. 10a.

The steps involved in solving elastoplastic problems using the strong form of PINNs are similar to traditional computational mechanics, except that the approximation function has shifted from the shape functions in FEM to neural networks. Due to the abundance of plasticity theories, aside from the J2 flow rule, other theories can also be simulated using PINNs, thus facilitating the evaluation of PINNs models in handling plastomechanical issues. Furthermore, He et al. (2023) [105] proposed using the energy method with the plastic variational formulation for calculating elastoplastic problems, as shown in the process diagram in Fig. 10a. The plastic variational formulation is based on the theory proposed by Simo et al. (2006) [152]. Since elastoplastic problems are related to the historical loading path, both the strong form and energy form of PINNs iteratively address each loading step.

Additionally, data-driven simulations of elastoplastic issues have been conducted, with Li et al. (2023) [114] using Geo-FNO (an enhanced version of FNO capable of operator learning for any geometry) to simulate forming and stamping processes. The algorithmic framework and simulation results are depicted in Fig. 10b.

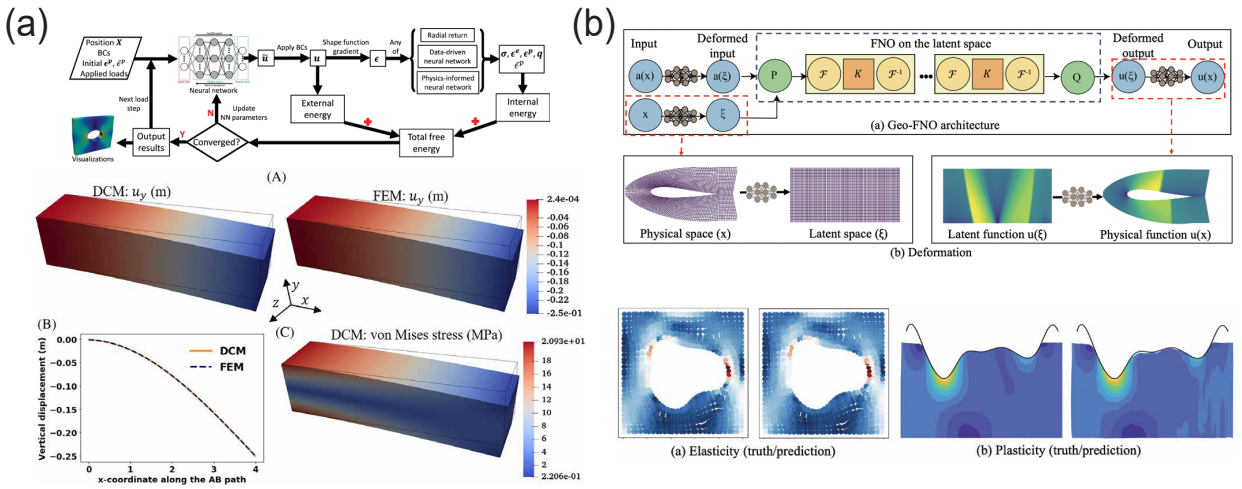


**Fig. 10.** Applications of PINNs and operator learning in elastoplastic mechanics: (a) PINNs in strong form [27] and energy form [105], (b) Operator learning with Geo-FNO [114].

The return mapping in Eq. (57) is a numerical algorithm for the elastoplastic KKT conditions. Mathematically, the KKT can be easily transformed into an optimization problem using e.g. Fischer-Burmeister replacement functions [153], which allows for much larger load steps and can lead to computational savings. Therefore, in the future, a PINNs algorithm based on Fischer-Burmeister replacement functions could be developed to solve elastoplastic problems.

### *4.1.3. Hyperelastic mechanics*

Hyperelastic problems not only involve geometrical nonlinearity but also material nonlinearity, and they can be computed using the minimum potential energy principle in the energy form of PINNs. Traditional hyperelastic problems are path-independent, making them a simpler type of nonlinear mechanics problem. Therefore, many studies in mechanics using PINNs often choose hyperelastic problems as examples. There are two main considerations in the PINNs simulation of hyperelastic problems: the first is the selection of the potential function according to the constitutive law of hyperelasticity material, and the second is that the kinematic equations must consider the nonlinear terms of large deformations. To illustrate, let us consider the potential function of the Neo-Hookean constitutive model:

$$\Psi = \frac{1}{2}\lambda(\ln J)^2 - G\ln J + \frac{1}{2}G(trace(\boldsymbol{C}) - 3) \quad , \tag{58}$$

where $J$ is the determinant of the deformation gradient $\boldsymbol{F}$, and $\boldsymbol{C}$ is the right Cauchy-Green deformation tensor ($\boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F}$). We first establish a neural network mapping from material coordinates $\boldsymbol{X}$ to spatial coordinates $\mathbf{x}$. We scatter points within the domain and then use automatic differentiation to calculate the deformation gradient $\boldsymbol{F} = \partial\boldsymbol{x}/\partial\boldsymbol{X}$, which is substituted into Eq. (58) to determine the density of the hyperelastic strain energy. Once $\boldsymbol{F}$ is known, hyperelastic strain energy can be obtained through some simple algebraic operations. Since the potential energy principle must consider the work done by external forces and the displacement field must satisfy displacement boundary conditions in advance, we consider displacement boundary conditions when hypothesizing admissible displacement fields, employing the construction method of Eq. (41). However, if the structure's shape is simple, the distance function can be directly constructed analytically (often by multiplying by coordinates, as suggested by [103].) For the work of external forces, the construction is entirely the same as in linear elasticity problems. Finally, we construct the overall loss function:

$$\mathcal{L} = \int_\Omega (\Psi - \boldsymbol{f}\cdot\boldsymbol{u})d\Omega - \int_{\Gamma^t} \bar{\boldsymbol{t}}\cdot\boldsymbol{u}d\Gamma. \tag{59}$$

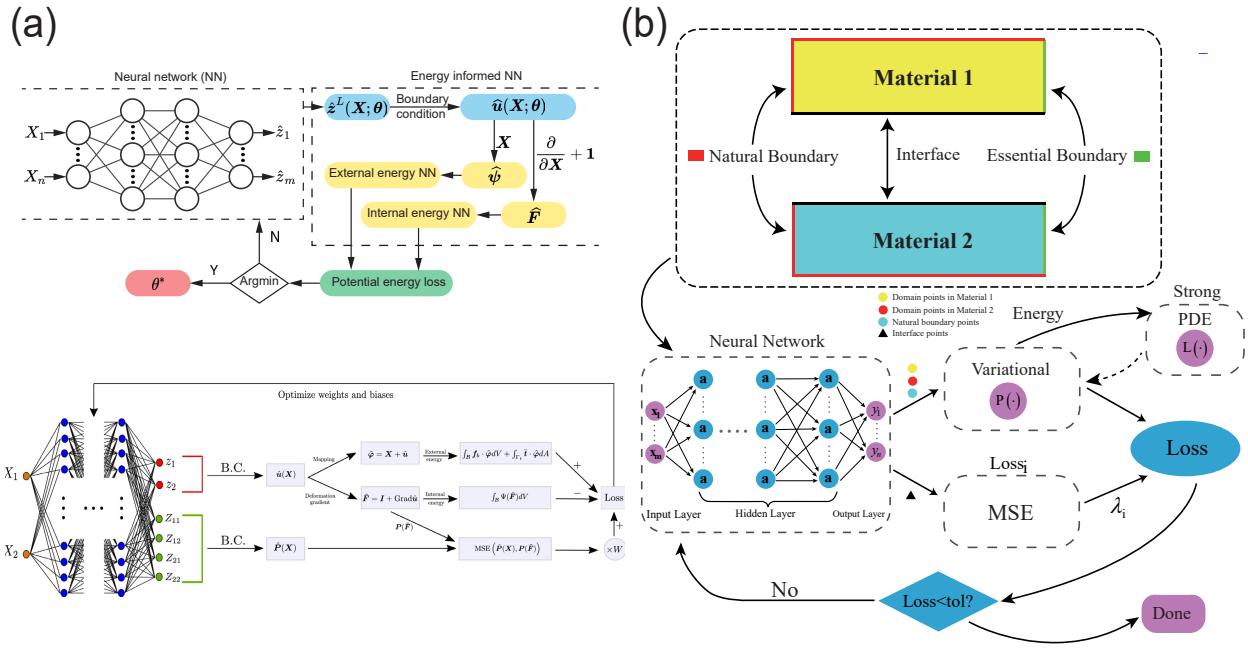


**Fig. 11.** Applications of PINNs in energy form for hyperelastic mechanics: (a) PINNs in energy form [103, 117] (b) PINNs in energy form with subdomains [31].

By optimizing the above loss function, we can obtain the neural network parameters for the mapping from material coordinates $\mathbf{X}$ to spatial coordinates $\mathbf{x}$, thus determining the corresponding mapping relationship. Once the mapping from material coordinates $\mathbf{X}$ to spatial coordinates $\mathbf{x}$ is known, all mechanical quantities become clear, including the displacement field, the first Piola-Kirchhoff stress $\mathbf{P}$, and the second Piola-Kirchhoff stress $\mathbf{S}$:

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{x} - \boldsymbol{X} \\ \boldsymbol{P} &= \frac{\partial \Psi}{\partial \boldsymbol{F}} = G\boldsymbol{F}^T + [\lambda \ln J - G]\boldsymbol{F}^{-1}. \\ \boldsymbol{S} &= \boldsymbol{P} \cdot \boldsymbol{F}^{-T} \end{aligned} \tag{60}$$

It is worth emphasizing that numerical integration in the energy form of PINNs is crucial, as it directly affects the calculation of functional. Nguyen-Thanh et al. (2020) [103] were the first to apply the PINNs energy principle to hyperelastic problems and introduced traditional numerical integration schemes into the PINNs energy form (DEM), as shown in Fig. 11a. Subsequently, Fuhg et al. (2022) [117] extended the PINNs energy form by dividing the loss function into two parts: one based on the minimum potential energy principle and the other constructed through the equilibrium equation, as shown in Fig. 11a below. Wang et al. (2022) [31] extended the PINNs energy form algorithm to a subdomain form to solve hyperelastic problems, as shown in Fig. 11b. Bai et al. (2024) [104] use Radial Basis Function (RBF) networks to solve hyperelasticity torsional buckling cases in energy form. Although Abueidda et al. (2022) [154] were not the first to use the PINNs energy method to solve hyperelastic problems, they were the first to propose using the PINNs energy method to solve viscoelastic problems. Of course, we can also use the PINNs strong form to simulate hyperelastic problems, but the order of derivatives will increase, thereby reducing computational efficiency and accuracy, such as Abueidda et al. (2021) [27] using the PINNs strong form, directly considering the balance equation and boundary conditions to construct the loss function. It can be seen that the core difference between the PINNs strong form and the energy form is in the construction of the loss function.

*4.1.4. Fracture mechanics*

The phase field method is a crucial technique for simulating crack propagation [155, 156], differing from traditional linear elasticity as it introduces surface energy $\Psi_c$ in addition to the strain energy $\Psi_e$:

$$\Psi_c = \int_{\Gamma^c} G_c d\Gamma, \tag{61}$$

where $G_c$ is the critical energy release rate required for crack propagation, and $\Gamma^c$ represents the crack surface. Tracking the crack surface involves complex calculations, so Bourdin et al. (2000) [157] and Francfort et al. (1998) [158] approximate the crack surface using a diffuse interface with finite width inspired by the Mumford-Shah model [159] used in image segmentation:

$$\begin{aligned} \Psi_c &= \int_{\Gamma^c} G_c d\Gamma \approx \int_{\Omega} f_c d\Omega \\ f_c &= \frac{G_c}{2l_o}(\phi^2 + l_0^2|\nabla\phi|^2) - g(\phi)H(\boldsymbol{x},t) \end{aligned}, \tag{62}$$

where $\phi$ is the phase field describing the extent of fracture damage, and $g(\phi) = (1-\phi)^2$ represents the diminishing stress softening term. $l_0$ is the length scale parameter. $H(\boldsymbol{x},t)$, as introduced by Miehe et al. (2010) [155], is a strain history function that accounts for the irreversibility of crack growth, where $H(\boldsymbol{x},0)$ can describe the initial crack:

$$H(\boldsymbol{x},t) = \max_{s\in[0,t]} f_e^+, \boldsymbol{x} \in \Omega, \tag{63}$$

where $f_e^+$ is the positive part of the strain energy $\Psi_e$, acknowledging that compression typically does not contribute to damage. Thus, it is necessary to adjust the strain energy:

$$\Psi_e = \int f_e d\Omega = \int [g(\phi)f_e^+ + f_e^-] d\Omega, \tag{64}$$

where $f_e^+$ and $f_e^-$ represent the tensile and compressive strain energy densities, respectively. A common calculation method for $f_e^+$ and $f_e^-$ is the principal strain space decomposition:

$$\begin{aligned} f_e^{\pm} &= \frac{1}{2}\lambda < trace(\boldsymbol{\varepsilon}) >_{\pm}^2 + G trace(< \boldsymbol{\varepsilon} >_{\pm}^2) \\ < \cdot >_{\pm} &= \frac{|\cdot| \pm \cdot}{2} \end{aligned}. \tag{65}$$

Next, based on the principle of minimum potential energy:

$$\begin{aligned} \{\phi, \boldsymbol{u}\} = \arg\min_{\phi,\boldsymbol{u}} \Psi &= \Psi_e + \Psi_c - \int_{\Omega} \boldsymbol{f}\cdot\boldsymbol{u} d\Omega - \int_{\Gamma^t} \bar{\boldsymbol{t}}\cdot\boldsymbol{u} d\Gamma \\ s.t.\ \boldsymbol{u}(\boldsymbol{x}) &= \bar{\boldsymbol{u}}, \boldsymbol{x} \in \Gamma^d \end{aligned}. \tag{66}$$

This process optimizes the loss function $\Psi$, yielding the phase field $\phi$ and displacement field $\boldsymbol{u}$ for the next incremental step. The application of PINNs in the energy form for fracture mechanics fundamentally involves constructing a neural network mapping from spatial coordinates $\boldsymbol{x}$ to $\phi$ and $\boldsymbol{u}$, then optimizing $\Psi$.

Next we mention some related works. Goswami et al. (2020) [102, 101] were the first to use the PINNs energy method to solve phase-field fracture problems. Due to the need to optimize each incremental step, they adopted the idea of transfer learning, freezing the initial layers of the neural network to accelerate the simulation process, as shown in Fig. 12a. Subsequently, Goswami et al. (2022) [41] combined operator learning with the PINNs energy method to improve efficiency and accuracy, as illustrated in Fig. 12b.

PINNs can also solve phase field fracture in the strong form. The governing PDEs for a phase field fracture are:

$$\begin{aligned} &\nabla\cdot(g(\phi)\boldsymbol{\sigma}) + \boldsymbol{f} = 0 \\ &\frac{G_c}{l_0}\phi - G_c l_0 \nabla^2\phi = -g'(\phi)H(\boldsymbol{x},t) \end{aligned} \tag{67}$$

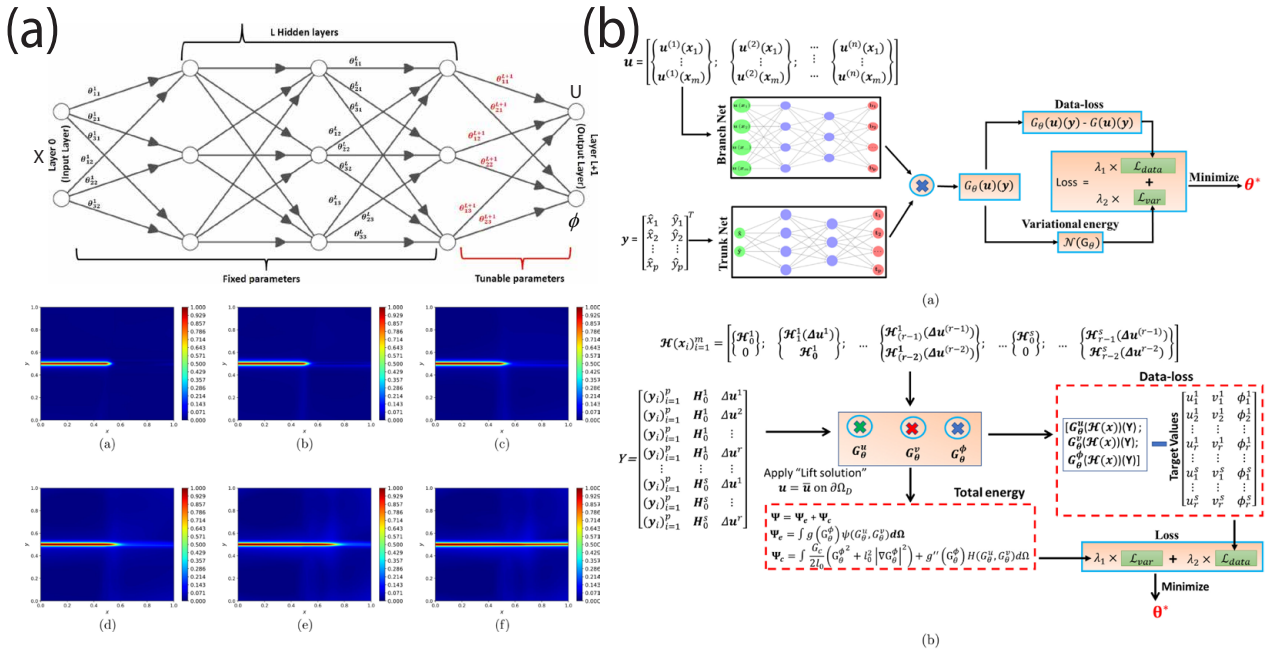


**Fig. 12.** Applications of PINNs and PINO in fracture mechanics using the phase field method: (a) PINNs in energy form [102] (b) Integration of PINNs in energy form with operator learning [41].

with boundary conditions:

$$\begin{aligned} g(\phi)\boldsymbol{\sigma}\cdot\boldsymbol{n} &= \bar{\boldsymbol{t}}, \boldsymbol{x}\in\Gamma^t \\ \boldsymbol{u} &= \bar{\boldsymbol{u}}, \boldsymbol{x}\in\Gamma^d. \\ \nabla\phi\cdot\boldsymbol{n} &= 0, \boldsymbol{x}\in\Gamma \end{aligned} \tag{68}$$

Through variational principles, it can be demonstrated that the strong forms shown in Eq. (67) with Eq. (68) and the energy form shown in Eq. (66) are theoretically interchangeable. However, it's crucial to note that even if different mathematical representations of the same physical law are theoretically equivalent, they often differ in computational efficiency and accuracy. Given that the PINNs' energy form involves lower orders of derivatives, it is particularly well-suited for solving phase field fracture theories.

In the future, since the Discrete Fracture Method (DFM) is a numerical technique used for simulating fractures and complex material behavior, a PINNs approach could be developed using DFM to simulate crack propagation problems.

### *4.1.5. Summary*

Overall, for addressing forward problems, current PINNs, in terms of accuracy and efficiency, still lag behind traditional numerical methods [26]. However, for inverse problems, PINNs offer a simpler approach because they avoid the traditional method of repeatedly solving forward problems to determine the parameters of the inverse problem. In PINNs, the parameters needed for the inverse problem are simply set as optimization variables within the existing deep learning framework, making the program and code structure quite straightforward. This does not imply that the underlying computations are easy; they involve derivatives of the loss function with respect to the optimization variables, which are still computationally intensive and complex. Yet, due to the effective encapsulation of current deep learning optimization frameworks, these complex derivation processes appear simpler.

Recent studies have shown that operator learning can significantly enhance the computational efficiency for forward problems. Although traditional operator learning is purely data-driven, current theoretical explanations suggest that after extensive data training, operator learning can learn the mappings of PDEs families.

Additionally, the integration of operator learning with physical equations holds promising academic and industrial prospects. This involves using existing data to propose a good initial solution, which is then fine-tuned using physical equations. This approach theoretically offers significant benefits in computational efficiency and accuracy, especially for complex nonlinear problems. In mathematics, some nonlinear problems can be transformed into iterations of nonlinear equation systems, where a good initial solution is vital for reducing iteration time. For instance, in hyperelasticity problems, operator learning can provide a good initial solution, followed by iterations based on the nonlinear equation system. Here, using finite element methods instead of PINNs to solve physical equations could enhance accuracy and efficiency, suggesting a combination of finite element methods and operator learning to greatly improve the computational efficiency of nonlinear problems.

This section has introduced the applications of AI for PDEs in addressing forward problems in solid mechanics. Next, we will discuss the applications of AI for PDEs in solving forward problems in fluid mechanics.

### *4.2. Fluid mechanics*

In this section, we will focus on the latest research on fluid mechanics with AI for PDEs. The algorithms in Section 2 are very popular in fluid mechanics.

#### *4.2.1. Hydrodynamics*

Hydrodynamics, a sub-discipline of fluid dynamics, focuses on the motion of liquids and the forces acting upon them. In hydrodynamics, the most popular governing equations, the Navier-Stokes (NS) equations, can be written as

$$\frac{D(\rho \boldsymbol{v})}{Dt} + \rho \boldsymbol{v}(\nabla \cdot \boldsymbol{v}) = \nabla \cdot \boldsymbol{\sigma} + \rho \boldsymbol{f} \tag{69}$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0, \tag{70}$$

where $\rho$ is the density of the fluid, $\boldsymbol{v}$ is the velocity vector in this section, $p$ is the pressure, $\boldsymbol{\sigma}$ is the stress tensor and $\boldsymbol{f}$ is the body force. Eq. (69) describes the conservation of momentum, while Eq. (70) ensures the conservation of mass. For incompressible flow, the material derivative of density is zero, i.e. $D\rho/Dt = \partial\rho/\partial t + \rho_{,i} v_i = 0$. Therefore, Eq. (70) can be simplified as

$$\nabla \cdot \boldsymbol{v} = 0. \tag{71}$$

To impose the incompressibility as a hard constraint in 2D problems, the stream function $\varphi$ can be used as the direct output of neural networks. In this manner, the velocity can be computed by

$$\begin{aligned} u &= \frac{\partial \varphi}{\partial y} \\ v &= -\frac{\partial \varphi}{\partial x} \end{aligned}. \tag{72}$$

It is easy to verify that Eq. (72) automatically satisfies the hard constraint of incompressibility in Eq. (71). For the sake of simplicity, herein we only introduce the Newtonian fluid model

$$\sigma_{ij} = (-p + \lambda S_{kk})\delta_{ij} + 2\mu S_{ij}^{'} \tag{73}$$

$$S_{ij}^{'} = S_{ij} - \frac{1}{3} S_{kk}\delta_{ij} \tag{74}$$

$$S_{ij} = \frac{1}{2}(v_{i,j} + v_{j,i}) \tag{75}$$

where $\mu$ and $\lambda$ are the viscosity coefficients. According to Eq. (71), we substitute Eq. (74) into Eq. (73), and obtain:

$$\sigma_{ij} = -p\delta_{ij} + \mu(v_{i,j} + v_{j,i}). \tag{76}$$

According to Eq. (71), substituting Eq. (76) into Eq. (69) yields the following equation:

$$\frac{D(\rho \boldsymbol{v})}{Dt} = -\nabla p + \mu \nabla^2 \boldsymbol{v} + \rho \boldsymbol{f}. \tag{77}$$

Guiding by Eq. (77), various formats of PDEs for fluid mechanics have been proposed, such as the VP format based on velocity and pressure, and the VV format based on vorticity and velocity. Rao et al. [97] proposed PINNs framework for incompressible laminar flows based VP formulation, as shown in Fig. 13a. In the framework, only the conservation of momentum is explicitly embedded into the loss function, while the incompressibility is naturally satisfied by using Eq. (72). Jin et al.[160] further provided a PINNs framework (NSFnets) informed by vorticity-velocity (VV) formulation for incompressible hydrodynamics and compared VV with VP formulation, as shown in Fig. 13b. By taking the curl of Eq. (77), VV formulation is obtained:

$$\begin{aligned}
\nabla \times \frac{D(\boldsymbol{v})}{Dt} &= \nabla \times (-\nabla \frac{p}{\rho} + \mu \nabla^2 \frac{\boldsymbol{v}}{\rho} + \boldsymbol{f}) \\
\epsilon_{ijk} \dot{v}_{j,i} + \epsilon_{ijk} (v_s v_{j,s})_{,i} &= -\frac{1}{\rho} \epsilon_{ijk} p_{,ji} + \nu \epsilon_{ijk} v_{j,ssi} + \epsilon_{ijk} f_{j,i} \\
\epsilon_{ijk} \dot{v}_{j,i} + \epsilon_{ijk} v_{s,i} v_{j,s} + \epsilon_{ijk} v_s v_{j,si} &= \nu \epsilon_{ijk} v_{j,ssi} \\
\frac{D(\boldsymbol{w})}{Dt} - \boldsymbol{w} \cdot (\nabla \boldsymbol{u}) &= \nu \nabla^2 \boldsymbol{w}
\end{aligned}, \tag{78}$$

where $\nu = \mu/\rho$ is the dynamic viscosity and $\boldsymbol{w} = \nabla \times \boldsymbol{v}$ denotes the vorticity. Note that the fluid is still incompressible, so in our derivation, we actually use $\partial \rho / \partial t = 0$.

