OPEN ACCESS
IOP Publishing

Journal of Physics: Condensed Matter

J. Phys.: Condens. Matter **37** (2025) 403003 (32pp)

https://doi.org/10.1088/1361-648X/ae096d

Topical Review

Deep learning for simulating the evolution of condensed matter systems at the continuum scale: methods and applications

Daniele Lanzoni[1,2], **Francesco Montalenti**[1] **and Roberto Bergamaschini**[1,*]

[1] Department of Materials Science, University of Milano-Bicocca, 20125 Milano, Italy
[2] Department of Physics, University of Genova, 16146 Genova, Italy

E-mail: roberto.bergamaschini@unimib.it

Received 2 March 2025, revised 4 July 2025
Accepted for publication 18 September 2025
Published 6 October 2025

Abstract

Studying the time-evolution of complex systems is key in any scientific field and a cornerstone for the understanding of condensed matter physics. To this end, continuum models have been developed since the early times of science. Despite the huge advancements in computational methods, they can still be challenging, especially when demanding high spatial and time resolutions over large scales. In recent years, neural networks have emerged as a possible alternative to speed-up or even replace traditional numerical schemes, promising fast, yet equally accurate solutions. This Review aims to recognize the state-of-the-art of these novel approaches. The literature is inspected with a dual purpose. First, the several strategies and architectures exploited so far to tackle time-dependent evolutions via neural networks are systematized. Second, the different applications and successful uses are showcased. A general distinction is drawn between data-driven approaches, relying on the availability of large datasets of solutions, and physics-informed strategies, exploiting neural networks to solve known sets of partial differential equations. Recent approaches mixing these two methods, as well as novel concepts, are also discussed. The analysis concludes with a general evaluation of the current trends and perspective developments, contrasted with the main challenges and drawbacks still limiting the use of neural network-based approaches as effective surrogates of conventional computational methods.

Keywords: neural networks, surrogate models, continuum models, time evolution

[*] Author to whom any correspondence should be addressed.

1. Introduction

Artificial intelligence and machine learning (ML) are nowadays penetrating everyday life in almost every area, from science and technology to economy and social dynamics and are expected to have a revolutionary impact in the coming years [1–5]. The strategic importance of such instruments has become evident to governments, big-tech leaders and research institutions, feeding ambitious development plans.

A great part of this success comes from the maturity reached by neural networks (NNs) [6–8], theorized since decades [9–12], but massively utilized only in the last few years thanks to the recent advancement of ad-hoc hardware technologies, most prominently graphical processing units (GPU), which perfectly conform with the tensor nature of NN computation. Nowadays, NNs are successfully exploited in main-stream applications like image and language recognition, vision tasks, video, music and large language models (LLM).

At the same time, applications of NNs for scientific research have flourished in almost any field [13]. Some inspirational examples can be found e.g. in fluid dynamics [14–16], mechanics [17], photonics [18, 19] and electrodynamics [20], and more specifically in materials science and condensed matter physics [21, 22]. The development and exploitation of NN interatomic potentials [23–28] is nowadays routine for molecular dynamics simulations. NNs proved successful also for novel materials discovery [29–36] as well as for the study of structure-properties relationships and inverse design [37–40].

The recent success of NNs in treating images and videos has triggered the research toward using NNs for modeling time-dependent problems, given that their solutions can be interpreted as sequences of image stacks. This holds in general for any partial differential equation (PDE) model tracing the dynamics of one or more fields on a prescribed domain. Among them, a major interest has been directed toward phase-field (PF) [41, 42] approaches, as providing an ideal framework to tackle complex geometries and multi-dimensional/multi-physics problems but resulting somewhat limited by high-computational costs. NN approaches, with their flexible approximation capabilities and fast evaluation times, promise to be able to accelerate or even surrogate simulations using conventional direct numerical solvers (DNSs) [43], allowing for high-throughput studies in fields in which these have been historically difficult. Moreover, NNs could be ideally devised to work directly on experimental data completely bypassing the classical DNS modeling in favor of a fully data-driven approach. This fascinating perspective, however, is generally unfeasible due to the lack of suitably large and complete datasets thus making the derivation of DNS-based NN models a smarter and effective choice.