Although the VP and VV formats mathematically address the same physical problem, they are not computationally equivalent. For instance, the VV format is more effective at high Reynolds numbers, while the VP format can directly describe velocity and pressure. The VV format involves first solving for vorticity and then using the resulting vorticity to update the velocity field through the Poisson equation.

Moreover, the adaptive weight technique proposed in [13] can be adopted. It is reported that PINNs can achieve high accuracy at $Re = 100$. Zhang et al. [161] introduced a PINN framework, the so-called dynamics random vorticity network (DRVN), via the random vortex method [162]. The proposed DRVN is capable of effectively addressing fractural and sharp singularity issues, which the NSFnet [160] struggles with. Rui et al. [163] implemented the Reynolds-averaged Navier-Stokes (RANS) equations. The Reynolds decomposition, which divides flow quantities into time-averaged and fluctuating terms, is applied to the NS equations. Thus, the governing equations in RANS can be written as

$$\rho \bar{\boldsymbol{v}} \cdot (\nabla \bar{\boldsymbol{v}}) = \rho \boldsymbol{f} - \nabla \bar{p} + \mu \nabla \cdot (\nabla \bar{\boldsymbol{v}} + \bar{\boldsymbol{v}} \nabla) - \rho \nabla \cdot \overline{\boldsymbol{v}' \boldsymbol{v}'}, \tag{79}$$

where $\overline{\boldsymbol{v}}$ and $\overline{\boldsymbol{v}'}$ are the mean (time-averaged) velocity and the fluctuating velocity, respectively. Reynolds decomposition refers that we can decompose velocity into the time average $\overline{\boldsymbol{v}}$ and the fluctuating velocity $\boldsymbol{v}'$, i.e. $\boldsymbol{v}(\boldsymbol{x}, t) = \bar{\boldsymbol{v}}(\boldsymbol{x}) + \boldsymbol{v}'(\boldsymbol{x}, t)$, note that although $\overline{\boldsymbol{v}'} = 0$, but $\overline{\boldsymbol{v}'\boldsymbol{v}'} \neq 0$. Thus, $-\rho \overline{\boldsymbol{v}'\boldsymbol{v}'}$ is the Reynolds stress due to the fluctuating velocity field. This nonlinear Reynolds stress term requires additional modeling to close the RANS equation and has led to the creation of many different turbulence models.

Various neural network structures are adopted in the framework of PINN. Wang et al. [164] combined the PINNs with long-short term memory (LSTM) [165] model for hydrodynamics. The LSTM has the advantage of extracting temporal features in raw data. Thus, the fusion of PINNs and LSTM further enhanced the extrapolation ability for hydrodynamics modeling. Han et al. [166] used criss-cross convolutional neural network [167] to learn the solution of parametric flow problems with spatial heterogeneity, while Cheng and Zhang [168] applied the Resnet [123] for flow modeling.

In the context of neural operators, hydrodynamics problems have become the most prevailing benchmarks. Initially fully data-driven, Li et al. [120] tested the performances of the original FNO through the 2D Darcy flow problems. Subsequently, operator learning is combined with physical equations. Zhu et al. [110] utilized the DeepONet framework, initially training it with data and subsequently fine-tuning it for specific problems based on physical equations. This process involved constructing PDEs loss functions using the automatic

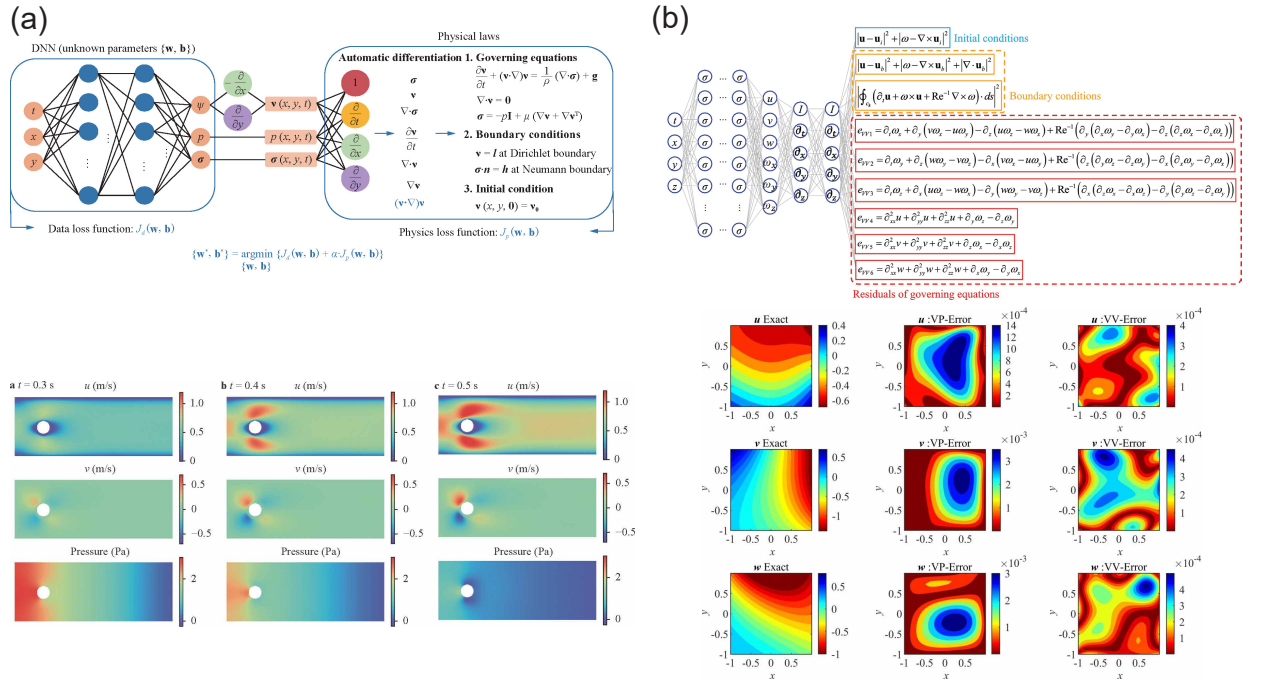


**Fig. 13.** Different Formats of PINNs for Solving Fluid Dynamics Equations: (a) VP format PINNs for incompressible laminar flow [97]. (b) VV format PINNs for incompressible Navier-Stokes equations[160].

differentiation (AD) algorithm, as shown in Fig. 14a. On the other hand, Li et al. [37] employed the Fourier Neural Operator (FNO) framework, initially training it with data and then constructing the PDEs' loss using finite differences for specific tasks, as shown in Fig. 14b. In specific fluid dynamics applications, Rosofsky et al. [169] leveraged the PINO to simulate the magnetohydrodynamics. Li and Shatarah [170] proposed a composite neural network, which comprised a series of DeepONet and a PINN. The proposed framework could not only accurately predict the velocity field and the particulate matter (PM) concentration contours by limited ground truth data, but also seamlessly coped with scenarios with different geometries.

*4.2.2. Aerodynamics and shock waves*

The NS equations are also applicable to describe the aerodynamics problems. Unlike hydrodynamics, fluids in aerodynamics are generally compressible and inviscidity. Consequently, in aerodynamics, the rheological model in Eq. (73) set the viscosity coefficients $\lambda = \mu = 0$. It is important to note that since aerodynamic fluids are compressible, $\nabla \cdot \boldsymbol{v} \neq 0$. Therefore, Eq. (69) is modified according to the compressible and inviscid properties of fluids in aerodynamics:

$$\begin{aligned} \frac{D(\rho \boldsymbol{v})}{Dt} + \rho \boldsymbol{v}(\nabla \cdot \boldsymbol{v}) &= -\nabla p + \rho \boldsymbol{f} \\ \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) &= 0 \end{aligned} . \tag{80}$$

Mao et al. [171] used Eq. (80) to model the high-speed compressible inviscid aerodynamics problems, for example, the Sod shock-tube problem, as shown Fig. 15a. The authors also discussed the strategies for selecting the positions of sample points for discontinuous problem. Instead of using FNN, Peng et al. [172] applied graph neural network (GNN) with the Euler equations to study the transient compressible fields via limited probed data. It has been shown that, once the network is well-trained, the predictions can be made within 1 ms and highly reliable explosive overpressure can be inferred. Ren et al. [173] extend PINNs to learn and forecast the steady-state aerodynamics flows around a cylinder with high Reynold numbers. Apart from the limited

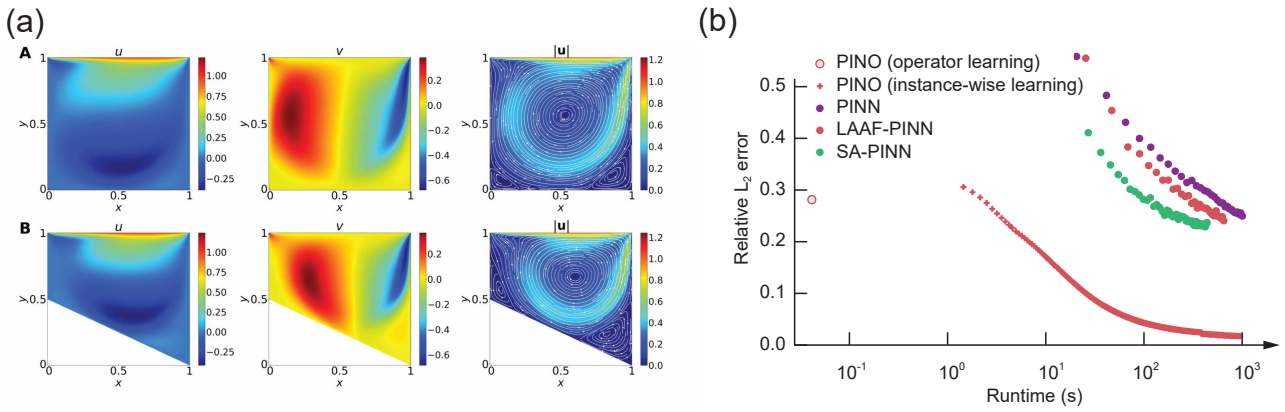


**Fig. 14.** Solving Fluid Dynamics Equations with PINO: (a) Utilizing DeepONet to provide an initial solution, followed by constructing PDEs losses using the AD algorithm [110] (b) Employing FNO to provide an initial solution, then constructing PDEs losses using finite differences [37].

measured data, the RANS equations in Eq. (79) and Eq. (80) served as additional knowledge for neural network training [173], as shown in Fig. 15b.

Other works have extended the application of PINNs to aerodynamics. Li et al. [174] utilized PINNs for predicting the gas bearing problems, including the flow field and aerodynamics characteristics. Joshi et al. [175] introduced specific weights (hyperparameters) to balance the loss terms from different equations for both low and high speeds. Auddy et al. [176] considered self-gravity of gas flows, which is essential in astrophysics.

*4.2.3. Multiphase and moving boundary problems*

The boundary tracking algorithms are also possible to be modeled by PINNs when facing multiphase and moving boundary problems. Wang and Perdikaris [96] tested the performances of PINNs with respect to simple Stefan problems where moving boundaries or free surfaces exist. In the numerical examples, the authors also presented promising results for multi phases scenarios. However, the moving and free boundaries considered in that work did not undergo any deformations. The moving boundaries have remained the same shape throughout the whole modeling. Jalili et al. [177] implemented the well-known Volume-of-Fluid (VOF) method [178], which is a prevailing boundary tracking algorithm in hydrodynamics study, in the PINNs loss function. By doing so, floating bubbles under a thermal field were modeled.

Based on the Lagrangian description, some hydrodynamics methods have been also developed. The use of the Lagrangian description makes the framework easily track free surface boundaries, which are denoted by training sample points, during the modeling. Among them, the neural particle method (NPM) is the most typical one for free surface boundaries [179]. As presented, the NPM discretized the temporal domain into pieces, while the neural network is utilized to approximate the spatial domain. Later, Bai. et al. [87] extended the framework and proposed the general neural particle method (gNPM). The flowchart of the gNPM is presented in Fig. 16a. In the gNPM, spatial derivatives and outputs of neural networks are simplified to speed up the training process. Moreover, a pressure normalization scheme is introduced into the framework so that the gNPM can produce a reliable pressure field. Due to the use of the Lagrangian particles, Shao et al. [180] integrated the alpha-shape algorithm [181] with the NPM to identify the boundary particles, enabling interactions between flow and solid boundaries or structures.

Another way to solve the moving boundaries is to transfer the Eulerian description into the Lagrangian frame [182], the transformation is given by

$$\frac{\partial}{\partial \boldsymbol{X}^L} = \frac{\partial \boldsymbol{x}^E}{\partial \boldsymbol{X}^L}\frac{\partial}{\partial \boldsymbol{x}^E}, \tag{81}$$

where $\boldsymbol{x}^E$ and $\boldsymbol{X}^L$ are the coordinates under Eulerian and Lagrangian descriptions, respectively. Besides, the distance network in Eq. (41) is applied to exactly impose the zero-pressure condition for the free surface. By

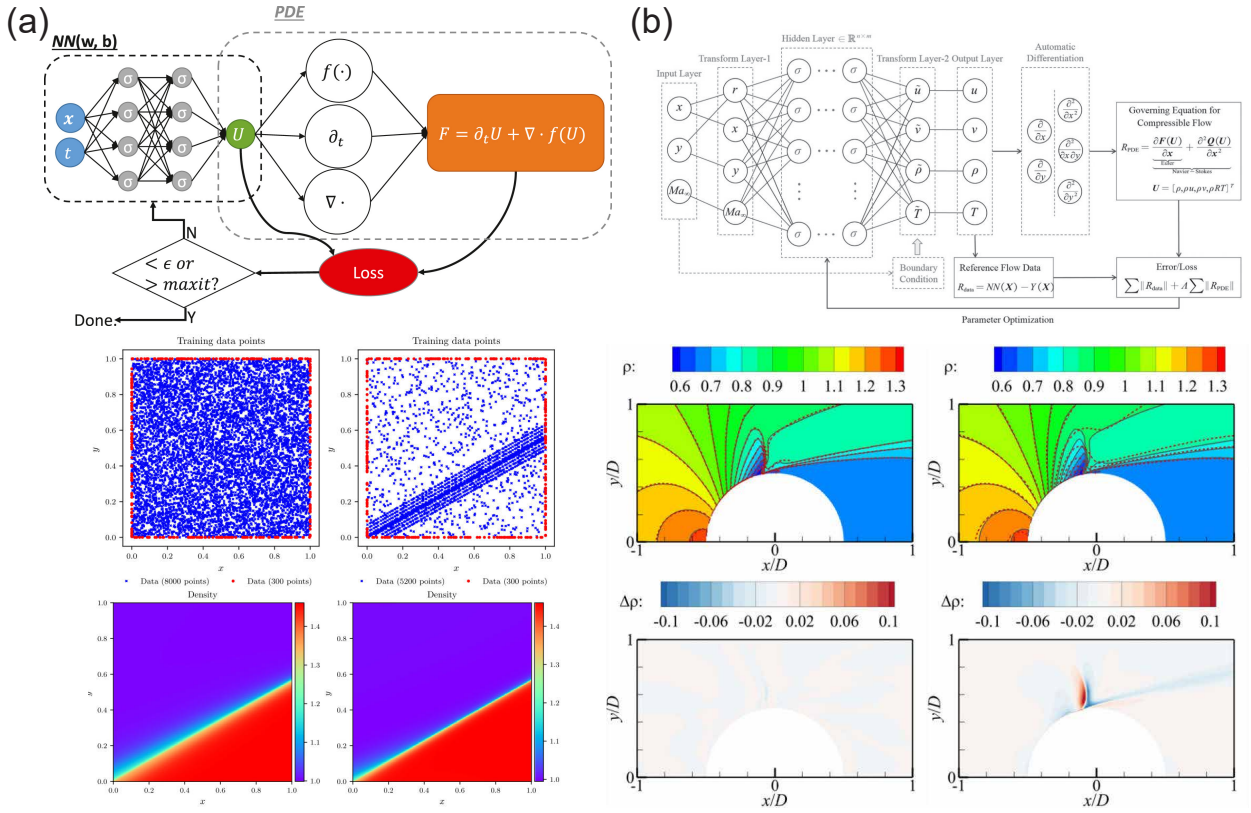


**Fig. 15.** Applications of PINNs in Aerodynamics: (a) Solving the compressible and inviscid Euler equations [171]. (b) Solving the RANS equations [173].

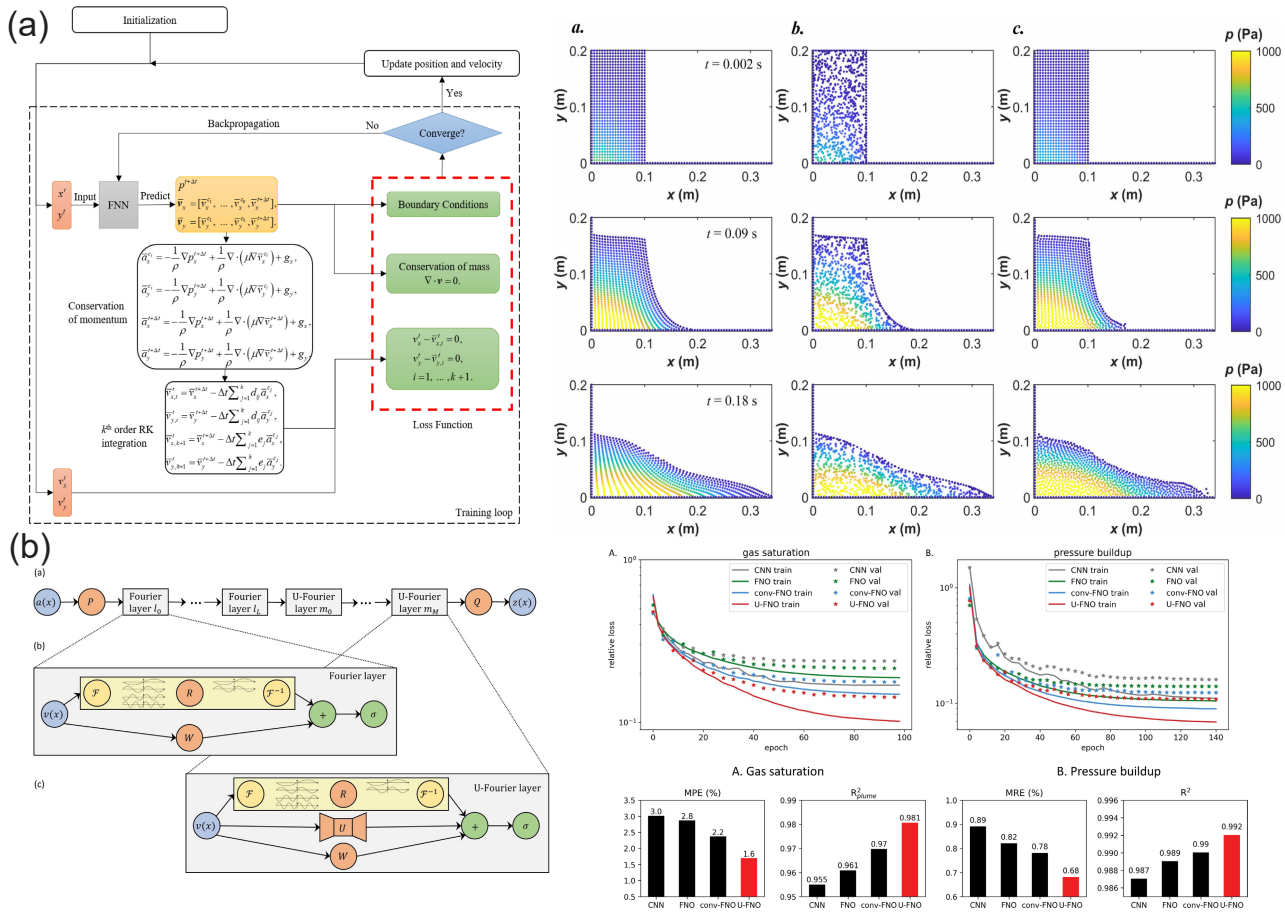


**Fig. 16.** Applications of PINNs and operator learning in multiphase and moving boundary problems: (a) The flowchart of the general neural particle method (gNPM) [87]: comparisons between the general neural particle method (gNPM) and the incompressible smoothed particle hydrodynamics (ISPH) for the 2D dam-breaking problem. (b) The schematic of the U-FNO for multiphase flow and error evolution for different algorithms [113].

exploiting the versatility of neural networks, PINNs-based algorithms are robust and stable for irregular node distributions, as shown in Fig. 16a.

Operator learning also serves as an alternative way to cope with multiphase and moving boundary problems. Wen et al. [113] integrated the U-Net structure with FNO and proposed the U-FNO model, as shown in Fig. 16b. In their work, the flow of $CO_2$ and water in the context of geological storage of $CO_2$ was considered. Compared to the traditional models of deep learning, the U-FNO exhibited excellent generalization properties and convergence rates. Furthermore, the U-FNO can achieve better accuracy when facing heterogeneous inputs. Diab et al. [183] proposed physics-informed DeepONet (PI-DeepONet) for multiphase flow in porous media. Using operator learning, the trained PI-DeepONet was capable of predicting solutions under any given flux functions (boundary conditions), which greatly alleviated the computational expense of multiphase modeling in porous media.

### *4.2.4. Multiscale and multiphysics*

The multiscale formulation also provides a different way to solve fluid mechanics. In general, the solution fields can be decomposed into the fine-scale term and the coarse-scale term. By applying the variational multiscale formulation, Hsieh and Huang [184] derived a novel physics-informed loss function based on the least squares multiscale functional form. In the case studies, the 2D and 3D advection-dominated flow problems

were conducted to demonstrate the effectiveness of the proposed loss function. Apart from using multiscale losses, Jin et al. [185] focused on neural network structures and developed the asymptotic-preserving neural network (APNN) for linear transportation equations, which decomposed the micro- and macro-variables via two independent networks, as shown in Fig. 17a.

Alongside PINNs, operator learning also sheds light on multiscale fluid modeling. Lin et al. [186] implemented the DeepONet to infer the bubble dynamics across different scales. In general, different equations and methods are required to deal with the bubble growth at different scales, for example, using the Rayleigh–Plesset (R-P) equation for the macroscale and the dissipative particle dynamics (DPD) for microscale. In the paper, the R-P and DPD modeling results were used to train different DeepONets. Through numerical tests, the authors demonstrate the possibility of using DeepONets to speed up the solving process of the bubble dynamics at different scales. Later, the framework was extended by seamlessly combining both the macroscale and microscale knowledge in a single DeepONet to predict the dynamic behaviors of bubbles at different scales [187]. It has been demonstrated that DeepONet successfully learned the mapping operators for bubble growth dynamics.

More importantly, the DeepONet offered a novel way to form a unified modeling method across scales instead of using traditional multiscale modeling approaches. Cai et al. [188] and Mao et al. [189] proposed the DeepM&Mnet, where the "M&M" denotes the words "multiphysics" and "multiscale" based on the operator learning technique. The proposed DeepM&Mnet comprised numbers of pre-trained DeepONets, as shown in Fig. 17b. In Mao's framework, the DeepM&Mnet was developed for hypersonic flow problems with shocks, which may involve sharp density gradients spanning across various magnitudes. By preparing data in terms of flow velocity and temperature under scenarios whose densities span eight orders of magnitude downstream of the shock, the DeepM&Mnet accurately and efficiently predicted velocity and density fields. To ensure the accuracy of the wide range of velocity fields, the mean absolute percentage error (MAPE) was implemented in the loss function.

Wu et al. [190] proposed two convolutional DeepONets to deal with the linear transportation problem. Ahmed and Stinis [191] developed the multi-fidelity operator network (MFON), where an auxiliary term that comes from the Galerkin proper orthogonal decomposition was added to a DeepONet. The effectiveness of the proposed MFON was then demonstrated via a viscous Burgers equation problem and a 2D vortex merger problem.

#### *4.2.5. Summary*

The same as aforementioned in solid mechanics, PINNs is currently still in its infancy as a forward problem solver in computational fluid mechanics. The computational efficiency still impedes the use of PINNs on many problems of interest. Besides, the variational loss is not commonly seen in the context of hydrodynamics problems, while the strong form loss function still plays a vital role. As it is well known, with an increasing number of loss terms, more challenges and difficulties will be encountered during the training process. Therefore, training robustness also remains a huge issue that hinders the applications of PINNs in the context of fluid modeling. However, the great potential of PINNs has been found as an effective tool for inverse problems in fluid mechanics, such as field reconstruction and essential parameter estimation. The above PINNs frameworks for forward problems can be applied to inverse studies with small modifications in fluid mechanics. More details regarding the PINNs for inverse problems can be found in Section 5.2.

On the contrary, operator learning serves as a surrogate modeling scheme for fluid flows. Despite its data-driven nature, physics knowledge in terms of governing equations can also guide the learning process, such as PINO. It has shown excellent generalization properties for predicting fluid flow problems even with sharp gradients and complex geometries. Moreover, once the neural operator is trained, the neural operator can be reused for different boundary conditions. This is an extraordinary advantage compared to PINNs, because a PINNs is generally applicable for one set of boundary conditions. Re-training is necessary for a PINN if the boundary conditions of the target fluid problem are changed. Another advantage of operator learning is its discretization-invariant properties. In fact, great efforts have been witnessed in fluid modeling enhanced by operator learning, trying to reveal details of fluid fields via low-resolution results. By using operator learning, it is straightforward and effective to obtain super-resolution results.

In the future, uncertainties, which are crucial in practical applications, should be effectively incorporated into or considered within AI for PDEs (AI4PDEs).

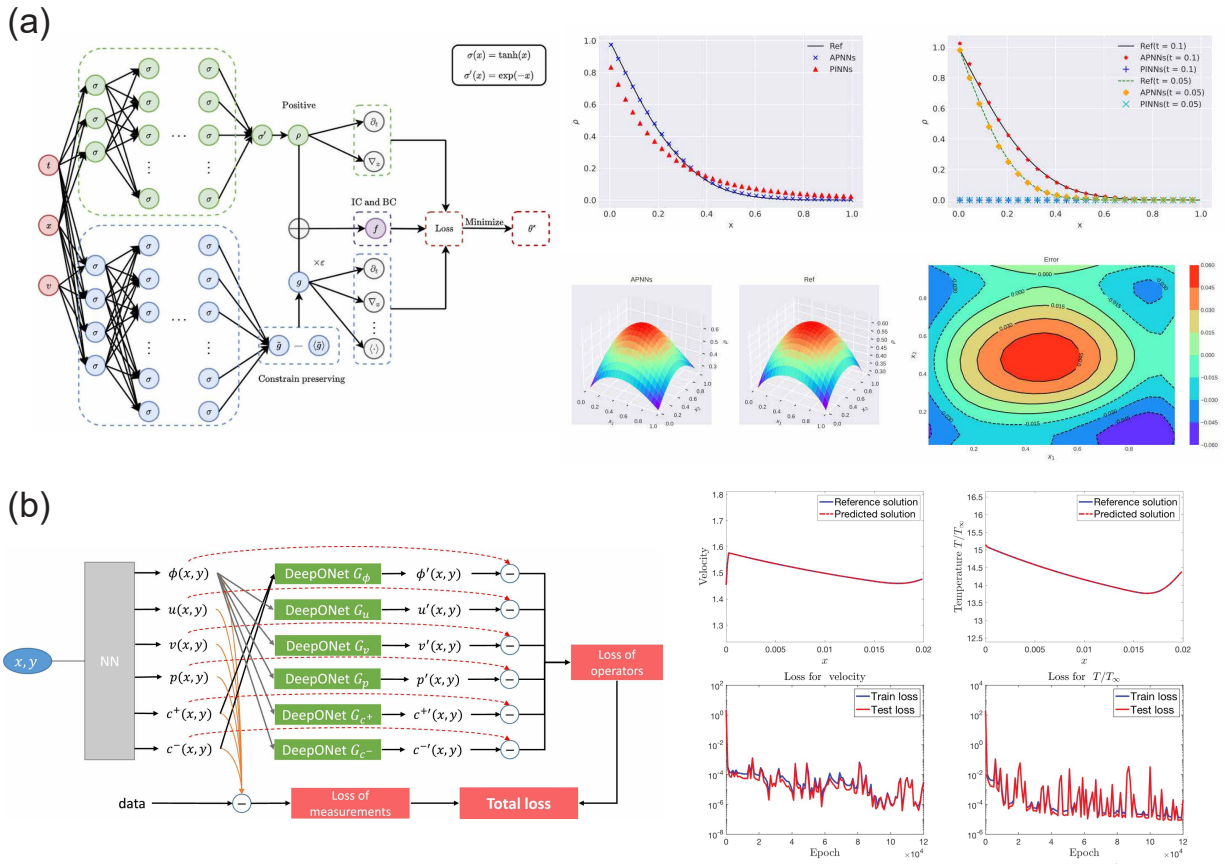


**Fig. 17.** Applications of PINNs and PINOs in multiscale and multiphysics: (a) Multiscale PINNs for linear transportation equations. Two neural networks are designed for micro- and macro-variables, respectively [185]. (b) The multiscale physics-informed DeepONets for multiscale fluid modeling problems [188, 189].

### *4.3. Biomechanics*

#### *4.3.1. Soft tissue deformation*

Computational models of soft tissue mechanics have the potential to provide valuable patient-specific diagnostic insights. However, their clinical deployment has been limited due to the high computational costs associated with traditional numerical solvers for biomechanical simulations. PINNs can be used to simulate the deformation of soft tissues, such as skin or organs, under various loads or boundary conditions [192]. This capability is critical for understanding and predicting tissue behavior in different scenarios.

Buoso et al. [108] introduce a novel approach to generate realistic numerical phantoms of the left ventricle by combining statistical shape learning, biophysical simulations, and tissue texture generation, as shown in Fig. 18a. This study addresses the limitations of previous in-silico phantoms by increasing variability and realism. The generated data can be used for standardized performance assessment of cardiovascular magnetic resonance acquisition, reconstruction, and processing methods.

PINNs based on graph convolutional networks (GCNs) and the variational principle has also been proposed, which can unify the solving of PDE-governed forward and inverse problems [75]. Then, Dalton et al. developed an efficient emulation of soft tissue mechanics using the PINNs based on graph neural networks (GNNs) [109], as shown in Fig. 18b. GNNs can handle the unique geometry of a patient's soft tissue without requiring low-order approximations. The physics-informed training approach simulates the soft tissue mechanical behavior. With PDEs, the approach based on PINNs accurately predicts the deformation field of the liver, left ventricle, and brain models under various loading conditions. PINNs based on GNNs are also used to solve brain mechanics, aiding in understanding the microstructure-mechanics relationship in human brain tissue [193].

We found that the approach to using PINNs for simulating soft tissue problems is similar to that of hyperelastic problems in solid mechanics, where the minimum potential energy function is used as the loss function.

#### *4.3.2. Blood flow of biomechanics*

Computational modeling of blood flow and cardiovascular dynamics presents significant challenges due to the complexity of the underlying physics and patient-specific geometries. PINNs integrate PDEs and data to provide a novel approach for modeling blood flow in biomechanics. This capability has significant implications for cardiovascular disease diagnosis, treatment planning, and personalized medicine.

Sun et al. [95] developed a physics-constrained deep learning approach shown in Fig. 19a for surrogate modeling of fluid flows without relying on any simulation data, making it suitable for parametric fluid dynamics problems where data is often scarce. Their numerical experiments on various internal flow problems relevant to hemodynamics demonstrate the potential of this approach. Additionally, PINNs has been used to model blood flow in patient-specific geometries by incorporating clinical data, providing insights into complex cardiovascular dynamics [194].

The applications of AI for PDEs in various cardiovascular imaging modalities, including echocardiography, cardiac computed tomography (CT), cardiovascular magnetic resonance imaging (CMR), and imaging in the catheterization laboratory, have also been explored, demonstrating the versatility of AI for PDEs [195]. Furthermore, various neural network architectures, such as fully connected networks, Fourier networks, and multiplicative filter networks have been employed to model 3D blood flows using PINNs, showcasing the potential for efficient and accurate modeling of complex blood flow shown in Fig. 19b [196].

Moreover, optimization techniques, such as variable-separated physics-informed neural networks based on adaptive weighted loss functions (AW-vsPINN), have been developed to apply to blood flow models in arteries. These methods decompose the blood flow problem into simpler sub-problems, reducing complexity and improving the efficiency of the neural network training process [197].

We have found that the simulation of blood flow in biomechanics closely aligns with the governing equations used in fluid mechanics. Therefore, the study of blood flow in biomechanics closely resembles the application of fluid mechanics to biological vascular flows.

#### *4.3.3. Morphogenesis of tissue and cell deformation*

Understanding how cells orchestrate their functions and self-organize into different patterns is a crucial question in developmental biology and tissue engineering. Traditional approaches, such as purely physics-based

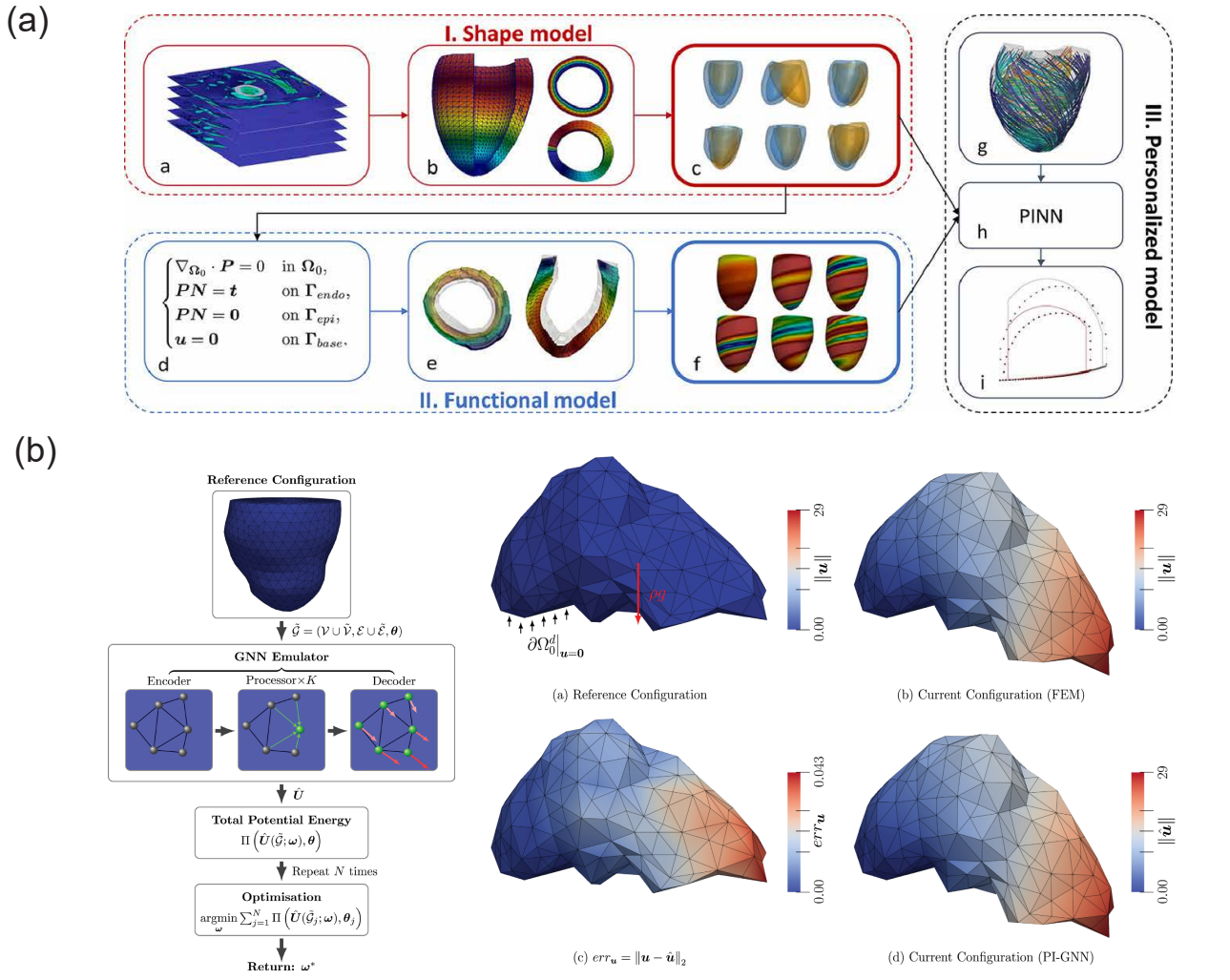


**Fig. 18.** Applications of PINNs in soft tissue: (a) three main blocks of the framework: generation of the shape model (I), generation of the functional model (II), and definition of the personalized biophysical left ventricular model [108]. (b) The schematic of a physics-informed GNN emulator and emulation results for Liver and Left Ventricle model [109].

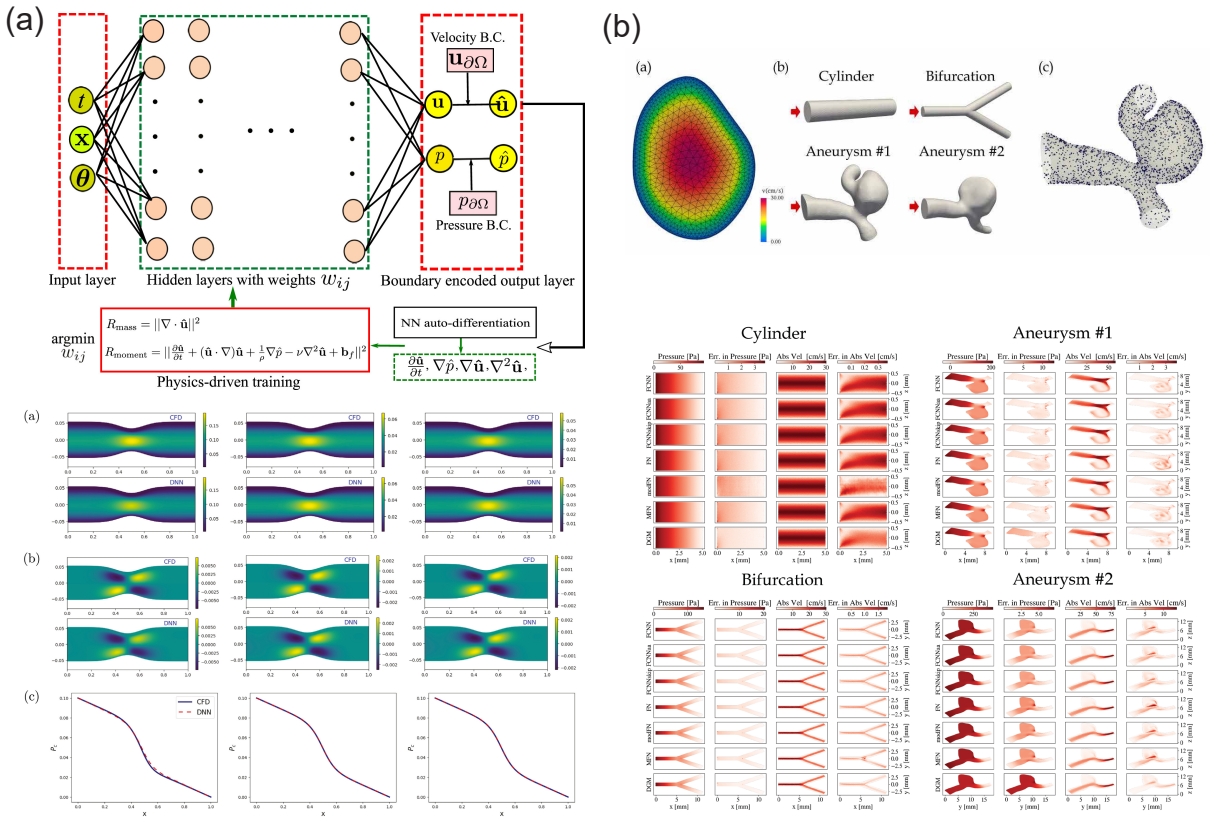


**Fig. 19.** Applications of PINNs in blood flow of biomechanics: (a) The PINNs framework for surrogate modeling of blood flows, and results of PINNs and CFD solutions of idealized stenotic flows at different viscosity parameters [95]. (b) Physics-constrained prediction of idealized stenotic flows and aneurysmal flows of different geometries, the simulation geometries included two idealized structures (cylinder, bifurcation), and two aneurysms that were segmented from brain digital subtraction angiography (DSA) images [196].

models or data-driven models have limitations in capturing the complex interplay between mechanical interactions, cell functions, and decision-making processes. Integrating physics-based models with data-driven models has the potential to advance our understanding of complex biological processes, such as organoid development, tissue engineering, and disease modeling.

Camacho-Gomez et al. [198] propose a hybrid framework that combines PDEs with data to study the morphogenesis of organoids, specifically murine pancreatic ductal adenocarcinoma (PDAC) cells grown as tumor organoids in 3D Matrigel cultures. This model unravels the principles through which cells activate different processes and self-organize in response to their microenvironment. In tissue development, Bayesian inference of agent-based cellular automaton models has been employed to study kidney branching morphogenesis, demonstrating how such AI for PDEs can provide insights into developmental processes [199].

Cavanagh et al. [200] introduce a novel approach combining physics-informed deep learning with experimental data to study the morphodynamics of Asian soybean rust disease, as shown in Fig. 20a. This framework integrates experimental data with a reaction-diffusion model describing the growth and morphological dynamics of the fungal pathogen, providing insights into the underlying mechanisms driving the disease's spread and informing effective management strategies. The trained model can accurately predict the spatiotemporal evolution of fungal morphology, including the formation of lesions, sporulation patterns, and the spread of the disease across the leaf surface. By combining experimental data with physics-informed modeling, the study provides insights into the underlying mechanisms driving the morphological dynamics of the fungal pathogen, such as the interplay between nutrient diffusion, hyphal growth, and sporulation. Thus, AI for PDEs help understand the morphological dynamics and develop effective disease management strategies.

Drying is a crucial process in the food industry, and understanding the underlying mechanisms of moisture transfer and shrinkage in plant cells is essential for optimizing drying processes and preserving product quality. Batuwatta-Gamage et al. [201] propose a novel approach that combines PINNs with experimental data to study the complex phenomena of moisture concentration and shrinkage during the drying of plant cells, as shown in Fig. 20b. Their framework couples two distinct neural networks: PINNs-MC, which predicts moisture concentration based on Fick's law of diffusion, and PINNs-S, which predicts shrinkage using the 'free shrinkage' hypothesis. By incorporating the governing physics of moisture transfer and shrinkage, this approach enables accurate predictions of both moisture concentration and shrinkage dynamics during the drying process. The framework demonstrates higher accuracy and stability in predicting moisture concentration, even for unknown spatiotemporal domains and varying drying process parameters. This dual PINNs approach offers insights into the mechanisms driving moisture transfer and shrinkage, facilitating a better understanding and optimization of the drying process in the food industry.

#### *4.3.4. Summary*

AI for PDEs is a powerful tool for addressing the forward problem in biomechanics. It has been applied to various biomechanical issues such as tissue deformation, blood flow simulation, and morphodynamics. By enhancing our understanding of these phenomena, AI for PDEs can aid in developing new treatments for diseases like cardiopathy and other cardiovascular conditions. By incorporating the governing equations of biomechanics into neural networks, AI for PDEs can provide more accurate predictions of biological tissue behavior. With proper training, it can solve complex biomechanical problems more efficiently than traditional numerical methods, which are often computationally expensive. Moreover, AI for PDEs can model a wide range of biomechanical phenomena, from simple to complex systems, thereby broadening their applicability in the field.

Biomechanics employs methodologies from both solid and fluid mechanics, such as the use of hyperelastic constitutive equations from solid mechanics. Therefore, the application of AI for PDEs in biomechanics shares similarities with both solid and fluid mechanics. However, solving biomechanical problems requires extra consideration of biochemical factors such as nutrient diffusion, growth factor concentration, and gene expression.