This Review offers a thorough overview of the research on NN schemes for solving time-dependent problems at the continuum scale, applied to the fields of condensed matter physics, materials science and engineering. Even if several approaches have found successful application to fluid dynamics problems, as already reported in some reviews [44–46], we here enlarge the view towards solid-state systems, even though the treatment and implications will be conducted from a general perspective and hence holds irrespective of the application field. Since most of the literature comes from the last five years and novel approaches continuously appear every month, our effort is meant to provide an up-to-date mapping of this bleeding-edge research area to be used both as an 'instant picture' of the current state of the art and as an entry point for the newcomers. On the one hand, we showcase the several applications reported in the literature to evidence the possible uses, advantages and pitfalls of NN approaches with respect to DNS; on the other, we provide a walkthrough of the key technical concepts and architectures that have been so far exploited for such numerical task. In doing so, we still aim at a treatment that remains accessible even to readers who are not experts in the field, to bridge the gap that still persists between the specific technical vocabulary of developers and the vague intuition of potential users, often perceiving NNs as black-boxes. To ease the reading, the main acronyms used in this work and commonly adopted in the literature are listed in table 1.

Table 1. List of acronyms.

ML	Machine learning
NN	Neural network
GPU	Graphical processing unit
PDE	Partial differential equation
DNS	Direct numerical solver
FEM	Finite element method
CFD	Computational fluid dynamics
PF	Phase field
PINN	Physics-informed neural networks
UA	Universal approximation
ReLU	Rectified linear unit
SGD	Stochastic gradient descend
ADAM	Adaptive moment (training algorithm)
MSE	Mean squared error
MAE	Mean absolute error
RMSE	Root mean squared error
MARE	Mean absolute relative error
MRE	Mean relative error
ResNet	Residual network
CNN	Convolutional neural network
RNN	Recurrent neural networks
LSTM	Long-short term memory
GRU	Gated recurrent unit
CRNN	Convolutional recurrent neural network
LLM	Large language model
PC	Principal component
PCA	Principal component analysis
AE	Autoencoder
VAE	Variational auto encoder
CAE	Contractive autoencoder
DeepONet	Deep operator network

First, in section 2, we focus on the methodology. After a brief overview of key general concepts, we proceed with a thorough analysis of the different NN architectures which are nowadays exploited for the modeling of time-dependent continuum problems, starting from the simplest feed-forward fully-connected structures to the more advanced ones built by leveraging their inherent modularity. General indications about NN training, parameter optimization and performance evaluation are also provided at the end of the section.

Then, we proceed with the actual review of the literature studies and distinguish between the two main conceptual approaches that have been proposed so far.

In section 3 we discuss data-driven models where NNs are trained to reproduce the solution of a PDE problem starting from a large dataset of cases obtained by a DNS. This strategy has been exploited in two different ways. On the one hand, it is possible to exploit a NN to support the actual DNS, e.g. to produce fast prediction of bottlenecking terms in coupled problems or to provide correction/refinement of approximated solutions. On the other hand, the NN can be used as a full surrogate of the DNS to obtain predictions of the entire PDE solution over time. This approach has proven very successful in many literature studies, even if it relies on the construction of possibly large datasets by DNS, which may still pose serious computational efforts.

In section 4, we instead discuss the complementary NN approaches which may be trained exploiting little to no data. In the present Review, we will mainly focus on the concept of so-called physics-informed NNs (PINNs), as they represent the most widespread framework. At variance with pure data-driven models, in PINNs a NN is used directly as an ansatz for the solution of the PDE at hand. The training is no longer based on examples obtained from DNS (optionally supplemented by a dataset), but the NN learns based on the difference between its predictions and the expected behavior of the problem solution. This makes PINNs particularly interesting in contexts where data collection is expensive or even impossible. Furthermore, this workflow is mesh-free, hence once the NN is trained it is possible to evaluate the solution at any point in space and in time.

Still, the reader should remember that the separation between data- and physics-driven approaches is mainly conceptual and the two have been combined at different levels. Actually, the addition of physics-based information inside data-driven approaches and vice versa is one of the main trends in the current research landscape. A discussion of such mixed approaches is therefore also included.

In section 5, additional strategies which do not fall in the previous classification but are expected to gain a key role in the future development of the field are reviewed. In particular, these comprise attention and transformers, reinforcement learning approaches and ensemble methods. A brief discussion on the use of PINNs to discover governing equations and how NN workflows can be conveniently integrated into numerical solution techniques for PDE problems is also provided.

Finally, in section 6 we critically discuss the current development stage of the reported NN approaches and assess some of the open challenges and questions to be addressed to expand the range of applications toward more complex and intriguing problems. Possible extensions and future trends are also envisaged.