Traditional modeling approaches often rely on simplified assumptions and conditions that struggle to capture the complexity of morphogenetic processes. However, recent advances in AI for PDEs offer promising avenues for integrating physical principles with data-driven approaches. In the future, AI for PDEs has the potential to combine multiscale data and bridge the gap between molecular, cellular, and tissue-level processes. AI for PDEs could be instrumental in uncovering missing or unknown physical relationships and mechanisms. By integrating

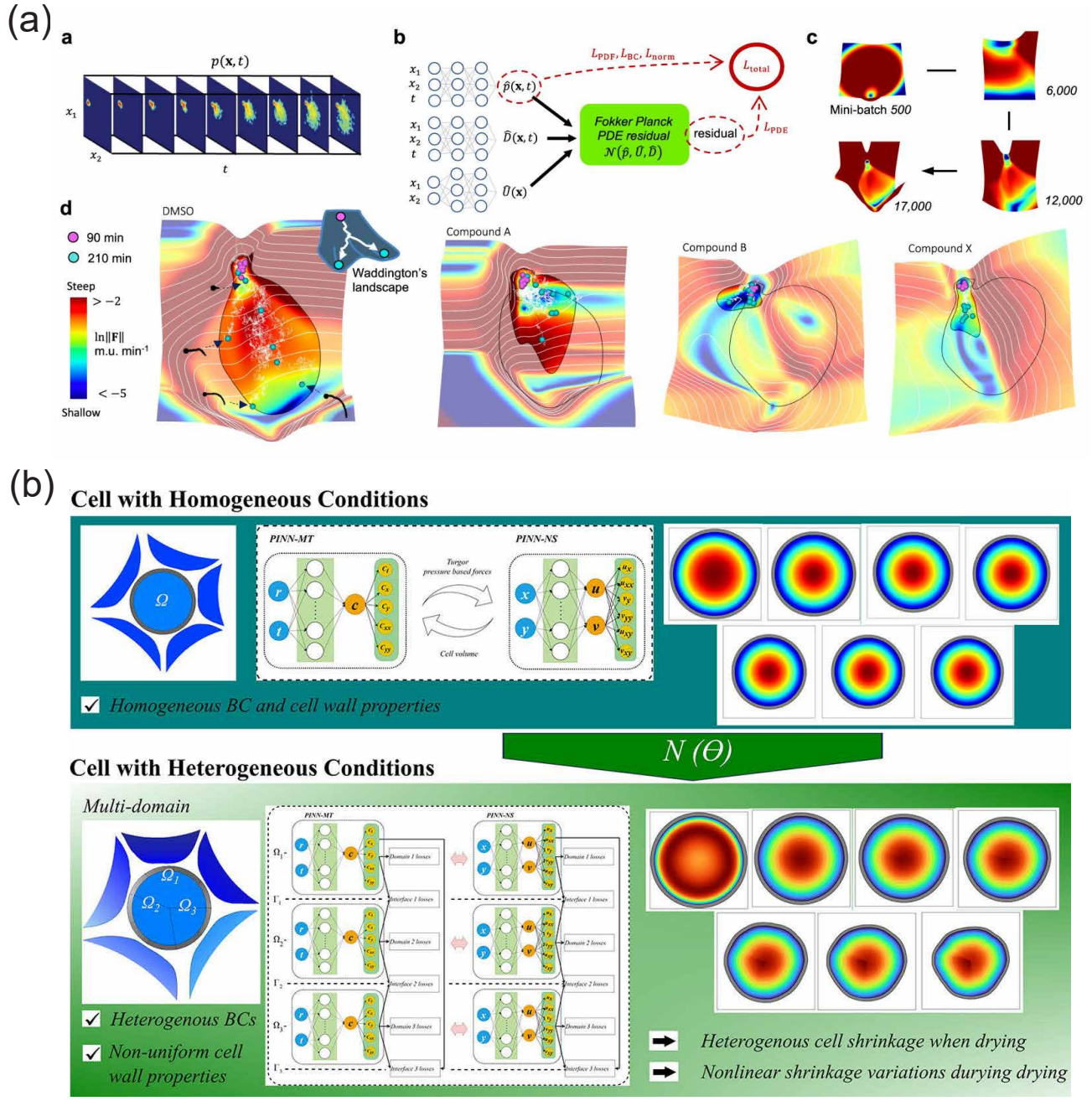


**Fig. 20.** Applications of PINOs in morphogenesis of tissue and cell deformation: (a) Morphodynamic landscapes [200]. (b) multi-domain analysis of the single plant cell [201].

PDEs with experimental data, we can potentially gain new insights into the intricate interplay of multiscale processes in biomechanics.

## 5. AI for PDEs in the application of inverse problems of computational mechanics

### *5.1. Solid mechanics*

#### *5.1.1. Identification of Elastic Modulus and Poisson's Ratio*

Haghighat et al. (2021) [26] were the first to propose the use of PINNs to solve inverse problems in solid mechanics, addressing both elastic and elastoplastic issues, as illustrated in Fig. 21a. In elasticity, the approach starts by gathering data through analytical solutions or high-precision numerical solutions. Multiple neural networks are used to fit multiple variables (displacement and stress), and a loss function is constructed using data loss, equilibrium equations, and constitutive equations. We consider the problem of two-dimensional plane stress to explain the idea:

$$\begin{aligned}\mathcal{L} = &|\hat{u}_x - u_x^*| + |\hat{u}_y - u_y^*| + |\hat{\sigma}_{xx} - \sigma_{xx}^*| + |\hat{\sigma}_{yy} - \sigma_{yy}^*| + |\hat{\sigma}_{xy} - \sigma_{xy}^*| \\ &+ |\sigma_{xx,x} + \sigma_{xy,y} - f_x^*| + |\sigma_{yx,x} + \sigma_{yy,y} - f_y^*| \\ &+ |(\lambda + 2G)\varepsilon_{xx} + \lambda\varepsilon_{yy} - \sigma_{xx}| + |(\lambda + 2G)\varepsilon_{yy} + \lambda\varepsilon_{xx} - \sigma_{yy}| + |2G\varepsilon_{xy} - \sigma_{xy}|\end{aligned} \tag{82}$$

where the superscript $*$ indicates the given data, and $\lambda$ and $G$ are Lame constants (set as the inverse problem variables to be optimized). The least squares $|\cdot|$ is used for fitting, and $\lambda$ and $G$ are obtained by optimization using the given displacement, stress, and body force fields.

For elastoplastic mechanics, the approach is similar, although the equations change to J2 flow theory for plane strain conditions. According to the J2 flow theory, we add plasticity multiplier $\gamma$ and the KKT conditions for convex optimization:

$$\begin{aligned}\mathcal{L} = &|\hat{u}_x - u_x^*| + |\hat{u}_y - u_y^*| + |\hat{\sigma}_{xx} - \sigma_{xx}^*| + |\hat{\sigma}_{yy} - \sigma_{yy}^*| + |\hat{\sigma}_{xy} - \sigma_{xy}^*| + |\hat{\sigma}_{zz} - \sigma_{zz}^*| \\ &+ |\sigma_{xx,x} + \sigma_{xy,y} - f_x^*| + |\sigma_{yx,x} + \sigma_{yy,y} - f_y^*| \\ &+ |(\lambda + \frac{2}{3}G)\varepsilon_{kk} + 2G(\varepsilon_{xx} - \varepsilon_{xx}^{*p}) - \sigma_{xx}| + |(\lambda + \frac{2}{3}G)\varepsilon_{kk} + 2G(\varepsilon_{yy} - \varepsilon_{yy}^{*p}) - \sigma_{yy}|, \\ &+ |(\lambda + \frac{2}{3}G)\varepsilon_{kk} + 2G(\varepsilon_{zz} - \varepsilon_{zz}^{*p}) - \sigma_{zz}| + |2G(\varepsilon_{xy} - \varepsilon_{xy}^{*p}) - \sigma_{xy}| \\ &+ |(\bar{\varepsilon} - \frac{\sigma_s}{3G}) - \bar{\varepsilon}^{*p}| + |[1 - \mathrm{sign}(\bar{\varepsilon}^{*p})]|\bar{\varepsilon}^{*p}|| + |(1 + \mathrm{sign}(\mathcal{F}))|\mathcal{F}|| + |\bar{\varepsilon}^{*p}\mathcal{F}|\end{aligned} \tag{83}$$

where the first line represents data loss (all variables marked with $*$ are given beforehand), the second line represents the equilibrium equations, the third and fourth lines are the constitutive equations with the superscript $p$ indicating the plastic part, and the last line includes the KKT conditions. The first item in the KKT conditions is the J2 theory plasticity multiplier $\gamma = \bar{\varepsilon}^{*p} = \bar{\varepsilon} - \sigma_s/(3G)$, $\bar{\varepsilon} = \sqrt{2\varepsilon_{ij}\varepsilon_{ij}/3}$, and $\bar{\varepsilon}^p = \sqrt{2\varepsilon_{ij}^p\varepsilon_{ij}^p/3}$. $\mathcal{F}$ is the yield surface function, defined by J2 theory as $\mathcal{F} = \sqrt{3s_{ij}s_{ij}/2} - \sigma_s$. sign is as follows:

$$\mathrm{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases} \tag{84}$$

Clearly, Eq. (83) encompasses all the PDEs described by J2 theory for elastoplastic mechanics. Hence, optimizing Eq. (83) involves fitting displacement and stress using multiple neural networks (similar to Eq. (82)), then setting $\lambda$, $G$, and $\sigma_s$ as optimization variables for solving the inverse problem. If a theory other than J2 is used, the method remains the same, though the specific PDEs would differ.

For isotropic but heterogeneous materials, Chen et al. (2021) [202] suggest identifying the elastic field in isotropic and inhomogeneous materials as shown in Fig. 21b. However, instead of mapping coordinates to displacement fields, it maps coordinates to elastic modulus fields. The strain field is calculated from the given

displacement field using kinematic equations, and the elastic field is then computed at the coordinates of the strain fields using neural networks. Stresses are subsequently calculated using isotropic elasticity constitutive relations, and these stresses are used to derive partial differential fields through convolutional kernel differentiation. The final loss function is established based on equilibrium equations, and it's optimized to tune the neural network parameters in the elastic modulus field. It's critical to note that merely using equilibrium equations is insufficient. Theoretically, we must also incorporate force boundary conditions or an average modulus of elasticity. This average modulus is used to calibrate the learned modulus back to realistic conditions because the modulus field learned solely from equilibrium equations can just represent a pattern, whose amplitude is undetermined. Thus, an average modulus is required to realign it with reality, assuming prior knowledge of the material's properties. If there is no prior knowledge, applying force boundary conditions is necessary. It is important to emphasize that these force boundary conditions are not only applied at the boundaries but can also be implemented in internal domains if the loading conditions are special. Details can be referred to [202]. Unlike learning a fixed modulus, this neural network mapping establishes a link from coordinates to the elastic modulus field, not merely as optimization variables. Additionally, learning physical equations through convolutional neural networks' kernel functions is feasible by setting the elastic modulus and strain fields in advance. There are also approaches that learn PDEs. Steven et al. [203] proposed SINDy, which discovers governing equations from data for nonlinear dynamical systems.

Furthermore, Liu et al. (2024) [204] presented a similar idea for solving the inverse problem of thermal conductivity for isotropic and heterogeneous materials. They fit temperature data using Data Net and then used PDEs Net, based on PINNs, to solve for non-uniform thermal conductivity, as depicted in Fig. 21c. This method combines direct data fitting with physical laws to refine the prediction of material properties.

### *5.1.2. Identification of Constitutive Equations*

The identification of constitutive equations remains a core issue in solid mechanics, primarily focusing on the relationship between stress and strain. Traditional methods usually involve defining a specific form of the constitutive equation and then determining the unknown parameters of the model based on experimental results (displacement fields and external load forces). Li et al. (2022) [205] proposed using a neural network to replace the constitutive equation, where the relationship from strain to stress is determined by the parameters of the neural network, as shown in Fig. 22a. The loss function is defined as:

$$\begin{aligned} \mathcal{L} &= \int_{\Omega} |\sigma_{ij,j} + f_i|^2 d\Omega + \sum_{s=1} \beta^s \Big| \int_{\Gamma^{t_s}} (\sigma_{ij} n_j) d\Gamma - \boldsymbol{T}^s \Big|^2 , \\ \sigma_{ij} &= NN(\varepsilon_{ij}; \boldsymbol{\theta}) \end{aligned} \tag{85}$$

where $\boldsymbol{T}^s$ is the total force for the s-th force boundary $\Gamma^{t_s}$, and $\boldsymbol{T}^s$ can be easily obtained from universal testing machines for the corresponding load. Additionally, the displacement field under $\boldsymbol{T}^s$ can be captured using Digital Image Correlation (DIC), and then the strain is calculated using kinematic equations and inputted into the neural network $NN(\varepsilon_{ij}; \boldsymbol{\theta})$ to obtain the corresponding stress. The stress field is used to establish the loss of internal balance equation and the force boundary conditions. The loss about force boundary conditions integrates over the force boundary $\Gamma^{t_s}$ using experimental load data. Note that non-homogeneous force boundary conditions are essential. If only displacement data are available without force boundary conditions, it is impossible to determine the constitutive equation theoretically. Fig. 22a outlines a non-parametric hyperelastic constitutive model using a neural network, but a parametric model as shown in Fig. 22b can also be established. For example, the form of the constitutive equation can be pre-determined (e.g., the HGO model for biomaterials by [192]), considering all possible base functions in the constitutive equation [206], and then using optimization techniques to determine the parameters ahead of these base functions to define the form of the constitutive equation.

It is important to note that both parametric and non-parametric models require non-homogeneous force boundary conditions. The essential difference between these implementation methods lies in the form of $\sigma_{ij} = NN(\varepsilon_{ij}; \boldsymbol{\theta})$ in Eq. (85), where the non-parametric model uses a neural network and the parametric model uses a predefined function. Currently, AI for PDEs applications in identifying constitutive equations are primarily focused on hyperelastic materials, with fewer studies on history- and path-dependent elastoplastic

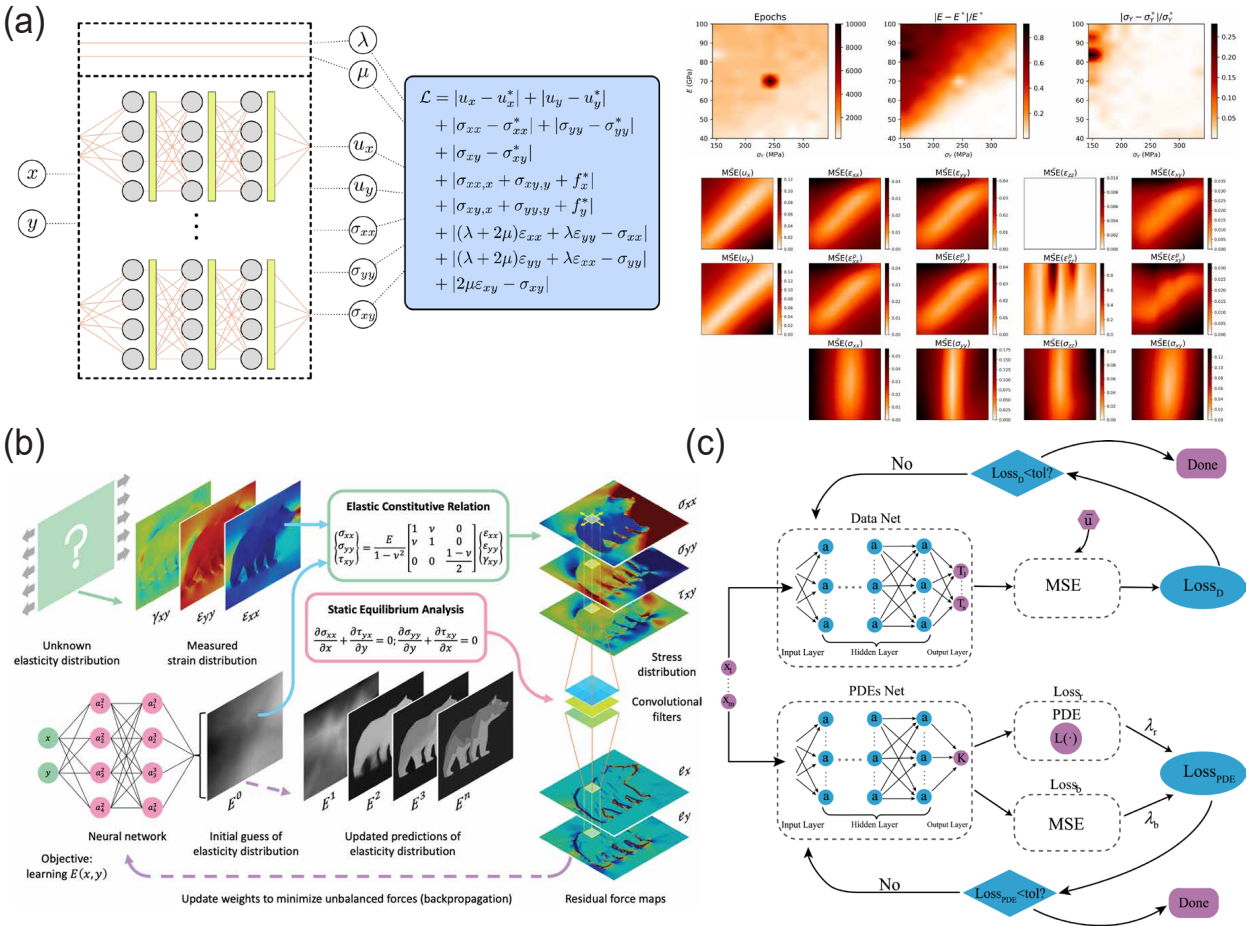


**Fig. 21.** Applications of PINNs in identifying elastic modulus and poisson's ratio: (a) Identification of elastic modulus, Poisson's ratio, and yield stress in homogeneous and isotropic elastoplastic materials [26], (b) Identification of the elastic modulus field in non-uniform isotropic materials [202], (c) Identification of thermal conductivity in non-uniform isotropic materials [204].

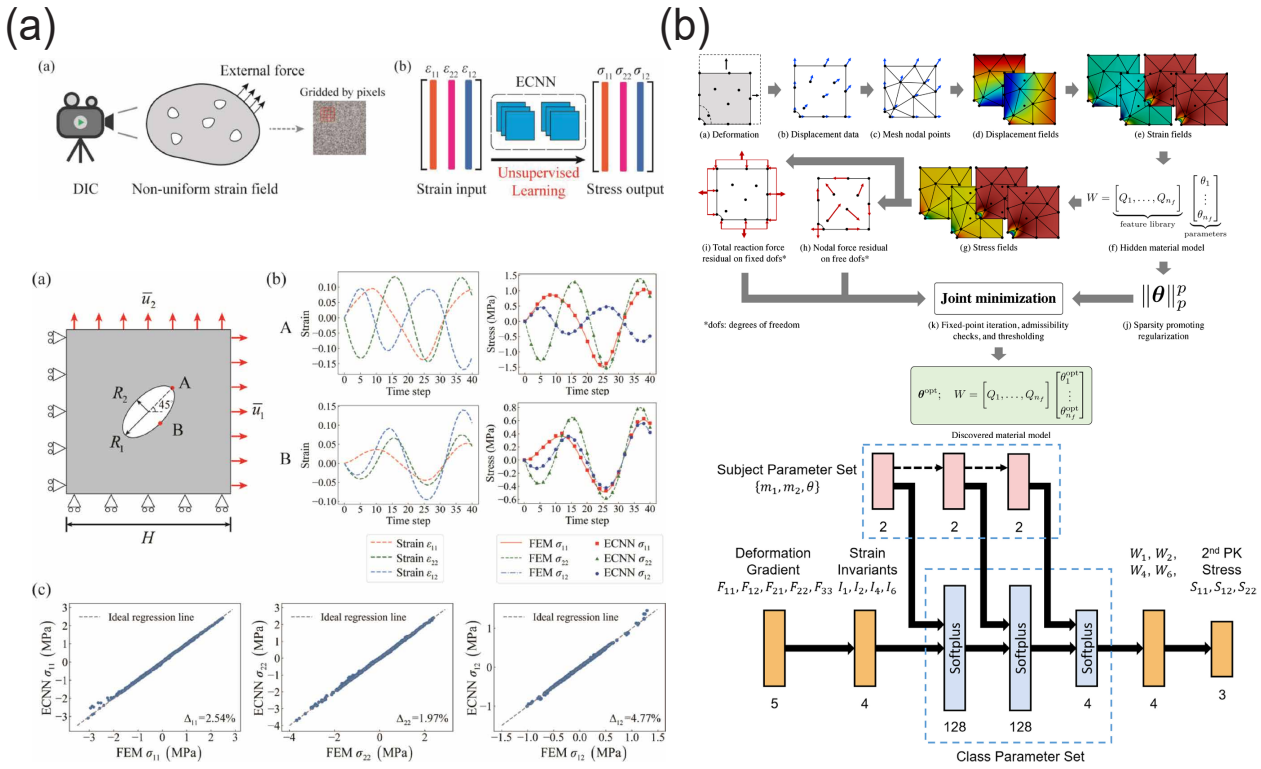


**Fig. 22.** Applications of AI for PDEs in Constitutive Equation Identification: (a) Non-parametric model: neural network fitting of strain and stress relationships in hyperelastic materials [205], (b) Parametric model: predefined hyperelastic constitutive forms with fitting of preceding parameters [192, 206].

materials, and rate-dependent viscoelastic materials. Future research using neural networks to replace constitutive equations will gradually address such complex materials requiring solid constitutive theory foundations to establish physically meaningful neural network constitutive models. We believe this research area holds significant potential and prospects because constitutive equations primarily involve fitting, and neural networks inherently possess robust fitting capabilities.

Additionally, Kirchdoerfer et al. (2016) [207] proposed a method that does not require a specific form of constitutive modeling. They used experimental stress-strain data points directly and minimized a distance functional to find the closest known stress-strain state to the current material state. Since their method did not use neural networks, we refer to [207] and [208] for more information.

### *5.1.3. Topology Optimization*

Topology optimization is a classic inverse problem. It modifies the material distribution based on a predefined objective function, such as minimizing compliance. Typically, changing the material distribution requires sensitivity analysis, which mathematically involves deriving the objective function with respect to the density field (the physical representation of material distribution). Notably, the objective function alone is insufficient without the stress-deformation state under the current density field, which can be mathematically expressed as:

$$\begin{aligned} \text{Objective Function (Minimize Compliance):}\ & \min_{\rho} \mathcal{F} = \frac{1}{2}\int_{\Omega} \boldsymbol{\sigma}^* : \boldsymbol{\varepsilon} d\Omega \\ \text{Material Constraint:}\ & \int_{\Omega} \rho(\boldsymbol{x}) d\Omega = \bar{V} \end{aligned}, \tag{86}$$

where $\rho$ is the density field (material distribution), a scalar field function ranging from 0 to 1. $\sigma^*$ is the stress field adjusted according to the density field $\rho$, dictating the material degradation, typically set as $\boldsymbol{\sigma}^*(\boldsymbol{x}) = \rho(\boldsymbol{x})^r * \boldsymbol{\sigma}(\boldsymbol{x})$, where $r$ is a hyperparameter often set to 3. $\boldsymbol{\sigma}(\boldsymbol{x})$ is determined by substituting the displacement field $\boldsymbol{u}$ into the kinematic equation and then through the constitutive relation. For linear elastic problems, the original stress field is modified to $\boldsymbol{\sigma}^*(\boldsymbol{x})$ as expressed mathematically:

$$\begin{aligned} \text{Minimum Potential Energy Principle:}\ & \min_{\boldsymbol{u}} \mathcal{L} = \int_{\Omega} \frac{1}{2}\boldsymbol{\sigma}^* : \boldsymbol{\varepsilon} d\Omega - \int_{\Omega} \boldsymbol{f} \cdot \boldsymbol{u} d\Omega - \int_{\Gamma^t} \bar{\boldsymbol{t}} \cdot \boldsymbol{u} d\Gamma \\ \text{Displacement Constraint:}\ & \boldsymbol{u}(\boldsymbol{x}) = \bar{\boldsymbol{u}}(\boldsymbol{x}), \boldsymbol{x} \in \Gamma^u \end{aligned}, \tag{87}$$

where $\boldsymbol{\sigma}^*(\boldsymbol{x})$ is derived from the displacement field and the current density field, with the constraint that the displacement field satisfies essential boundary ($\Gamma^u$) conditions. Eq. (86) and Eq. (87) represent optimization problems for obtaining the density and displacement fields, respectively. These two optimization problems differ in that each update to the density field in Eq. (86) through gradient descent, $\rho^* = \rho - \alpha \partial \mathcal{F} / \partial \rho$, necessitates a complete optimization of the potential energy $\mathcal{L}$. Thus, mathematically, it is a coupled optimization problem under material and displacement constraints. While compliance optimization can rely on traditional optimization algorithms, the displacement field can be obtained using solvers like the traditional finite element method.

Recent studies have proposed using DEM to replace compliance and potential energy optimization. He et al. (2023) [107] adopted the idea from Zehnder et al. (2021) [209] to replace the finite element calculation with the DEM method for minimum potential energy, but the target compliance minimization function is still calculated using traditional methods like the moving asymptote method, as illustrated in Fig. 23a. Subsequently, Jeong et al. (2023) [210] also replaced the traditional algorithms with an idea similar to DEM for target compliance minimization function, establishing a fully connected neural network function mapping coordinates to the density field. Compliance obtained from the displacement field and the density field is then optimized once, updating the neural network parameters of the density field and this optimized density field is re-entered into the DEM to compute the displacement field, repeating until convergence, as shown in Fig. 23b. It is noteworthy that the combination of DEM and topology optimization centralizes some optimization problems using neural network optimization algorithms instead of traditional methods, such as using PINNs or DEM to replace finite element calculations for PDEs in solid mechanics. With target optimization functions solved using PINNs methods [210], traditional SIMP [211], or traditional MMA [107], different algorithms for solving target optimization

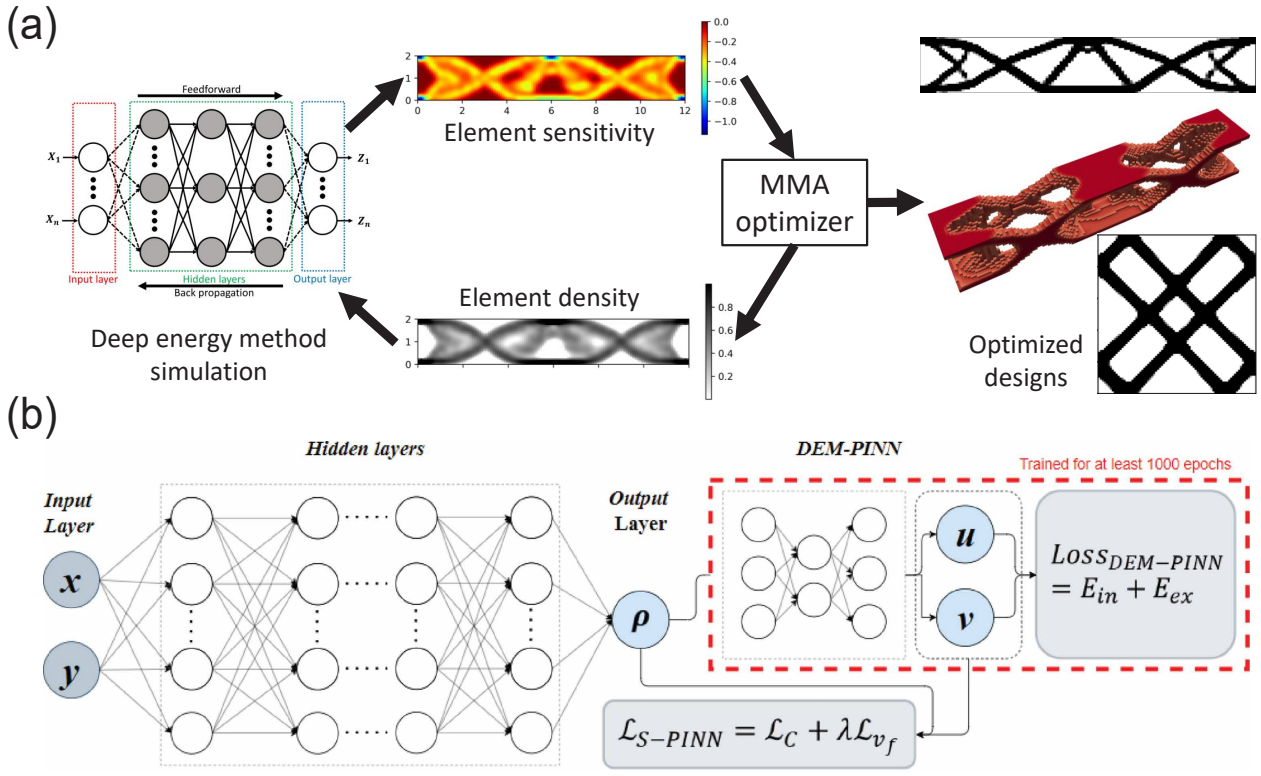


**Fig. 23.** Applications of DEM in topology optimization: (a) Traditional methods such as the moving asymptote method is used to iterate the density field for compliance minimization, and DEM is for the displacement field [107]. (b) DEM is used for both the density and displacement field [210]

functions and different methods for solving mechanical displacement fields form various works combining deep learning and topology optimization.

Since the bulk of the computational effort comes from continually recalculating the displacement field for updated geometric topologies, using operator learning to accelerate the computation of the forward problem in topology optimization has significant potential to enhance the total speed of topology optimization.

### *5.1.4. Defect Identification*

Defect identification in solid mechanics has garnered significant attention and presents a challenging task because it involves solving inverse problems with unknown geometries, topologies (such as the locations of the internal defects), and material properties of inclusions. Therefore, characterizing internal inclusions and defects is a classical and complex inverse problem. Zhang et al. (2022) [212] applied PINNs to defect identification. Their approach involves placing displacement sensors along the boundaries, using known loads and boundary displacement fields to infer the locations of internal defects in problems without body forces. The mathematical expression, similar to classical PINNs that include data and PDEs loss, is as follows:

$$\mathcal{L} = \beta_1 \mathcal{L}_{pdes} + \beta_2 \mathcal{L}_u + \beta_3 \mathcal{L}_t + \beta_4 \mathcal{L}_{data}, \tag{88}$$

where $\mathcal{L}_{pdes}$, $\mathcal{L}_u$, $\mathcal{L}_t$, and $\mathcal{L}_{data}$ respectively represent the loss from physical equations, displacement boundary conditions, force boundary conditions, and boundary data loss functions as follows:

$$\begin{aligned}
\mathcal{L}_{pdes} &= \frac{1}{N_{pdes}} \sum_{n=1}^{N_{pde}} |\sigma(\boldsymbol{x}^{(n)})_{ij,j}|^2, \boldsymbol{x}^{(n)} \in \Omega(\boldsymbol{\theta}_{geo}) \\
\mathcal{L}_u &= \frac{1}{N_u} \sum_{n=1}^{N_u} |u_i^{(n)}(\boldsymbol{x}^{(n)}) - \bar{u}_i^{(n)}(\boldsymbol{x}^{(n)})|^2, \boldsymbol{x}^{(n)} \in \Gamma^u(\boldsymbol{\theta}_{geo}) \\
\mathcal{L}_t &= \frac{1}{N_t} \sum_{n=1}^{N_t} |\sigma_{ij}(\boldsymbol{x}^{(n)}) n_j - \bar{t}_i(\boldsymbol{x}^{(n)})|^2, \boldsymbol{x}^{(n)} \in \Gamma^t(\boldsymbol{\theta}_{geo}) \\
\mathcal{L}_{data} &= \frac{1}{N_d} \sum_{n=1}^{N_d} |u_i(\boldsymbol{x}^{(n)}) - \bar{u}_i^*(\boldsymbol{x}^{(n)})|^2, \boldsymbol{x}^{(n)} \in \Gamma^{data}(\boldsymbol{\theta}_{geo})
\end{aligned} \tag{89}$$

where the domain $\Omega(\boldsymbol{\theta}_{geo})$ and the boundaries $\Gamma^u(\boldsymbol{\theta}_{geo})$, $\Gamma^t(\boldsymbol{\theta}_{geo})$, $\Gamma^{data}(\boldsymbol{\theta}_{geo})$ are functions of the geometric parameters $\boldsymbol{\theta}_{geo}$, which denote the internal defect locations. For instance, a circular defect could be described by the circle's center and radius, $\boldsymbol{\theta}_{geo} = \{x_c, y_c, r\}$. This approach's core integrates $\boldsymbol{\theta}_{geo}$ into the sampling coordinates, allowing optimization of both the parameters of the neural network for coordinates-to-displacement field and $\boldsymbol{\theta}_{geo}$ during backpropagation. Ultimately, extracting the $\boldsymbol{\theta}_{geo}$ parameters serves to pinpoint the defect location. Notably, Zhang et al. (2022) [212] applied PINNs not only to linear elasticity but also to hyperelasticity and elastoplasticity problems, detecting internal defects' material parameters by treating them as optimization variables. The overall framework of their method is shown in Fig. 24a. Sun et al. (2023) [213] proposed a purely data-driven algorithm for identifying multiple defects, using boundary element methods to calculate defects and displacement fields and employing convolutional neural networks to map boundary displacement fields to defects as shown in Fig. 24b.

However, current defect detection has not yet incorporated operator learning, which could potentially accelerate the computation of the forward problem under various defects, thereby enhancing the overall speed of defect detection.

#### *5.1.5. Summary*

We have found that there are generally two main approaches for applying AI for PDEs to inverse problems in solid mechanics. The first approach involves setting the variables needed for the inverse problem as optimization variables, such as identifying elastic moduli and Poisson's ratios and using PINNs for defect detection. The other approach is used when the inverse problem is more complex, such as a heterogeneous field, where a new neural network is constructed to approximate the heterogeneous field, incorporated into the loss function for optimization, such as in density function optimization for topology and constitutive equation identification (from strain to stress via neural network).

In the future, incorporating operator learning into inverse problems holds significant potential, as the core of most current inverse problems remains using PINNs to substitute finite element computations. Integrating operator learning could significantly accelerate the computation of each forward problem to enhance the computational efficiency of inverse problems substantially.

### *5.2. Fluid mechanics*

#### *5.2.1. Field reconstruction*

Raissi et al. [214] introduced the hidden fluid mechanics (HFM) that combined the knowledge from observation snapshots and the NS equations to evacuate and reconstruct the velocity, pressure fields, and even diffusions of passive scalars. A typical example of the HFM is shown in Fig. 25a. It has been demonstrated to be effective for studying the 3D hemodynamics of intracranial aneurysms without boundary conditions and inlet flow waveform, where only the concentration of a given passive scalar was known. Cai et al. [215] harnessed PINNs to infer 3D fields on an espresso cup. In that work, the tomographic background-oriented Schlieren

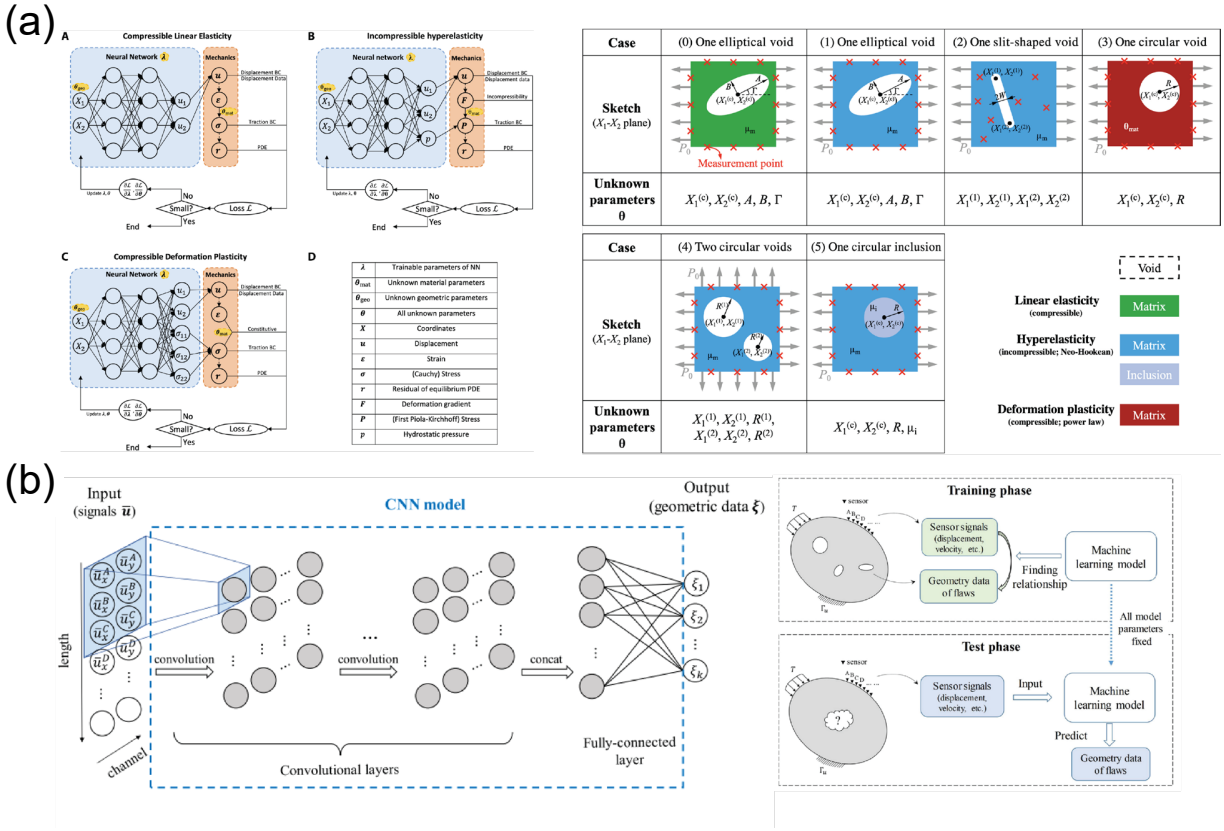


**Fig. 24.** Applications of AI for PDEs in defect identification: (a) PINNs for defect detection: Utilizing PINNs to identify internal defects by analyzing known loads and displacement fields at boundaries, applied to linear, hyperelastic, and elastoplastic issues by [212]. (b) Data-driven defect detection: An approach by [213] employing boundary element methods and convolutional neural networks to map displacement fields to detect multiple defects.

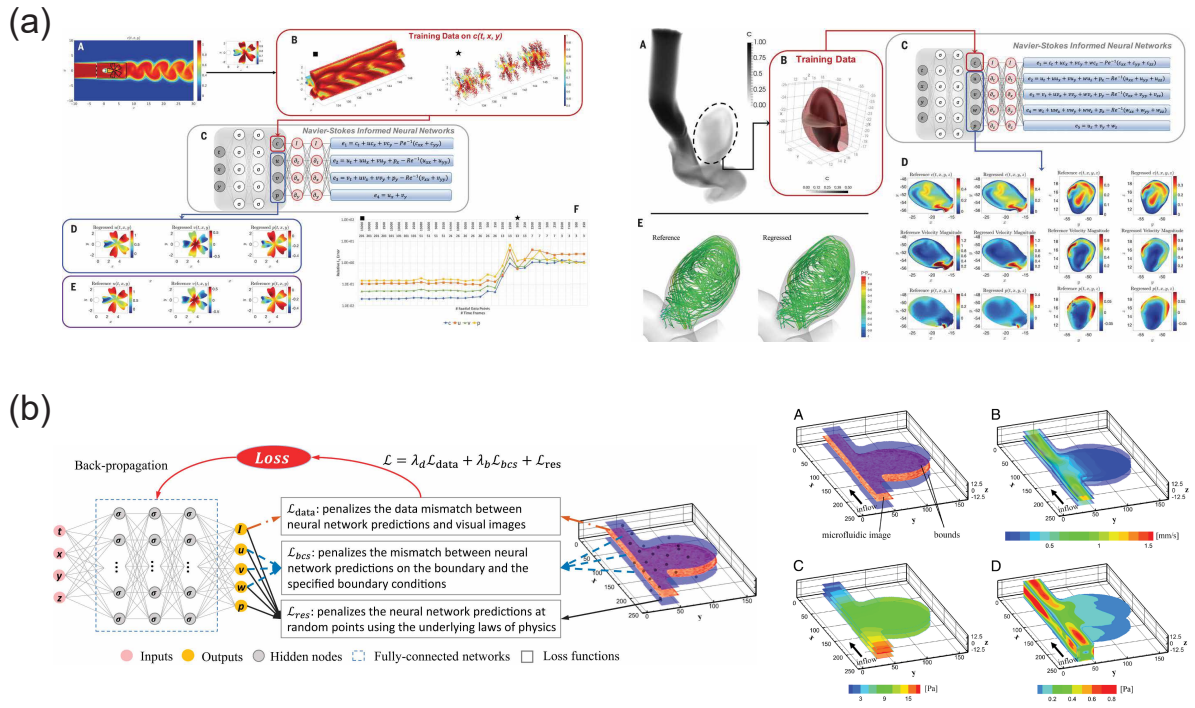


**Fig. 25.** Applications of PINNs in field reconstruction: (a) The typical example of the hidden fluid mechanics (HFM) [214]. (b) The artificial intelligence velocimetry (AIV) framework [98].

(Tomo-BOS) imaging technique measures the density and temperature fields above a cup of espresso. Then, the authors embedded the measurements as well as the NS equations into the training process, enabling PINNs to reconstruct the 3D field in terms of velocity and pressure.

Later, a framework based on PINNs named artificial intelligence velocimetry (AIV), as shown in Fig. 25b, was proposed to recover the blood flow in physiology problems [98]. Only 2D velocity data, which was measured by snapshots of the platelet-tracking technique, was applied without knowledge of the inlet and outlet conditions. The AIV successfully reconstructs various 3D fields of quantities, including the velocity, pressure, and even the shear stress at vessel boundaries, as shown in Fig. 25b. The same framework also was extended to study the murine perivascular flows [216] and non-Newtonian fluids [217]. By combining the NS equations, non-linear rheological model, and sparse observation data, the PINNs successfully reconstructed both the velocity and stress fields at not only the in-domain area but also boundaries. Molnar et al. [218] also integrated background-oriented Schlieren techniques with PINN to study the global velocity and pressure fields in supersonic flows. In their framework, the Euler equations as well as the irrotational equation are treated as additional knowledge for learning.

Turbulent flow estimation is even more challenging due to its complexity. Integrating the NS equation and adjoint-variational data assimilation, PINNs has been demonstrated to be effective for evaluating the spatial-temporal evolution of turbulent flow with low dimension [219]. However, one of the disadvantages of the PINNs for turbulence is that it is hard to assess the neural network structure. The accuracy of PINNs is highly related to networks' structure, while the relation remains to be explored [219]. Zhang et al. [220] harnessed PINNs to recover the vorticity fields of tornados with limited measurements. The predicted velocity can achieve great accuracies at different stages of tornados, including the single-cell stage, vortex breakdown stage, and multi-cell stage.

The neural operator is another way to recover the field quantities of fluid flows. Renn et al. [221] leveraged FNO to infer the vortex flow in subcritical cylinder wakes. It has been shown that the FNO was able to achieve

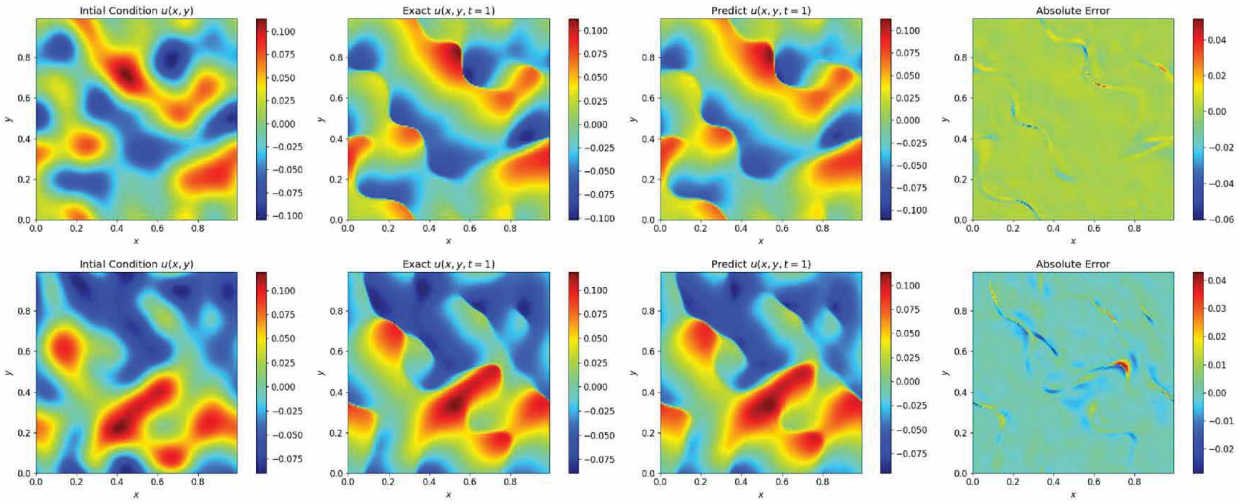


**Fig. 26.** Applications of FNO and PINO in field reconstruction for 2D inviscid Burgers equation [222].

high accuracy even at a Re number of 3060. Meanwhile, the FNO exhibited favorable computational efficiency, potentially enabling faster than real-time modeling and reconstruction. Rosofsky et al. [222] offered a wide range of inverse fluid benchmarks by using PINO, including wave equations, linear and nonlinear shallow water equations, and Burgers equations, as shown in Fig. 26. Lu et al. [223] proposed the multifidality DeepONet for the Boltzmann transportation equation. It is worth highlighting that, with the fast reconstructed global field quantities and traditional topology optimization algorithms, it is straightforward to further guide the design of structures under fluid flow conditions for multiple objectives. Li et al. [224] applied FNO to cope with 3D turbulent flow and large eddy simulation (LES). In terms of velocity spectrum, probability density functions of vorticity, and velocity increments, the results showed that the FNO performed better and more efficiently than traditional LES methods. More importantly, FNO can be extended to LES and turbulent modeling with high Re numbers.

### *5.2.2. Parameter estimation and identification in fluid*

Estimating and identifying parameters of fluids in terms of viscosity, density and other passive scalars are also of great importance for fluid mechanics. In general, the unknown parameters can be simultaneously tuned along with the field reconstruction. Jagtap et al. [72] elucidated a general form of extracting essential parameters from fluid flows. In their work, the governing equations of fluid flows were considered to consist of various differential terms and a nonlinear operator

$$\begin{aligned} \frac{\partial v}{\partial t} =& \mathcal{H}(x, t, v, v_x, v_{xx}, \cdots, v^2, v^3, \cdots, v_x v^2, \cdots) \\ \mathcal{H} =& a_0 + a_1 x + a_2 x^2 + \cdots + b_1 v + b_2 v^2 + \cdots + c_1 v_x + c_2 v_{xx} + \cdots + d_1 v_x v^2 + \cdots, \end{aligned} \tag{90}$$

where $a$, $b$, $c$ and $d$ are unknown parameters that remain to be determined.