2. NN methods for time-dependent problems

In this section, we provide a quick introduction to the fundamentals of NNs, focusing on aspects which are most important to tackle time-dependent continuum problems. The scope is to provide the unfamiliar reader with a lightweight introduction to the topic, which at the same time covers the major approaches and terminology in present research. Additional details and more technical discussions may be found in the excellent books at [6, 47, 48], the Reviews at [8, 49] and references therein.

Topics are presented following a deep learning conceptual workflow. First, foundational aspects in supervised deep learning are discussed. Next, NN architectures are presented: after the description of prototypical structures which serve as building blocks, the section develops by progressively increasing the complexity of models, discussing more advanced composite networks. Recurrent NN (RNN) are then presented in details, due to their important role in time-dependent problems modeling and their peculiar input-to-output topology. Following the presentation of these most common NNs, an overview of the training procedure is provided, with a particular emphasis on strategies for preventing overfitting. Finally, evaluation metrics for trained models are discussed distinguishing between local and global approaches.

2.1. *Generalities on deep learning*

It is useful to introduce NNs from a ‘supervised learning’ problem perspective. By this term, we refer to the set of approaches which aim at finding a good approximation of an (unknown) function given a set of input-output, or features-labels—(x, y), examples. As will be discussed in section 4.1, this conceptual framework may be generalized to the related context of approximating the solution of a known PDE even if examples are few or missing. This is in contrast with so-called ‘unsupervised’ learning tasks, in which one tries to infer probability distributions and properties from unlabeled data. In both cases, NNs are one of the most successful tools, although alternatives exist [47].

In supervised learning, the goal is to find an approximation of the $x \rightarrow y$ mapping in a context in which this is unknown or expensive, hence a reasonably accurate and possibly fast approximation is desired. The procedure of obtaining a ‘good’ map is called ‘training’. Since NNs are nonlinear, parametric function approximators, this is done by adjusting internal parameters $\vec{\vartheta}$ so that the predicted outputs for a given set of features is as close as possible to the ones present in the dataset. Such discrepancy is measured in terms of a loss function $\mathcal{L}(\vec{\vartheta})$, to be minimized during training. This yields strong similarities with optimization tasks and is often efficiently performed based on loss gradient information (more details are provided in section 2.5).

The reasons for NNs success can be traced back to three principles:

- *Universal approximation* (UA). NNs are capable of approximating functions to an arbitrary degree of precision. Several proofs of UA theorems have been formulated, see [50–52] for early statements and [53, 54] for comprehensive reviews. Strictly speaking, UA is achieved only in the limiting case of an infinite number of NN parameters. Hence, in real-world applications, the tradeoff between the achievable accuracy and the model size is to be assessed. Unfortunately, this task is mostly problem-dependent so that a considerable part of NN setup is devoted to it (see also section 2.5.2).
- *Manifold hypothesis*. While multi-variate data are represented as very high-dimensional vectors, real samples are expected to live on a smaller, nonlinear manifold [6]. If such lower-dimensional space is properly identified and analyzed, the entire variability of a dataset could be expressed by a limited set of descriptors, much more tractable in computations. NNs consist of several, subsequent nonlinear transformations, that can be interpreted as a series of hierarchical projections selecting the right manifold [55]. This allows one to effectively train models without falling into the 'curse of dimensionality', i.e. the exponential increase of dataset size, hindering the exploitation of other classical approaches, such as generalized linear models and basis function expansion [47], in high dimensional settings.
- *Architectural modularity*. Virtually the output of any NN can be combined with, or fed to, other networks. This easily enables the construction of derived, higher-level architectures capable of handling more specialized tasks. Such flexibility is crucial for processing various data types, such as images, time series and graphs. The motivation is not only computational efficiency, as there is a strong connection between the structure of a NN and the input-output relationship, making it possible to encode 'inductive biases' [6, 47]. This is critical in physics-related problems as it is often the place where prior knowledge enters into the model training procedure.

2.2. Feed-forward NNs

In this section we dive a bit deeper into the basic NN architectures relying on a feed-forward structure, i.e. a mapping of inputs to outputs through the composition of one-directional consecutive processing layers.

2.2.1. Fully connected NNs.

The archetypal structure of a feed-forward NN is the fully connected, multilayer perceptron [9], capable of mapping a vector-valued $\vec{x}$ input to a vector-valued $\vec{y}$ output and schematized by the computational graph in figure 1. It is obtained by stacking multiple affine transformations (multiplication by a matrix and addition of a constant vector called 'bias' b) and nonlinear transformations, or 'activation functions' f. Components of the input, output and of intermediate calculations vectors (the so-called 'hidden layers' $\vec{h}^{(i)}$) are nodes; arrows starting from the ith node at one layer and ending at the jth one in the following layer represent multiplication with the ij element of a matrix $\overline{W^{(i)}}$. These entries are commonly referred to as 'weights'. When multiple arrows enter the same node, summation is implied. Additionally, at hidden nodes, the nonlinear activation function is applied, as shown in figure 1(a). The explicit presence of the bias term is often absorbed into the matrix multiplication, as it is sufficient to add a dummy, constant node equal to 1 to each layer [6]. The number of affine-nonlinear layers, is commonly referred to as the 'depth' of the NN. The term 'deep learning' comes from the use of NNs with tens or even hundreds of stacked layers [56, 57]. The dimensionality of the hidden layers, instead, is usually called 'width'. This quantity generally differs from the dimensionality of the input layer and can also vary across hidden layers, giving an additional degree of freedom in the definition of the NN architecture.

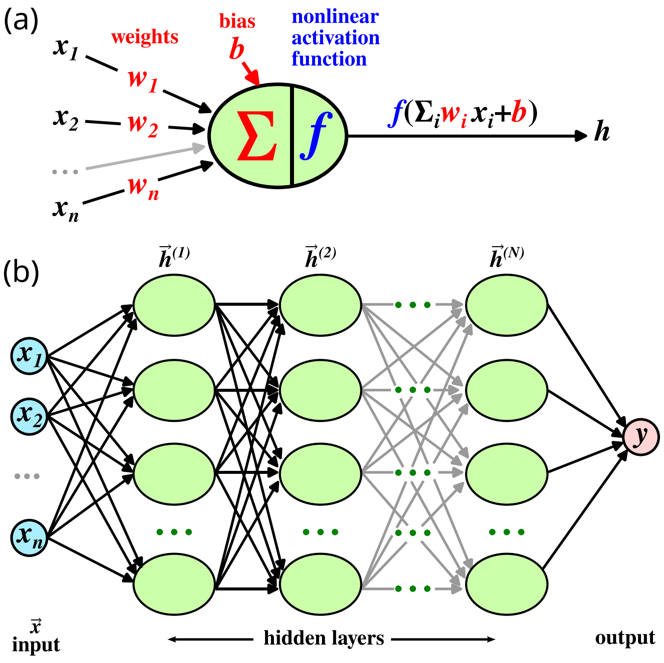

Figure 1. Schematics of the structure of (a) a single node and (b) a multilayer NN. In this case, the NN has a vector-valued input $\vec{x}$ (with components x_i), a scalar output y and N hidden layers.

NNs have been used as models for the human brain [9]. This explains why many deep learning terms have a biological or neuroscientific flavor. Following the neuroscience analogy, nodes represent 'neurons' and weights represent the strength of synaptic connections. We will not, however, discuss these aspects here and refer the interested reader to the relevant literature [58–60].

From a more mathematical perspective, multilayer perceptions (and NNs in general) are obtained by composing multiple functions, i.e. for an architecture with N hidden layers:

$$\vec{h}^{(i)} = f\left(\overline{W}^{(i)} \cdot \vec{h}^{(i-1)}\right) \qquad i \in [1, N] \tag{1}$$

$$\vec{y} = \overline{W}^{(N+1)} \cdot \vec{h}^{(N)} \tag{2}$$

where $\overline{W}^{(i)}$ are affine transformations, f is the element-wise nonlinear 'activation function' and $\vec{h}^{(i)}$ is the ith hidden layer with the convention that $\vec{h}^{(0)} = \vec{x}$. An additional activation may be used in the last layer (equation (2)) if $\vec{y}$ is to be bounded to a specific range. An example is classification tasks, in which the NN output is to be interpreted as a probability. The consecutive application of these transformations is referred to as the 'forward pass'.

In the initial years of NN development, the most commonly used were sigmoidal functions such as the hyperbolic tangent and the logistic sigmoid σ:

$$\sigma(\vec{x}) = \frac{1}{1 + \exp(-\vec{x})} \tag{3}$$

where, with abuse of notation, operations on vectors are to be intended component-by-component. More recently, the Rectified Linear Unit (ReLU) function became a more typical choice:

$$\mathrm{ReLU}(\vec{x}) = \max(\vec{x}, 0). \tag{4}$$

However, a huge number of alternative activation functions, each with different use cases, strengths and weaknesses, have been proposed, as reported in the comprehensive review in [61].