The effectiveness of that PINNs framework was demonstrated by a 1D viscous Burgers equation and a 2D inviscid Burgers equation. Yu et al. [225] employed PINNs with a gradient-enhanced technique to infer the effective viscosity and permeability in porous media. By numerical examples, the proposed PINNs framework provided reliable viscosity as well as permeability with only 5 to 10 sets of measured data at different sensor locations. Lou. et al. [226] extended this framework by the Bhatnagar-Gross-Krook (BGK) collision model. The numerical examples have shown that it is very efficient in solving inverse problems, even if the problem is ill-posed. Kou et al. [227] utilized PINNs to estimate the turbulent viscosity and mass diffusivity for turbulent mass transfer problems. Jin et al. [160] applied PINNs to infer unknown Re numbers using velocity data and NS equations. More surprisingly, the PINNs can be also effective even with missing or noisy boundary

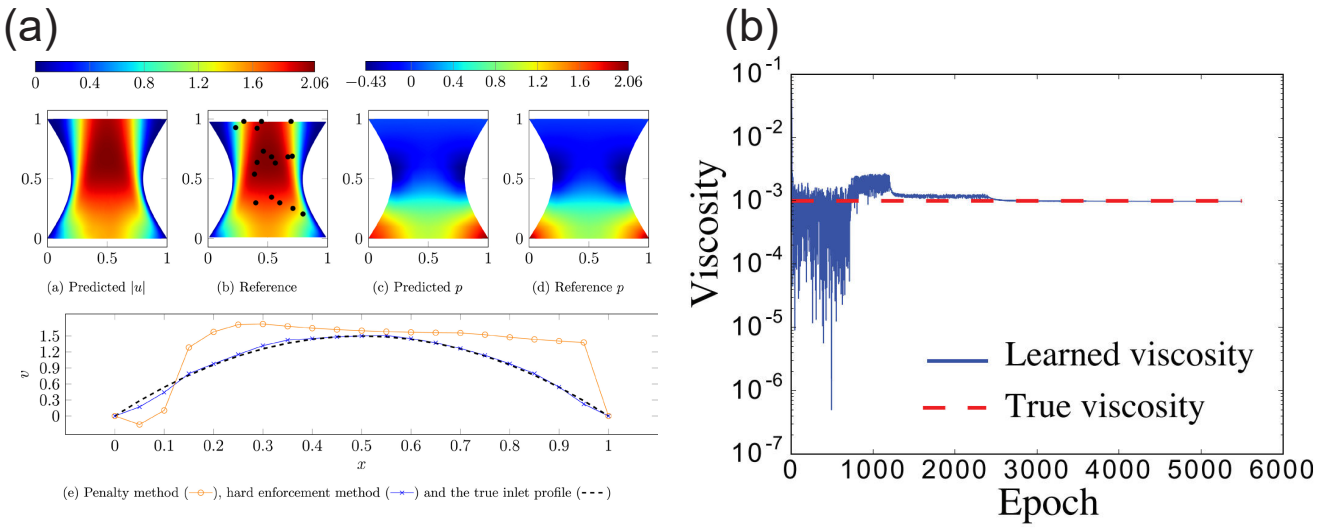


**Fig. 27.** Applications of PINNs in fluid parameter estimation and identification: (a) Inferring the boundary velocity condition of the lid-driven stenosis problem [75]. (b) Unveil the viscosity using PINNs [228].

conditions. Gao et al. [75] utilized the graph neural networks to predict the velocity boundary condition of a lid-driven stenosis, as shown in Fig. 27a. Arzani et al. [228] developed a PINNs framework that incorporates the NS equations and boundary conditions as soft constraints during the training process, enabling the accurate reconstruction of near-wall blood flow from sparse data. Moreover, the framework was able to infer the viscosity of fluids, as shown in Fig. 27b. The proposed framework has potential applications in cardiovascular disease diagnosis and treatment planning.

#### *5.2.3. Summary*

Field reconstruction and the estimation of the parameters are the major inverse problems in fluid mechanics. It has been shown that AI for PDEs is very effective not only for laminar flows, but also for turbulent flows with vorticities.

In summary, the loss functions, which are informed by physics, regulate the reconstructed fields from neural networks along with the limited observations. Thus, PINNs offers a novel and effective way to unveil the knowledge deeply hidden inside fluid flow, which is considered challenging to measure by conventional experimental equipment. As for operator learning, it focuses on learning the mapping between functions by available data. Its extraordinary generalization and discretization-invariant characteristics have made the operator learning distinguish from other CFD and deep learning-based fluid algorithms. In other words, a well-trained operator learning framework for fluid modeling can be generalized to a wide range of scenarios (with different boundary conditions, essential parameters, and even problem geometries) as well as maintaining accuracy and achieving super-resolutions. Therefore, it can be concluded that both PINNs and neural operators have appended the arsenal of computational fluid dynamics (CFD) and serve as additional tools for studying fluid phenomena inversely.

### *5.3. Biomechanics*

#### *5.3.1. Modeling Blood Flow*

Accurate modeling of blood flow is essential for understanding and predicting cardiovascular diseases such as atherosclerosis and heart failure. Traditional CFD methods require high-resolution data, which is often challenging to obtain, especially near vessel walls. AI for PDEs can bridge this gap by incorporating governing equations into the learning framework, thus enabling accurate predictions from limited data.

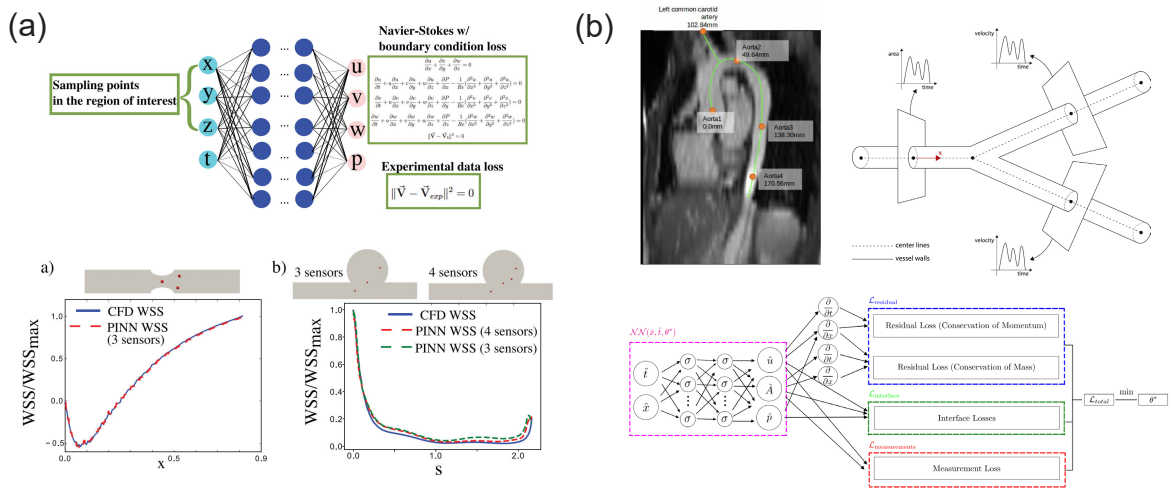


**Fig. 28.** Applications of PINNs in modeling blood flow: (a) PINNs could be used for solving Navier–Stokes equations. A few sensors (red dots) are used in the 2D stenosis model and aneurysm model, and the results predicted by PINNs are compared with the reference CFD data [228]. (b) Flow through the aorta/carotid bifurcation of a healthy human subject [229].

Near-wall blood flow is particularly crucial for understanding cardiovascular diseases and designing effective treatments. Due to the limitations of medical imaging techniques, obtaining high-resolution data near vessel walls is challenging. Arzani et al. [228] developed a PINNs framework that incorporates the Navier-Stokes equations and boundary conditions as soft constraints during the training process, enabling the accurate reconstruction of near-wall blood flow from sparse data, as shown in Fig. 28a. This PINNs framework accounts for uncertainties in the data and enables accurate reconstruction of near-wall blood flow from sparse data. Besides, the coordinate transformation technique that maps the irregular physical domain to a rectangular computational domain is applied, which can handle complex vessel geometries. This represents a significant advancement over traditional CFD methods, which require high-resolution data for accurate simulations. The proposed method has potential applications in cardiovascular disease diagnosis and treatment planning.

Accurate prediction of arterial blood pressure is vital for cardiovascular disease diagnosis and treatment planning. Traditional methods, such as invasive catheterization or simplified computational models, often lack patient specificity. Kissas et al. [229] addressed these limitations by leveraging non-invasive 4D flow Magnetic Resonance Imaging (MRI) data and physics-informed machine learning techniques, as shown in Fig. 28b.. Their PINNs framework incorporates the governing equations of fluid dynamics (Navier-Stokes equations) and the arterial Windkessel model into the neural network architecture. In addition to predicting blood pressure, PINNs has been used for the super-resolution and denoising of 4D flow MRI data, enhancing the quality and utility of MRI data in clinical applications. These advanced techniques ensure more accurate and reliable measurements, further supporting cardiovascular disease diagnosis and treatment planning [230].

### *5.3.2. Material parameter identification in soft tissue*

Elasticity imaging aims to uncover the spatial distribution of mechanical properties such as the elastic modulus and Poisson's ratio within biological tissues. This information is critical for various applications, including tumor detection and characterization. Traditional methods often rely on simplifying assumptions, such as homogeneous distributions for one material parameter, which can result in inaccuracies. Kamali et al. [231] leverage PINNs to solve linear elasticity problems and discover space-dependent distributions of both Young's modulus and Poisson's ratio using strain data, normal stress boundary conditions, and the governing physics equations, as shown in Fig. 29a. Unlike conventional methods that estimate one material parameter while assuming the other is homogeneous, the framework based on PINNs simultaneously estimates the spatial distributions of both elastic modulus and Poisson's ratio. This approach was successfully demonstrated on a simulated hydrogel

sample containing a human brain slice with distinct gray matter and white matter regions, accurately capturing the spatial distribution of mechanical properties and tissue interfaces [231]. The application of PINNs in nonhomogeneous soft tissue identification allows for the precise determination of mechanical properties based on full-field displacement measurements under quasi-static loading conditions [232]. The PINNs also provides a tool for cardiac electrophysiology characterization and parameter estimation from sparse data [233].

Brain hemodynamics, which involves quantifying blood flow velocity, vessel cross-sectional area, and pressure, is crucial for diagnosing and treating cerebrovascular diseases. Transcranial Doppler (TCD) ultrasound is a common clinical technique for non-invasively measuring blood flow velocity within cerebral arteries. However, TCD is spatially limited due to constrained accessibility through the skull's acoustic windows, providing measurements only at select locations across the cerebrovasculature. To address these limitations, Sarabian et al. [234] propose a novel physics-informed deep learning framework that integrates real-time TCD velocity measurements with baseline vessel cross-sectional areas obtained from 3D angiography images, as shown in Fig. 29b. This framework utilizes a physics-informed deep learning model to generate high-resolution maps of velocity, area, and pressure throughout the entire brain vasculature. By augmenting sparse clinical measurements with one-dimensional (1D) Reduced-Order Model (ROM) simulations, the model ensures that the predicted hemodynamic parameters are physically consistent and adhere to the governing equations of fluid dynamics. This approach enables the estimation and quantification of subject-specific cerebral hemodynamic variables with high accuracy, even without detailed knowledge of inlet and outlet boundary conditions [235].

#### *5.3.3. Protein structure prediction*

Proteins are essential biomolecules, and understanding their three-dimensional (3D) structure is crucial for elucidating their function. Experimental methods for determining protein structures, such as X-ray crystallography and nuclear magnetic resonance (NMR) spectroscopy, are time-consuming and resource-intensive. Computational methods for protein structure prediction have been an active area of research for over 50 years, aiming to predict the 3D structure of a protein from its amino acid sequence alone.

AlphaFold [7, 236], developed by DeepMind, represents a significant milestone in the application of artificial intelligence (AI) to protein structure prediction. AlphaFold incorporates physical and biological knowledge about protein structure, leveraging multiple sequence alignments, into the design of a deep learning algorithm, as shown in Fig. 30. This enables it to predict protein structures with atomic-level accuracy, even in cases where no similar structure is known. The success of AlphaFold can be attributed to three key factors:

- Incorporation of Multiple Sequence Alignments (MSAs): By using MSAs, AlphaFold captures evolutionary information that helps it predict how different parts of a protein sequence might interact in three-dimensional space.
- Integration of Physical and Biological Knowledge: AlphaFold's algorithm includes constraints based on known physical and biological principles, ensuring that the predicted structures are not only accurate but also physically plausible.
- Advanced Deep Learning Techniques: The use of sophisticated neural network architectures and training techniques allows AlphaFold to learn complex patterns from vast amounts of protein data.

The advent of AlphaFold marks a transformative development in protein structure prediction, showcasing the power of AI in tackling complex biological problems. By combining deep learning with evolutionary and physical constraints, AlphaFold sets a new standard for accuracy and opens up new avenues for research and therapeutic development. This breakthrough highlights the potential of AI to revolutionize our understanding of biomolecular structures and functions, paving the way for advancements in various fields of biology and medicine.

#### *5.3.4. Summary*

While AI for PDEs represents significant advancements in the field of biomechanics, offering the potential for more accurate and efficient modeling of complex biological systems, several challenges remain. Addressing these challenges—data acquisition, interpretability, validation, model complexity, and generalizability—will be

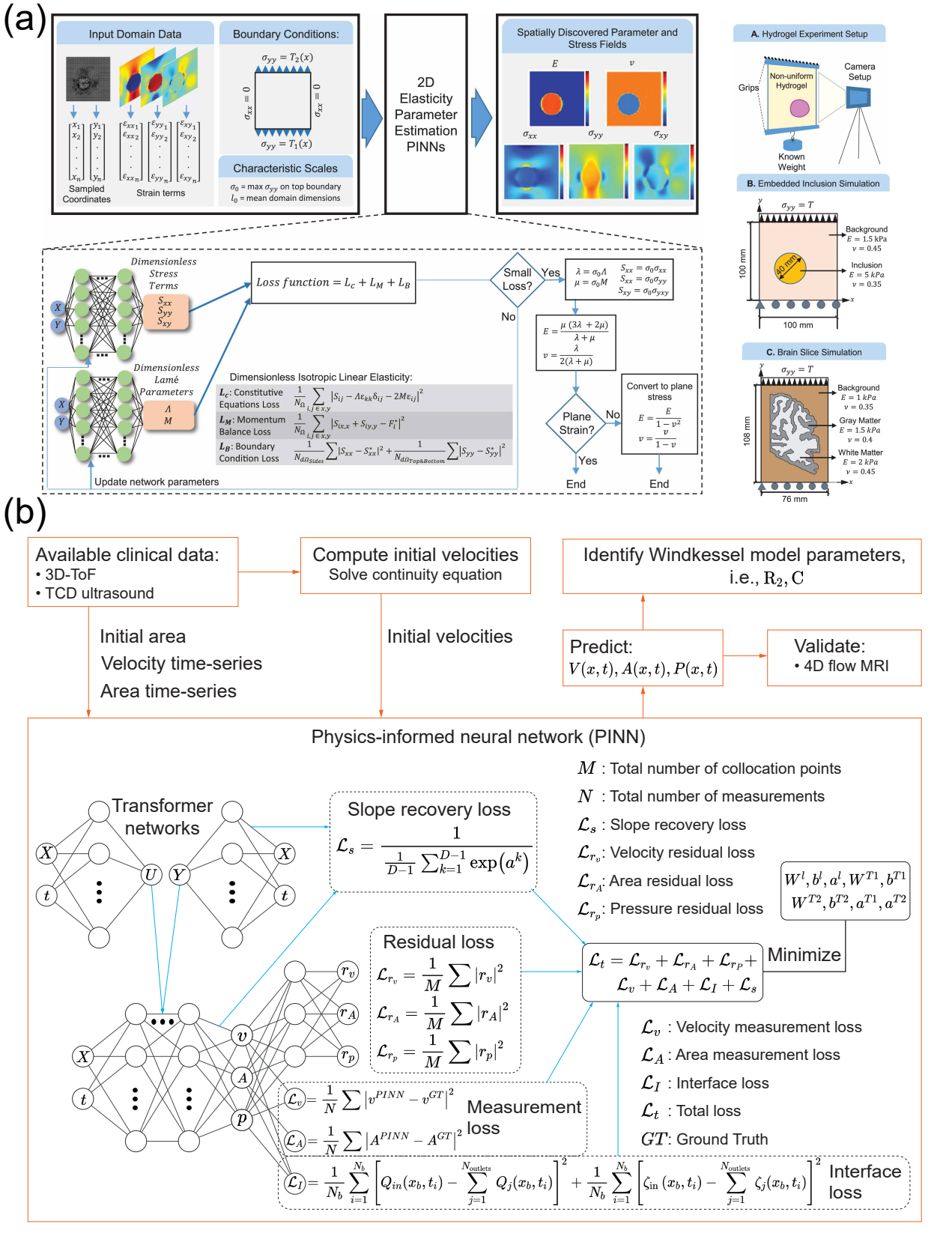


**Fig. 29.** Applications of PINNs in material parameters identification in soft issue . (a) Parameter estimation PINNs for 2D elasticity and results for the discovery of material parameter and stress distributions for the brain [231]. (b) Overview of the procedure of brain hemodynamics [234].

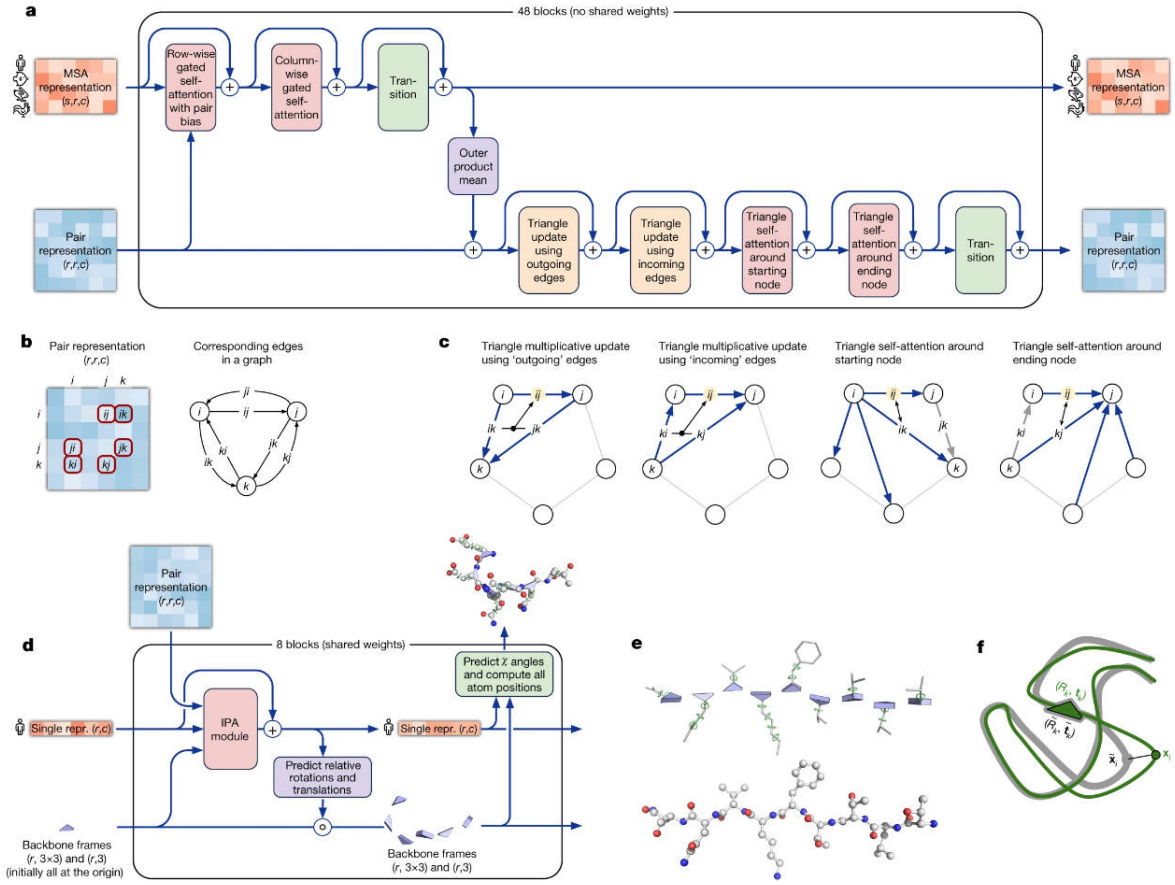


**Fig. 30.** AI for protein structure prediction: AlphaFold architecture [236].

essential for the broader adoption and successful application of AI for PDEs in biomechanics and clinical practice. Future research should focus on developing strategies to overcome these barriers, such as improved data collection methods, enhanced interpretability techniques, robust validation protocols, simplified model implementation processes, and approaches to increase model generalizability. Through these efforts, the full potential of AI for PDEs in advancing our understanding and treatment of biomechanical phenomena can be realized.

## 6. Conclusion and outlook

This paper mainly reviews the algorithms of AI for PDEs (including Physics-Informed Neural Networks, Operator Learning, and Physics-Informed Neural Operators), the related theoretical research, and applications in the forward and inverse problems of computational mechanics, including solid mechanics, fluid mechanics, and biomechanics. Based on the current state of research, possible future directions for AI for PDEs in computational mechanics might include:

(1) **Nonlinear Problems**: The core component of AI for PDEs is neural networks, which have strong nonlinear capabilities. Therefore, future research on nonlinear problems theoretically has good prospects. For linear elasticity problems in solid mechanics, finite element methods are already quite perfect due to the positive definiteness and sparsity of the stiffness matrix, which can be quickly and accurately solved by direct matrix inversion. However, for some nonlinear problems, traditional finite element methods either solve the nonlinear equation system directly using Newton's iterative method or transform and solve it explicitly using incremental steps. Neural networks, due to their inherent nonlinear capabilities, theoretically have an advantage in solving nonlinear problems in computational mechanics. Also, by training operator neural networks with existing numerical simulations or experimental results, and then using operator learning to provide a good initial solution for the initial iteration vector of the nonlinear equation system, the computational efficiency could be greatly improved theoretically. Hyperelastic problems, for example, are a good entry point because they are path-independent and can be directly formulated as a nonlinear equation system.

(2) **Complex Phenomena with Inadequate Understanding**: For such problems, due to their complexity, the mathematical PDEs descriptions of these issues can only be approximations, meaning the simulation results still differ from actual experimental results. In this case, we can rely on data to fine-tune the results. That is, an approximate solution is first provided by the boundary conditions and an approximate physical equation, and then the simulation results are fine-tuned according to the experimental data, blending a small amount of data with an approximate physical equation, especially for simulating complex phenomena. As humanity's understanding of the phenomenon becomes clearer, only the physical equations need to be corrected, and this framework remains unchanged.

(3) **Constitutive Equations**: Constitutive equations have always been a core issue in mechanics, where most of the work involves fitting. Typically, experts first construct a specific form of the constitutive model based on some basic physical principles and then fit the parameters in the model according to experimental stress-strain points. However, because of the fitting characteristics of neural networks, theoretically, they can replace constitutive equations. Most current research uses neural networks to directly fit the relationship between strain and stress. However, future studies could integrate fundamental principles of constitutive behavior, such as Noll's three theorems and thermodynamic postulates like Drucker's postulate in plasticity, into the neural network framework for constitutive equations. This would improve the convergence speed of the constitutive model and reduce the experimental data needed for stress and strain. Currently, most constitutive work in solid mechanics focuses on elasticity and hyperelasticity, but future explorations could include rate-dependent viscoelasticity and history-dependent elastoplasticity.