The fully-connected multilayer perceptron (figure 1) is well suited for vector-to-vector maps, but it may not be the most efficient architecture for any task, despite its UA capabilities, thus making it convenient to specialize its basic structure according to the problem at hand.

2.2.2. Convolutional NNs (CNNs).

CNN [62, 63] are one of the most common and popular NN architectures, devised to manipulate data on structured, regular grids such as images. CNNs may be obtained from the multilayer perceptron described in figure 1 by substituting the generic affine transformations $\overline{W}^{(i)}$ with convolution operations. The hidden state $\vec{h}^{(i)}$ in a convolutional layer is then obtained as:

$$\vec{h}^{(i)} = f\left(\overline{K}^{(i)} * \vec{h}^{(i-1)} + b^{(i)}\right) \tag{5}$$

where we have explicitly reported the bias term $b^{(i)}$ since $\overline{K}^{(i)}$ now represent convolution kernels (i.e. linear transformations) and f is the activation function. Convolution (technically cross-correlation) operates as in figure 2: the kernel (usually a square matrix, but rectangular ones are also possible) is centered on a pixel of the input and corresponding values are multiplied. The results are then summed together and with the bias value $b^{(i)}$ and placed in the corresponding output pixel. The convolution kernel is then shifted to the next pixel and

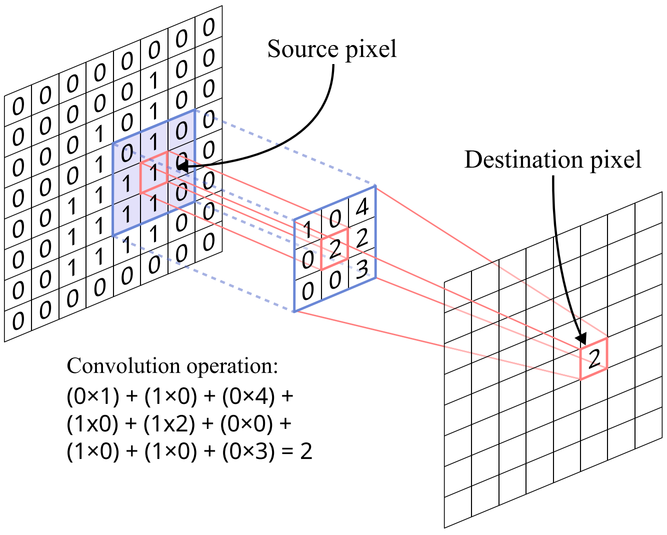

Figure 2. Schematics of a convolutional operation on a 10 × 10 binary image, using a 3 × 3 kernel. The source pixel and the destination one are contoured in red, while pixels in the receptive field are highlighted in blue.

the procedure is iterated over the whole image. Larger shifts, named 'stride', may also be used, a possibility exploited to reduce the width and height of the input image [6, 48][3]. This is particularly important for tasks involving dimensionality reduction (see e.g. the AE, section 2.3.1) or in image classification, where a whole image should be assigned a single scalar value, representing the probability that the input data corresponds to a specific class.

Boundary regions have to be treated with care due to missing neighbors. If $\overline{K}^{(i)}$ is centered on a boundary pixel, it is possible to fill missing values ('padding') with virtual ones. Depending on the strategy, this may be used to implement periodic boundary conditions ('circular' padding), Dirichlet boundaries ('zero' or 'constant' padding) or zero-flux boundaries ('reflection' padding).

The convolution operation, so far referred to the case of a (2D) image, can suitably be generalized to arrays (1D), volumetric data (3D) or their higher-dimensional counterparts. Between hidden layers, the number of scalar values associated with each pixel, called 'channels' as in RGB images, may be modified, similarly to how in the simple multilayer perceptron it is possible to modify the width of layers.

A crucial feature of convolution kernels is that they have a finite support, hence the value in a given pixel in $\vec{h}^{(i)}$ is only influenced by values in nearby pixels in $\vec{h}^{(i-1)}$. In other words, CNNs encode the 'locality' property in the layer-to-layer and, as a consequence, $x \to y$ mapping. The range over which pixel values may communicate through the convolution operation is sometimes called 'receptive field', in analogy with biological systems. A CNN's local behavior is markedly different with

[3] Gifs showing different convolution modalities are available at https://hannibunny.github.io/mlbook/neuralnetworks/convolutionDemos.html.

respect to the vanilla multilayer perceptron described before, in which all the components of $\vec{h}^{(i)}$ depend, at least in principle, on all the components of $\vec{h}^{(i-1)}$. For this reason, such architectures are called 'fully-connected', in contrast with the local CNNs. In other words, CNN may be considered as sparse fully-connected networks in which weights connecting pixels which are outside of the kernel receptive fields are set to zero exactly, i.e. they are (infinitely strongly) regularized networks. Additionally, the use of convolution not only may have theoretical justifications (for spatial data local information is more important than non-local one) but also has the important advantage of reducing the effective number of parameters, making training less expensive in terms of computational resources and feasible with far less examples.