(4) **Foundation Models in Computational Mechanics**: Based on current research on PINNs and operator learning, it is very likely that the foundation model in computational mechanics will emerge in the future. Although current PINNs and traditional finite element methods still have significant gaps in accuracy and efficiency, PINNs may ultimately teach us how to incorporate physical equations into neural networks. Recent studies have found that operator learning can significantly speed up the simulation of forward problems, and there is theoretical evidence that operator learning can learn the families behind PDEs. Therefore, combining operator learning with algorithms for solving PDEs (algorithms for solving PDEs are not limited to PINNs,

as finite element methods are also applicable) has a very promising academic and industrial future. Not only could it significantly increase the simulation speed for forward problems (accelerating by tens of thousands of times), but it could also use experimental data to simulate complex phenomena and integrate approximate constitutive equations to form a whole new set of foundation models in computational mechanics. In the past, expert systems were developed, but their performance was often lacking. However, with the advent of neural networks, there is renewed potential to revitalize expert systems. AI for PDEs soon play a crucial role in determining which models, physical laws, discretizations, and other factors are best suited for a given problem. Moreover, AI possesses various capabilities that can enhance and improve traditional techniques, solvers, and mesh generators. The recent achievement of AI earning a silver medal in the International Mathematical Olympiad (IMO) demonstrates that AI can perform complex mathematical reasoning [237]. These developments suggest that AI for PDEs holds significant promise in computational mechanics, potentially leading to the emergence of foundation models in this field.

Scientific Artificial Intelligence (AI4Science) has garnered significant attention, similar to the work on neural networks in computer vision and language over the past decade. Its growth could mirror the rapid development seen in the Cambrian explosion of species. Initially, deep learning showed potential in these traditional scientific fields, but over time, its limitations became apparent. To overcome these shortcomings, traditional domain knowledge has been integrated to enhance AI's effectiveness in these scientific areas. AI for PDEs is a prime example of such integration.

PDEs reflect a profound and quantified expression of human understanding of the physical world. Thus, PDEs constitute a form of knowledge that can be integrated into neural network training. The core idea behind AI for PDEs is the integration of data with physical equations, aimed at scenarios with infinite data, where the physical laws derived from human understanding of nature would become unnecessary. However, in the current era where data is not infinite, integrating physical equations into data-driven approaches provides a useful transition and balancing point. This integration involves using previously calculated results for operator learning to pre-train the system, then providing a good initial solution for specific problems, and finally fine-tuning with physical equations. Most importantly, the combination of data and physical laws enables a self-learning algorithm that becomes faster with more accurate data. All past computed results are used to train operator learning, improving the accuracy of operator learning and thus reducing the iterations needed for fine-tuning by physical equations. Therefore, AI for PDEs possesses a self-updating iterative capability, promising a significant and forward-looking direction in computational mechanics.

AI for PDEs represents the future of computational mechanics. In the past, the emergence of computers brought about computational mechanics. At present, artificial intelligence is actually the previous computer, so artificial intelligence will have a new manifestation in computational mechanics, that is, AI for PDEs.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgement

The study was supported by the Key Project of the National Natural Science Foundation of China (12332005) and scholarship from Bauhaus University in Weimar. Thank Zhongkai Hao for the helpful discussion.

## Author Contributions Statement

Yizheng Wang wrote the methodology, theoretical research, and the application of AI4PDEs to solid mechanics. Jinshuai Bai wrote the application of AI4PDEs to fluid mechanics. Zhongya Lin wrote the application of AI4PDEs to biomechanics. Qimin Wang applied the permissions of pictures and drew some pictures. Cosmin Anitescu reviewed the paper. Mohammad Sadegh Eshaghi reviewed the paper. Yuantong Gu supervised the project. Xiqiao Feng supervised the project. Xiaoying Zhuang supervised the project. Timon Rabczuk reviewed the paper and supervised the project. Yinghua Liu supervised the project.

## References


[1] W. Yizheng, Z. Xiaoying, T. Rabczuk, L. Yinghua, Ai for pdes in solid mechanics: A review, Advances in Mechanics 54 (2024) 1–57.

[2] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012) 1097–1105.

[3] A. Graves, S. Fernández, F. Gomez, J. Schmidhuber, Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks (2006) 369–376.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489, oA status: bronze. doi:10.1038/nature16961.
URL https://www.nature.com/articles/nature16961.pdf

[6] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al., Grandmaster level in starcraft ii using multi-agent reinforcement learning, Nature 575 (7782) (2019) 350–354. doi:10.1038/s41586-019-1724-z.

[7] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland, et al., Improved protein structure prediction using potentials from deep learning, Nature 577 (7792) (2020) 706–710. doi:10.1038/s41586-019-1923-7.

[8] X. Zhang, L. Wang, J. Helwig, Y. Luo, C. Fu, Y. Xie, M. Liu, Y. Lin, Z. Xu, K. Yan, et al., Artificial intelligence for science in quantum, atomistic, and continuum systems, arXiv preprint arXiv:2307.08423 (2023).

[9] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707.

[10] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nature Reviews Physics 3 (6) (2021) 422–440. doi:10.1038/s42254-021-00314-5.

[11] H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, P. Chandak, S. Liu, P. Van Katwyk, A. Deac, et al., Scientific discovery in the age of artificial intelligence, Nature 620 (7972) (2023) 47–60.

[12] X. Li, Z. Liu, S. Cui, C. Luo, C. Li, Z. Zhuang, Predicting the effective mechanical property of heterogeneous materials by image based modeling and deep learning, Computer Methods in Applied Mechanics and Engineering 347 (2019) 735–753.

[13] S. Wang, Y. Teng, P. J. S. J. o. S. C. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM Journal on Scientific Computing 43 (5) (2021) A3055–A3081.

[14] R. Mattey, S. Ghosh, A novel sequential method to train physics informed neural networks for allen cahn and cahn hilliard equations, Computer Methods in Applied Mechanics and Engineering 390 (2022) 114474.

[15] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proceedings of the National Academy of Sciences 115 (34) (2018) 8505–8510.

[16] K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu, Q. Tian, Accurate medium-range global weather forecasting with 3d neural networks, Nature (2023) 1–6.

[17] E. Kharazmi, Z. Zhang, G. E. Karniadakis, hp-vpinns: Variational physics-informed neural networks with domain decomposition, Computer Methods in Applied Mechanics and Engineering 374 (2021) 113547.

[18] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, G. E. Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data, Computer Methods in Applied Mechanics and Engineering 393 (2022) 114778.

[19] H. Lee, I. S. Kang, Neural algorithm for solving differential equations, Journal of Computational Physics 91 (1) (1990) 110–131.

[20] M. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, communications in Numerical Methods in Engineering 10 (3) (1994) 195–201.

[21] I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE transactions on neural networks 9 (5) (1998) 987–1000.

[22] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics–informed neural networks: Where we are and what next, Journal of Scientific Computing 92 (3) (2022) 88.

[23] L. D. McClenny, U. M. Braga-Neto, Self-adaptive physics-informed neural networks, Journal of Computational Physics 474 (2023) 111722.

[24] Z. Wang, Z. Zhang, A mesh-free method for interface problems using the deep learning approach, Journal of Computational Physics 400 (2020) 108963. doi:10.1016/j.jcp.2019.108963.

[25] B. Yu, et al., The deep ritz method: a deep learning-based numerical algorithm for solving variational problems, Communications in Mathematics and Statistics 6 (1) (2018) 1–12.

[26] E. Haghighat, M. Raissi, A. Moure, H. Gomez, R. Juanes, A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics, Computer Methods in Applied Mechanics and Engineering 379 (2021) 113741. doi:10.1016/j.cma.2021.113741.

[27] D. W. Abueidda, Q. Lu, S. Koric, Meshless physics-informed deep learning method for three-dimensional solid mechanics, International Journal for Numerical Methods in Engineering 122 (23) (2021) 7182–7201.

[28] L. Yang, D. Zhang, G. E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, SIAM Journal on Scientific Computing 42 (1) (2020) A292–A317.

[29] D. Zhang, L. Lu, L. Guo, G. E. Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, Journal of Computational Physics 397 (2019) 108850.
[30] L. Yuan, Y.-Q. Ni, X.-Y. Deng, S. Hao, A-pinn: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations, Journal of Computational Physics 462 (2022) 111260.
[31] Y. Wang, J. Sun, W. Li, Z. Lu, Y. Liu, Cenn: Conservative energy method based on neural networks with subdomains for solving variational problems involving heterogeneous and complex geometries, Computer Methods in Applied Mechanics and Engineering 400 (2022) 115491.
[32] Y. Wang, J. Sun, T. Rabczuk, Y. Liu, Dcem: A deep complementary energy method for solid mechanics, International Journal for Numerical Methods in Engineering (2024). doi:10.1002/nme.7585.
[33] E. Samaniego, C. Anitescu, S. Goswami, V. M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications, Computer Methods in Applied Mechanics and Engineering 362 (2020) 112790.
[34] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nature Machine Intelligence 3 (3) (2021) 218–229. doi:10.1038/s42256-021-00302-5.
[35] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895 (2020).
[36] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, A. Anandkumar, Neural operator: Learning maps between function spaces with applications to pdes., J. Mach. Learn. Res. 24 (89) (2023) 1–97.
[37] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, A. Anandkumar, Physics-informed neural operator for learning partial differential equations, ACM/JMS Journal of Data Science 1 (3) (2024) 1–27.
[38] S. Chakraborty, Transfer learning based multi-fidelity physics informed deep neural network, Journal of Computational Physics 426 (2021) 109942.
[39] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Transactions on Neural Networks 6 (4) (1995) 911–917.
[40] F. Bartolucci, E. de Bézenac, B. Raonic, R. Molinaro, S. Mishra, R. Alaifari, Representation equivalent neural operators: a framework for alias-free operator learning, Advances in Neural Information Processing Systems 36 (2024).
[41] S. Goswami, M. Yin, Y. Yu, G. E. Karniadakis, A physics-informed variational deeponet for predicting crack path in quasi-brittle materials, Computer Methods in Applied Mechanics and Engineering 391 (2022) 114587.
[42] S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed deeponets, Science advances 7 (40) (2021) eabi8605.
[43] K. Hao, Ai has cracked a key mathematical puzzle for understanding our world, MIT Technology Review 15 (2020) 2021.
[44] O. C. Zienkiewicz, R. L. Taylor, J. Z. Zhu, The finite element method: its basis and fundamentals, Elsevier, 2005.
[45] T. J. Hughes, The finite element method: linear static and dynamic finite element analysis, Courier Corporation, 2012.
[46] K.-J. Bathe, Finite element procedures, Klaus-Jurgen Bathe, 2006.
[47] J. N. Reddy, Introduction to the finite element method, McGraw-Hill Education, 2019.
[48] X. Zhang, Z. Chen, Y. Liu, The material point method: a continuum-based particle method for extreme loading cases, Academic Press, 2016.
[49] G.-R. Liu, D. Karamanlidis, Mesh free methods: moving beyond the finite element method, Appl. Mech. Rev. 56 (2) (2003) B17–B18.
[50] T. Rabczuk, T. Belytschko, Cracking particles: a simplified meshfree method for arbitrary evolving cracks, International journal for numerical methods in engineering 61 (13) (2004) 2316–2343.
[51] T. Rabczuk, T. Belytschko, A three-dimensional large deformation meshfree method for arbitrary evolving cracks, Computer methods in applied mechanics and engineering 196 (29-30) (2007) 2777–2799.
[52] T. Rabczuk, J.-H. Song, X. Zhuang, C. Anitescu, Extended finite element and meshfree methods, Academic Press, 2019.
[53] V. P. Nguyen, T. Rabczuk, S. Bordas, M. Duflot, Meshless methods: a review and computer implementation aspects, Mathematics and computers in simulation 79 (3) (2008) 763–813.
[54] R. J. LeVeque, Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems, SIAM, 2007.
[55] M. Darwish, F. Moukalled, The finite volume method in computational fluid dynamics: an advanced introduction with OpenFOAM® and Matlab®, Springer, 2016.
[56] C. A. Brebbia, J. C. F. Telles, L. C. Wrobel, Boundary element techniques: theory and applications in engineering, Springer Science & Business Media, 2012.
[57] G. Karniadakis, S. J. Sherwin, Spectral/hp element methods for computational fluid dynamics, Oxford University Press, USA, 2005.
[58] S. Cai, Z. Mao, Z. Wang, M. Yin, G. E. Karniadakis, Physics-informed neural networks (pinns) for fluid mechanics: A review, Acta Mechanica Sinica 37 (12) (2021) 1727–1738.
[59] H. Jasak, A. Jemcov, Z. Tukovic, et al., Openfoam: A c++ library for complex physics simulations, in: International workshop on coupled methods in numerical dynamics, Vol. 1000, 2007, pp. 1–20.
[60] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.
[61] T. Trinh, T. Luong, Alphageometry: An olympiad-level ai system for geometry (2024).
URL https://deepmind.google/discover/blog/alphageometry-an-olympiad-level-ai-system-for-geometry/
[62] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of control, signals and systems 2 (4) (1989) 303–314.
[63] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5)

(1989) 359–366.
[64] L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, Deepxde: A deep learning library for solving differential equations, SIAM Review 63 (1) (2021) 208–228. doi:10.1137/19m1274067.
[65] J. Bai, H. Jeong, C. Batuwatta-Gamage, S. Xiao, Q. Wang, C. Rathnayaka, L. Alzubaidi, G. R. Liu, Y. Gu, An introduction to programming physics-informed neural network-based computational solid mechanics, International Journal of Computational Methods (2023).
[66] H. Xu, Y. Chen, D. Zhang, Worth of prior knowledge for enhancing deep learning, Nexus (2024).
[67] F. Lauer, G. Bloch, Incorporating prior knowledge in support vector machines for classification: A review, Neurocomputing 71 (7-9) (2008) 1578–1594.
[68] J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solving partial differential equations, Journal of computational physics 375 (2018) 1339–1364.
[69] E. Kharazmi, Z. Zhang, G. E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, arXiv preprint arXiv:1912.00873 (2019).
[70] V. Dwivedi, B. Srinivasan, Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations, Neurocomputing 391 (2020) 96–118.
[71] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1-3) (2006) 489–501.
[72] A. D. Jagtap, E. Kharazmi, G. E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Computer Methods in Applied Mechanics and Engineering 365 (2020) 113028.
[73] A. D. Jagtap, G. E. Karniadakis, 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) 2002–2041.
[74] H. Gao, L. Sun, J.-X. Wang, Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain, Journal of Computational Physics 428 (2021) 110079.
[75] H. Gao, M. J. Zahr, J.-X. Wang, Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems, Computer Methods in Applied Mechanics and Engineering 390 (2022) 114502.
[76] A. A. Ramabathiran, P. Ramachandran, Spinn: sparse, physics-based, and partially interpretable neural networks for pdes, Journal of Computational Physics 445 (2021) 110600.
[77] J. Sun, Y. Liu, Y. Wang, Z. Yao, X. Zheng, Binn: A deep learning approach for computational mechanics problems based on boundary integral equations, Computer Methods in Applied Mechanics and Engineering 410 (2023) 116012.
[78] Y. Wang, J. Sun, J. Bai, C. Anitescu, M. S. Eshaghi, X. Zhuang, T. Rabczuk, Y. Liu, A physics-informed deep learning framework for solving forward and inverse problems based on kolmogorov–arnold networks, Computer Methods in Applied Mechanics and Engineering 433 (2025) 117518.
[79] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, M. Tegmark, Kan: Kolmogorov-arnold networks, arXiv preprint arXiv:2404.19756 (2024).
[80] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, S. G. Johnson, Physics-informed neural networks with hard constraints for inverse design, SIAM Journal on Scientific Computing 43 (6) (2021) B1105–B1132.
[81] M. R. Hestenes, Multiplier and gradient methods, Journal of optimization theory and applications 4 (5) (1969) 303–320.
[82] D. He, S. Li, W. Shi, X. Gao, J. Zhang, J. Bian, L. Wang, T.-Y. Liu, Learning physics-informed neural networks without stacked back-propagation, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2023, pp. 3034–3047.
[83] Z. Fang, A high-efficient hybrid physics-informed neural networks based on convolutional neural network, IEEE Transactions on Neural Networks and Learning Systems 33 (10) (2021) 5514–5526.
[84] K. Shukla, A. D. Jagtap, G. E. Karniadakis, Parallel physics-informed neural networks via domain decomposition, Journal of Computational Physics 447 (2021) 110683. doi:10.1016/j.jcp.2021.110683.
[85] J. Bai, G.-R. Liu, A. Gupta, L. Alzubaidi, X.-Q. Feng, Y. Gu, Physics-informed radial basis network (pirbn): A local approximating neural network for solving nonlinear partial differential equations, Computer Methods in Applied Mechanics and Engineering 415 (2023) 116290.
[86] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering 384 (2021) 113938. doi:10.1016/j.cma.2021.113938.
[87] J. Bai, Y. Zhou, Y. Ma, H. Jeong, H. Zhan, C. Rathnayaka, E. Sauret, Y. Gu, A general neural particle method for hydrodynamics modeling, Computer Methods in Applied Mechanics and Engineering 393 (2022) 114740.
[88] X. Meng, Z. Li, D. Zhang, G. E. Karniadakis, Ppinn: Parareal physics-informed neural network for time-dependent pdes, Computer Methods in Applied Mechanics and Engineering 370 (2020) 113250.
[89] X. Meng, G. E. Karniadakis, A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems, Journal of Computational Physics 401 (2020) 109020.
[90] E. Haghighat, R. Juanes, Sciann: A keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks, Computer Methods in Applied Mechanics and Engineering 373 (2021) 113552.
[91] K. Zubov, Z. McCarthy, Y. Ma, F. Calisto, V. Pagliarino, S. Azeglio, L. Bottero, E. Luján, V. Sulzer, A. Bharambe, et al., Neuralpde: Automating physics-informed neural networks (pinns) with error approximations, arXiv preprint arXiv:2107.09443 (2021).
[92] O. Hennigh, S. Narasimhan, M. A. Nabian, A. Subramaniam, K. Tangsali, Z. Fang, M. Rietmann, W. Byeon, S. Choudhry, Nvidia simnet: An ai-accelerated multi-physics simulation framework, in: International conference on computational science, Springer, 2021, pp. 447–461.

[93] C. Rao, H. Sun, Y. Liu, Physics-informed deep learning for computational elastodynamics without labeled data, Journal of Engineering Mechanics 147 (8) (2021) 04021043.
[94] T. Sahin, M. von Danwitz, A. Popp, Solving forward and inverse problems of contact mechanics using physics-informed neural networks, Advanced Modeling and Simulation in Engineering Sciences 11 (1) (2024) 11.
[95] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Computer Methods in Applied Mechanics and Engineering 361 (2020) 112732.
[96] S. Wang, P. Perdikaris, Deep learning of free boundary and stefan problems, Journal of Computational Physics 428 (2021) 109914.
[97] C. Rao, H. Sun, Y. Liu, Physics-informed deep learning for incompressible laminar flows, Theoretical and Applied Mechanics Letters 10 (3) (2020) 207–212.
[98] S. Cai, H. Li, F. Zheng, F. Kong, M. Dao, G. E. Karniadakis, S. Suresh, Artificial intelligence velocimetry and microaneurysm-on-a-chip for three-dimensional analysis of blood flow in physiology and disease, Proceedings of the National Academy of Sciences 118 (13) (2021) e2100697118.
[99] X. Zhuang, H. Guo, N. Alajlan, H. Zhu, T. Rabczuk, Deep autoencoder based energy method for the bending, vibration, and buckling analysis of kirchhoff plates with transfer learning, European Journal of Mechanics-A/Solids 87 (2021) 104225.
[100] W. Li, M. Z. Bazant, J. Zhu, A physics-guided neural network framework for elastic plates: Comparison of governing equations-based and energy-based approaches, Computer Methods in Applied Mechanics and Engineering 383 (2021) 113933. doi:10.1016/j.cma.2021.113933.
[101] S. Goswami, C. Anitescu, T. Rabczuk, Adaptive fourth-order phase field analysis using deep energy minimization, Theoretical and Applied Fracture Mechanics 107 (2020) 102527.
[102] S. Goswami, C. Anitescu, S. Chakraborty, T. Rabczuk, Transfer learning enhanced physics informed neural network for phase-field modeling of fracture, Theoretical and Applied Fracture Mechanics 106 (2020) 102447.
[103] V. M. Nguyen-Thanh, X. Zhuang, T. Rabczuk, A deep energy method for finite deformation hyperelasticity, European Journal of Mechanics-A/Solids 80 (2020) 103874.
[104] J. Bai, G.-R. Liu, T. Rabczuk, Y. Wang, X.-Q. Feng, Y. Gu, A robust radial point interpolation method empowered with neural network solvers (rpim-nns) for nonlinear solid mechanics, Computer Methods in Applied Mechanics and Engineering 429 (2024) 117159.
[105] J. He, D. Abueidda, R. A. Al-Rub, S. Koric, I. Jasiuk, A deep learning energy-based method for classical elastoplasticity, International Journal of Plasticity (2023) 103531.
[106] V. M. Nguyen-Thanh, C. Anitescu, N. Alajlan, T. Rabczuk, X. Zhuang, Parametric deep energy approach for elasticity accounting for strain gradient effects, Computer Methods in Applied Mechanics and Engineering 386 (2021) 114096. doi:10.1016/j.cma.2021.114096.
[107] J. He, C. Chadha, S. Kushwaha, S. Koric, D. Abueidda, I. Jasiuk, Deep energy method in topology optimization applications, Acta Mechanica 234 (4) (2023) 1365–1379.
[108] S. Buoso, T. Joyce, S. Kozerke, Personalising left-ventricular biophysical models of the heart using parametric physics-informed neural networks, Medical Image Analysis 71 (2021) 102066.
[109] D. Dalton, D. Husmeier, H. Gao, Physics-informed graph neural network emulation of soft-tissue mechanics, Computer Methods in Applied Mechanics and Engineering 417 (2023) 116351.
[110] M. Zhu, H. Zhang, A. Jiao, G. E. Karniadakis, L. Lu, Reliable extrapolation of deep neural operators informed by physics or sparse observations, Computer Methods in Applied Mechanics and Engineering 412 (2023) 116064.
[111] J. He, S. Koric, S. Kushwaha, J. Park, D. Abueidda, I. Jasiuk, Novel deeponet architecture to predict stresses in elastoplastic structures with variable complex geometries and loads, Computer Methods in Applied Mechanics and Engineering 415 (2023) 116277.
[112] M. S. Es-haghi, M. Bamdad, C. Anitescu, Y. Wang, X. Zhuang, T. Rabczuk, Deepnetbeam: A framework for the analysis of functionally graded porous beams, Available at SSRN 4846935.
[113] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, S. M. Benson, U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow, Advances in Water Resources 163 (2022) 104180.
[114] Z. Li, D. Z. Huang, B. Liu, A. Anandkumar, Fourier neural operator with learned deformations for pdes on general geometries, Journal of Machine Learning Research 24 (388) (2023) 1–26.
[115] S. Li, W. K. Liu, Meshfree and particle methods and their applications, Appl. Mech. Rev. 55 (1) (2002) 1–34.
[116] H. Sheng, C. Yang, Pfnn: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries, Journal of Computational Physics 428 (2021) 110085.
[117] J. N. Fuhg, N. Bouklas, The mixed deep energy method for resolving concentration features in finite strain hyperelasticity, Journal of Computational Physics 451 (2022) 110839.
[118] J. He, D. Abueidda, S. Koric, I. Jasiuk, On the use of graph neural networks and shape-function-based gradient computation in the deep energy method, International Journal for Numerical Methods in Engineering 124 (4) (2023) 864–879.
[119] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, M.-H. Yang, Diffusion models: A comprehensive survey of methods and applications, ACM Computing Surveys (2022).
[120] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, arXiv preprint arXiv:2003.03485 (2020).
[121] T. De Ryck, S. Mishra, Generic bounds on the approximation error for physics-informed (and) operator learning, Advances in Neural Information Processing Systems 35 (2022) 10945–10958.
[122] N. Kovachki, S. Lanthaler, S. Mishra, On universal approximation and error bounds for fourier neural operators, The Journal of Machine Learning Research 22 (1) (2021) 13237–13312.
[123] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on

computer vision and pattern recognition, 2016, pp. 770–778.
[124] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, Y. Bengio, An empirical investigation of catastrophic forgetting in gradient-based neural networks, arXiv preprint arXiv:1312.6211 (2013).
[125] M. S. Eshaghi, C. Anitescu, M. Thombre, Y. Wang, X. Zhuang, T. Rabczuk, Variational physics-informed neural operator (vino) for solving partial differential equations, arXiv preprint arXiv:2411.06587 (2024).
[126] A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, Journal of Computational Physics 404 (2020) 109136. doi:10.1016/j.jcp.2019.109136.
[127] A. D. Jagtap, K. Kawaguchi, G. Em Karniadakis, Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 476 (2239) (2020) 20200334. doi:10.1098/rspa.2020.0334.
[128] A. Pinkus, Approximation theory of the mlp model in neural networks, Acta numerica 8 (1999) 143–195.
[129] S. Wang, X. Yu, P. Perdikaris, When and why pinns fail to train: A neural tangent kernel perspective, Journal of Computational Physics 449 (2022) 110768.
[130] D. Liu, Y. Wang, A dual-dimer method for training physics-constrained neural networks with minimax architecture, Neural Networks 136 (2021) 112–125.
[131] C. Xu, B. T. Cao, Y. Yuan, G. Meschke, Transfer learning based physics-informed neural networks for solving inverse problems in engineering structures under different loading scenarios, Computer Methods in Applied Mechanics and Engineering 405 (2023) 115852.
[132] A. Jacot, F. Gabriel, C. Hongler, Neural tangent kernel: Convergence and generalization in neural networks, Advances in neural information processing systems 31 (2018).
[133] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, A. Courville, On the spectral bias of neural networks, in: International conference on machine learning, PMLR, 2019, pp. 5301–5310.
[134] A. Kendall, Y. Gal, R. Cipolla, Multi-task learning using uncertainty to weigh losses for scene geometry and semantics, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7482–7491.
[135] I. E. Lagaris, A. C. Likas, D. G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Transactions on Neural Networks 11 (5) (2000) 1041–1049.
[136] K. S. McFall, An artificial neural network method for solving boundary value problems with arbitrary irregular boundaries, Georgia Institute of Technology, 2006.
[137] K. S. McFall, J. R. Mahan, Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions, IEEE Transactions on Neural Networks 20 (8) (2009) 1221–1233.
[138] N. Sukumar, A. Srivastava, Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks, Computer Methods in Applied Mechanics and Engineering 389 (2022) 114333.
[139] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation in machine learning: a survey, Journal of Marchine Learning Research 18 (2018) 1–43.
[140] A. K. Noor, C. Andersen, Computerized symbolic manipulation in structural mechanics—progress and potential, Computers & Structures 10 (1-2) (1979) 95–118.
[141] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural networks 4 (2) (1991) 251–257.
[142] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.
[143] N. Cohen, O. Sharir, A. Shashua, On the expressive power of deep learning: A tensor analysis, in: Conference on learning theory, PMLR, 2016, pp. 698–728.
[144] Y. Shin, J. Darbon, G. E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes, arXiv preprint arXiv:2004.01806 (2020).
[145] S. Mishra, R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for pdes, IMA Journal of Numerical Analysis 42 (2) (2022) 981–1022.
[146] A. F. Psaros, X. Meng, Z. Zou, L. Guo, G. E. Karniadakis, Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons, Journal of Computational Physics 477 (2023) 111902.
[147] S. Lanthaler, S. Mishra, G. E. Karniadakis, Error estimates for deeponets: A deep learning framework in infinite dimensions, Transactions of Mathematics and Its Applications 6 (1) (2022) tnac001.
[148] H. Guo, X. Zhuang, T. Rabczuk, A deep collocation method for the bending analysis of kirchhoff plate, arXiv preprint arXiv:2102.02617 (2021).
[149] J. Bai, T. Rabczuk, A. Gupta, L. Alzubaidi, Y. Gu, A physics-informed neural network technique based on a modified loss function for computational 2d and 3d solid mechanics, Computational Mechanics 71 (3) (2023) 543–562.
[150] Y. Wang, X. Li, Z. Yan, Y. Du, J. Bai, B. Liu, T. Rabczuk, et al., Homogenius: a foundation model of homogenization for rapid prediction of effective mechanical properties using neural operators (2024).
[151] B. Liu, Y. Wang, Y. Liu, K. Feng, T. Rabczuk, T. Olofsson, J. Bai, W. Lu, Deep learning-based multi-scale modeling of thermal conductivity in polyurethane with phase change materials via neural operators, Available at SSRN 4702962 (2024).
[152] J. C. Simo, T. J. Hughes, Computational inelasticity, Vol. 7, Springer Science & Business Media, 2006.
[153] X. Zhou, D. Lu, Y. Zhang, X. Du, T. Rabczuk, An open-source unconstrained stress updating algorithm for the modified cam-clay model, Computer Methods in Applied Mechanics and Engineering 390 (2022) 114356.
[154] D. W. Abueidda, S. Koric, R. A. Al-Rub, C. M. Parrott, K. A. James, N. A. Sobh, A deep learning energy method for hyperelasticity and viscoelasticity, European Journal of Mechanics-A/Solids 95 (2022) 104639.
[155] C. Miehe, M. Hofacker, F. Welschinger, A phase field model for rate-independent crack propagation: Robust algorithmic implementation based on operator splits, Computer Methods in Applied Mechanics and Engineering 199 (45-48) (2010) 2765–2778.
[156] C. Miehe, F. Welschinger, M. Hofacker, Thermodynamically consistent phase-field models of fracture: Variational principles

and multi-field fe implementations, International journal for numerical methods in engineering 83 (10) (2010) 1273–1311.
[157] B. Bourdin, G. A. Francfort, J.-J. Marigo, Numerical experiments in revisited brittle fracture, Journal of the Mechanics and Physics of Solids 48 (4) (2000) 797–826.
[158] G. A. Francfort, J.-J. Marigo, Revisiting brittle fracture as an energy minimization problem, Journal of the Mechanics and Physics of Solids 46 (8) (1998) 1319–1342.
[159] D. B. Mumford, J. Shah, Optimal approximations by piecewise smooth functions and associated variational problems, Communications on pure and applied mathematics (1989).
[160] X. Jin, S. Cai, H. Li, G. E. Karniadakis, Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations, Journal of Computational Physics 426 (2021) 109951.
[161] R. Zhang, P. Hu, Q. Meng, Y. Wang, R. Zhu, B. Chen, Z.-M. Ma, T.-Y. Liu, Drvn (deep random vortex network): A new physics-informed machine learning method for simulating and inferring incompressible fluid flows, Physics of Fluids 34 (10) (2022).
[162] A. J. Majda, A. L. Bertozzi, A. Ogawa, Vorticity and incompressible flow. cambridge texts in applied mathematics, Appl. Mech. Rev. 55 (4) (2002) B77–B78.
[163] E.-Z. Rui, G.-Z. Zeng, Y.-Q. Ni, Z.-W. Chen, S. Hao, Time-averaged flow field reconstruction based on a multifidelity model using physics-informed neural network (pinn) and nonlinear information fusion, International Journal of Numerical Methods for Heat & Fluid Flow 34 (1) (2024) 131–149.
[164] T.-s. Wang, G. Xi, Z.-g. Sun, Z. Huang, The prediction of external flow field and hydrodynamic force with limited data using deep neural network, Journal of Hydrodynamics 35 (3) (2023) 549–570.
[165] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.
[166] J. Han, L. Xue, Y. Jia, M. S. Mwasamwasa, F. Nanguka, C. Sangweni, H. Liu, Q. Li, Prediction of porous media fluid flow with spatial heterogeneity using criss-cross physics-informed convolutional neural networks, CMES-Computer Modeling in Engineering & Sciences 138 (2) (2024) 1323–1340.
[167] Z. Huang, X. Wang, L. Huang, C. Huang, Y. Wei, W. Liu, Ccnet: Criss-cross attention for semantic segmentation, in: Proceedings of the IEEE/CVF international conference on computer vision, 2019, pp. 603–612.
[168] C. Cheng, G.-T. Zhang, Deep learning method based on physics informed neural network with resnet block for solving fluid flow problems, Water 13 (4) (2021) 423.
[169] S. G. Rosofsky, E. Huerta, Magnetohydrodynamics with physics informed neural operators, Machine Learning: Science and Technology 4 (3) (2023) 035002.
[170] H. Li, M. Shatarah, Operator learning for urban water clarification hydrodynamics and particulate matter transport with physics-informed neural networks, Water Research 251 (2024) 121123.
[171] Z. Mao, A. D. Jagtap, G. E. Karniadakis, Physics-informed neural networks for high-speed flows, Computer Methods in Applied Mechanics and Engineering 360 (2020) 112789.
[172] J.-Z. Peng, Z.-Q. Wang, X. Rong, M. Mei, M. Wang, Y. He, W.-T. Wu, Rapid and sparse reconstruction of high-speed steady-state and transient compressible flow fields using physics-informed graph neural networks, Physics of Fluids 36 (4) (2024).
[173] X. Ren, P. Hu, H. Su, F. Zhang, H. Yu, Physics-informed neural networks for transonic flow around a cylinder with high reynolds number, Physics of Fluids 36 (3) (2024).
[174] L. Li, Y. Li, Q. Du, T. Liu, Y. Xie, Ref-nets: Physics-informed neural network for reynolds equation of gas bearing, Computer Methods in Applied Mechanics and Engineering 391 (2022) 114524.
[175] A. Joshi, A. Papados, R. Kumar, Investigation of low and high-speed fluid dynamics problems using physics-informed neural network, International Journal of Computational Fluid Dynamics 37 (2) (2023) 149–166.
[176] S. Auddy, R. Dey, N. J. Turner, S. Basu, Grinn: a physics-informed neural network for solving hydrodynamic systems in the presence of self-gravity, Machine Learning: Science and Technology 5 (2) (2024) 025014.
[177] D. Jalili, S. Jang, M. Jadidi, G. Giustini, A. Keshmiri, Y. Mahmoudi, Physics-informed neural networks for heat transfer prediction in two-phase flows, International Journal of Heat and Mass Transfer 221 (2024) 125089.
[178] C. W. Hirt, B. D. Nichols, Volume of fluid (vof) method for the dynamics of free boundaries, Journal of computational physics 39 (1) (1981) 201–225.
[179] H. Wessels, C. Weißenfels, P. Wriggers, The neural particle method–an updated lagrangian physics informed neural network for computational fluid dynamics, Computer Methods in Applied Mechanics and Engineering 368 (2020) 113127.
[180] K. Shao, Y. Wu, S. Jia, An improved neural particle method for complex free surface flow simulation using physics-informed neural networks, Mathematics 11 (8) (2023) 1805.
[181] D. Kirkpatrick, R. Seidel, On the shape of a set of points in the plane, IEEE Transactions on Information Theory 29 (4) (1983) 551–559.
[182] Y. H. Huang, Z. Xu, C. Qian, L. Liu, Solving free-surface problems for non-shallow water using boundary and initial conditions-free physics-informed neural network (bif-pinn), Journal of Computational Physics 479 (2023) 112003.
[183] W. Diab, O. Chaabi, S. Alkobaisi, A. Awotunde, M. Al Kobaisi, Learning generic solutions for multiphase transport in porous media via the flux functions operator, Advances in Water Resources 183 (2024) 104609.
[184] T.-Y. Hsieh, T.-H. Huang, A multiscale stabilized physics informed neural networks with weakly imposed boundary conditions transfer learning method for modeling advection dominated flow, Engineering with Computers (2024) 1–35.
[185] S. Jin, Z. Ma, K. Wu, Asymptotic-preserving neural networks for multiscale time-dependent linear transport equations, Journal of Scientific Computing 94 (3) (2023) 57.
[186] C. Lin, Z. Li, L. Lu, S. Cai, M. Maxey, G. E. Karniadakis, Operator learning for predicting multiscale bubble growth dynamics, The Journal of Chemical Physics 154 (10) (2021).
[187] C. Lin, M. Maxey, Z. Li, G. E. Karniadakis, A seamless multiscale operator neural network for inferring bubble dynamics,

Journal of Fluid Mechanics 929 (2021) A18.
[188] S. Cai, Z. Wang, L. Lu, T. A. Zaki, G. E. Karniadakis, Deepm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks, Journal of Computational Physics 436 (2021) 110296.
[189] Z. Mao, L. Lu, O. Marxen, T. A. Zaki, G. E. Karniadakis, Deepm&mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators, Journal of computational physics 447 (2021) 110698.
[190] K. Wu, X.-B. Yan, S. Jin, Z. Ma, Capturing the diffusive behavior of the multiscale linear transport equations by asymptotic-preserving convolutional deeponets, Computer Methods in Applied Mechanics and Engineering 418 (2024) 116531.
[191] S. E. Ahmed, P. Stinis, A multifidelity deep operator network approach to closure for multiscale systems, Computer Methods in Applied Mechanics and Engineering 414 (2023) 116161.
[192] M. Liu, L. Liang, W. Sun, A generic physics-informed neural network-based constitutive model for soft biological tissues, Computer methods in applied mechanics and engineering 372 (2020) 113402.
[193] K. Linka, N. Reiter, J. Würges, M. Schicht, L. Bräuer, C. J. Cyron, F. Paulsen, S. Budday, Unraveling the local relation between tissue composition and human brain mechanics through machine learning, Frontiers in bioengineering and biotechnology 9 (2021) 704738.
[194] R. Qiu, R. Huang, Y. Xiao, J. Wang, Z. Zhang, J. Yue, Z. Zeng, Y. Wang, Physics-informed neural networks for phase-field method in two-phase flow, Physics of Fluids 34 (5) (2022).
[195] L. J. Lim, G. H. Tison, F. N. Delling, Artificial intelligence in cardiovascular imaging, Methodist DeBakey Cardiovascular Journal 16 (2) (2020) 138.
[196] P. Moser, W. Fenz, S. Thumfart, I. Ganitzer, M. Giretzlehner, Modeling of 3d blood flows with physics-informed neural networks: comparison of network architectures, Fluids 8 (2) (2023) 46.
[197] Y. Liu, L. Cai, Y. Chen, P. Ma, Q. Zhong, Variable separated physics-informed neural networks based on adaptive weighted loss functions for blood flow model, Computers & Mathematics with Applications 153 (2024) 108–122.
[198] D. Camacho-Gomez, I. Sorzabal-Bellido, C. Ortiz-de Solorzano, J. M. Garcia-Aznar, M. J. Gomez-Benito, A hybrid physics-based and data-driven framework for cellular biological systems: Application to the morphogenesis of organoids, Iscience 26 (7) (2023).
[199] B. Lambert, A. L. MacLean, A. G. Fletcher, A. N. Combes, M. H. Little, H. M. Byrne, Bayesian inference of agent-based models: a tool for studying kidney branching morphogenesis, Journal of mathematical biology 76 (2018) 1673–1697.
[200] H. Cavanagh, A. Mosbach, G. Scalliet, R. Lind, R. G. Endres, Physics-informed deep learning characterizes morphodynamics of asian soybean rust disease, Nature communications 12 (1) (2021) 6424.
[201] C. Batuwatta-Gamage, C. Rathnayaka, H. C. P. Karunasena, W. Wijerathne, H. Jeong, Z. Welsh, M. Karim, Y. Gu, A physics-informed neural network-based surrogate framework to predict moisture concentration and shrinkage of a plant cell during drying, Journal of Food Engineering 332 (2022) 111137.
[202] C.-T. Chen, G. X. Gu, Learning hidden elasticity with deep neural networks, Proceedings of the National Academy of Sciences 118 (31) (2021) e2102721118.
[203] S. L. Brunton, J. L. Proctor, J. N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proceedings of the national academy of sciences 113 (15) (2016) 3932–3937.
[204] B. Liu, Y. Wang, T. Rabczuk, T. Olofsson, W. Lu, Multi-scale modeling in thermal conductivity of polyurethane incorporated with phase change materials using physics-informed neural networks, Renewable Energy 220 (2024) 119565.
[205] L. Li, C. Chen, Equilibrium-based convolution neural networks for constitutive modeling of hyperelastic materials, Journal of the Mechanics and Physics of Solids 164 (2022) 104931.
[206] M. Flaschel, S. Kumar, L. De Lorenzis, Unsupervised discovery of interpretable hyperelastic constitutive laws, Computer Methods in Applied Mechanics and Engineering 381 (2021) 113852.
[207] T. Kirchdoerfer, M. Ortiz, Data-driven computational mechanics, Computer Methods in Applied Mechanics and Engineering 304 (2016) 81–101.
[208] T. Kirchdoerfer, M. Ortiz, Data-driven computing in dynamics, International Journal for Numerical Methods in Engineering 113 (11) (2018) 1697–1710.
[209] J. Zehnder, Y. Li, S. Coros, B. Thomaszewski, Ntopo: Mesh-free topology optimization using implicit neural representations, Advances in Neural Information Processing Systems 34 (2021) 10368–10381.
[210] H. Jeong, C. Batuwatta-Gamage, J. Bai, Y. M. Xie, C. Rathnayaka, Y. Zhou, Y. Gu, A complete physics-informed neural network-based framework for structural topology optimization, Computer Methods in Applied Mechanics and Engineering 417 (2023) 116401.
[211] H. Jeong, J. Bai, C. P. Batuwatta-Gamage, C. Rathnayaka, Y. Zhou, Y. Gu, A physics-informed neural network-based topology optimization (pinnto) framework for structural optimization, Engineering Structures 278 (2023) 115484.
[212] E. Zhang, M. Dao, G. E. Karniadakis, S. Suresh, Analyses of internal structures and defects in materials using physics-informed neural networks, Science advances 8 (7) (2022) eabk0644.
[213] J. Sun, Y. Liu, Z. Yao, X. Zheng, A data-driven multi-flaw detection strategy based on deep learning and boundary element method, Computational Mechanics 71 (3) (2023) 517–542.
[214] M. Raissi, A. Yazdani, G. E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (6481) (2020) 1026–1030.
[215] S. Cai, Z. Wang, F. Fuest, Y. J. Jeon, C. Gray, G. E. Karniadakis, Flow over an espresso cup: inferring 3-d velocity and pressure fields from tomographic background oriented schlieren via physics-informed neural networks, Journal of Fluid Mechanics 915 (2021) A102.
[216] K. A. Boster, S. Cai, A. Ladrón-de Guevara, J. Sun, X. Zheng, T. Du, J. H. Thomas, M. Nedergaard, G. E. Karniadakis, D. H. Kelley, Artificial intelligence velocimetry reveals in vivo flow rates, pressure gradients, and shear stresses in murine

perivascular flows, Proceedings of the National Academy of Sciences 120 (14) (2023) e2217744120.
[217] M. Mahmoudabadbozchelou, G. E. Karniadakis, S. Jamali, nn-pinns: Non-newtonian physics-informed neural networks for complex fluid modeling, Soft Matter 18 (1) (2022) 172–185.
[218] M. Schierholz, Y. Hassab, C. Schuster, Engineering-informed design space reduction for pcb based power delivery networks, IEEE Transactions on Components, Packaging and Manufacturing Technology (2023).
[219] Y. Du, M. Wang, T. A. Zaki, State estimation in minimal turbulent channel flow: A comparative study of 4dvar and pinn, International Journal of Heat and Fluid Flow 99 (2023) 109073.
[220] H. Zhang, H. Wang, Z. Xu, Z. Liu, B. C. Khoo, A physics-informed neural network-based approach to reconstruct the tornado vortices from limited observed data, Journal of Wind Engineering and Industrial Aerodynamics 241 (2023) 105534.
[221] P. I. Renn, C. Wang, S. Lale, Z. Li, A. Anandkumar, M. Gharib, Forecasting subcritical cylinder wakes with fourier neural operators, arXiv preprint arXiv:2301.08290 (2023).
[222] S. G. Rosofsky, H. Al Majed, E. Huerta, Applications of physics informed neural operators, Machine Learning: Science and Technology 4 (2) (2023) 025022.
[223] L. Lu, R. Pestourie, S. G. Johnson, G. Romano, Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport, Physical Review Research 4 (2) (2022) 023210.
[224] Z. Li, W. Peng, Z. Yuan, J. Wang, Fourier neural operator approach to large eddy simulation of three-dimensional turbulence, Theoretical and Applied Mechanics Letters 12 (6) (2022) 100389.
[225] J. Yu, L. Lu, X. Meng, G. E. Karniadakis, Gradient-enhanced physics-informed neural networks for forward and inverse pde problems, Computer Methods in Applied Mechanics and Engineering 393 (2022) 114823.
[226] Q. Lou, X. Meng, G. E. Karniadakis, Physics-informed neural networks for solving forward and inverse flow problems via the boltzmann-bgk formulation, Journal of Computational Physics 447 (2021) 110676.
[227] C. Kou, Y. Yin, Y. Zeng, S. Jia, Y. Luo, X. Yuan, Physics-informed neural network integrate with unclosed mechanism model for turbulent mass transfer, Chemical Engineering Science 288 (2024) 119752.
[228] A. Arzani, J.-X. Wang, R. M. D'Souza, Uncovering near-wall blood flow from sparse data with physics-informed neural networks, Physics of Fluids 33 (7) (2021).
[229] G. Kissas, Y. Yang, E. Hwuang, W. R. Witschey, J. A. Detre, P. Perdikaris, Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering 358 (2020) 112623.
[230] M. F. Fathi, I. Perez-Raya, A. Baghaie, P. Berg, G. Janiga, A. Arzani, R. M. D'Souza, Super-resolution and denoising of 4d-flow mri using physics-informed deep neural nets, Computer Methods and Programs in Biomedicine 197 (2020) 105729.
[231] A. Kamali, M. Sarabian, K. Laksari, Elasticity imaging using physics-informed neural networks: Spatial discovery of elastic modulus and poisson's ratio, Acta biomaterialia 155 (2023) 400–409.
[232] E. Zhang, M. Yin, G. E. Karniadakis, Physics-informed neural networks for nonhomogeneous material identification in elasticity imaging, arXiv preprint arXiv:2009.04525 (2020).
[233] C. Herrero Martin, A. Oved, R. A. Chowdhury, E. Ullmann, N. S. Peters, A. A. Bharath, M. Varela, Ep-pinns: Cardiac electrophysiology characterisation using physics-informed neural networks, Frontiers in Cardiovascular Medicine 8 (2022) 768419.
[234] M. Sarabian, H. Babaee, K. Laksari, Physics-informed neural networks for brain hemodynamic predictions using medical imaging, IEEE transactions on medical imaging 41 (9) (2022) 2285–2303.
[235] C. Wu, Y. Xu, J. Fang, Q. Li, Machine learning in biomaterials, biomechanics/mechanobiology, and biofabrication: State of the art and perspective, Archives of Computational Methods in Engineering (2024) 1–67.
[236] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al., Highly accurate protein structure prediction with alphafold, nature 596 (7873) (2021) 583–589.
[237] AlphaProof, A. teams, Ai achieves silver-medal standard solving international mathematical olympiad problems (2024).
URL https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/