Locality is not the only property encoded by the convolutional structure. By construction, a NN which only uses convolutions as affine transformations is also translationally equivariant, i.e. its application commutes with spatial translations, represented by pixel shifts. This property is critical, as many physical systems exhibit this kind of symmetry. CNNs are therefore one of the most used architectures when dealing with spatial data which may be discretized on regular grids, such as evolutions coming from PDEs or, in principle, experimental imaging tools.

An alternative approach to strided convolutions to reduce the size of an image in CNNs is the use of so-called 'pooling layers'. The underlying principle is quite straightforward: once a receptive field is decided, all pixels in it are converted into a single one, effectively reducing the size of the hidden layer. The algorithms used to perform this operation are several, with the simplest and most common ones being the 'average pooling' (the mean value of the pixels is calculated) and the 'max pooling' (the maximum value is selected). A comprehensive review of pooling strategies may be found at [64]. The best pooling strategy depends, in general, on the physical problem at hand and on the overall NN architecture. The possibility of exploiting the NN itself for learning the most effective pooling strategy has also been explored in [65]. Careful combinations of convolutions and pooling may be used to obtain NNs which are approximately invariant with respect to spatial translations [6, 48, 66].

2.2.3. *Graph-NN.*

Graph-NNs extend the basic principles of CNNs to more general data topologies [67]. Whereas CNNs operate on values organized on regular, orthogonal grids, such as images or volumetric data, Graph-NNs allow the manipulation of general graph-structured data, organized by nodes and edges. These emerge naturally in (computational) physics when dealing with molecular structures [68–70], branched objects such as dislocations [71, 72] and polymer chains [73], crystal-grain microstructures [74–76] and in general for unstructured meshes [77–79].

In a Graph-NN, a hidden state is associated with each node/edge and its value is updated by aggregating information coming from nearby nodes. This may be done by summing an affine transformation of nearby nodal features, as in regular convolutions, or using alternative schemes, such as recurrent structures or the attention mechanism. We point the interested reader to the comprehensive review in [80]. Similarly to other NN structures, it is possible to stack multiple convolution-aggregation units to make the GNN deeper.

2.2.4. *ResNet.*

Residual networks, or ResNets [56], are a particular NN architecture in which specific inter-layer mappings called 'skip connections' are introduced to improve the trainability of the network. In particular, one of the possible issues coming from equations (1) and (2) is the so-called 'vanishing gradient' phenomenon, which may prevent the training of very deep networks (see section 2.5 for more details). The possible solution comes from the modification of the forward-pass rule of the standard fully-connected architecture (equation (1)). For the ith hidden layer:

$$\vec{h}^{(i)} = f\left(\overline{W}^{(i)} \cdot \vec{h}^{(i-1)}\right) + \overline{W}^{(i)}_{\text{skip}} \cdot \vec{h}^{(i-1)} \tag{6}$$

with the convention that $\vec{h}^{(0)} = \vec{x}$, f is the activation function and $\overline{W}^{(i)}$ and $\overline{W}^{(i)}_{\text{skip}}$ are affine transformations, the latter implementing the 'skip connection'. In other words, the scope of the layer is to learn the nonlinear 'residual' with respect to such affine transformations. Since the skip connection is a affine mapping, it is far less prone to the vanishing gradient problem. Indeed, ResNets have been proven to be particularly useful in deep architectures [56]. The introduction of skip connections also has another advantage, as they ensure that the class of functions which can be approximated by the NN increases with depth, a property which is not necessarily true for multilayer perceptrons [48, 56].

2.3. *Derived architectures*

One of the strengths of NN is their intrinsic modularity. The multilayer structure makes it easy to compose and combine models and architectures in order to obtain more effective tools. In this section, we outline some of the main architectures built by combination of base NN blocks as sub-networks, commonly reported for handling physical problems at the continuum scale.

2.3.1. *Autoencoders.*

In many ML applications, it is desirable to reduce the dimensionality of the data. Finding more compact representations is useful as it helps to remove unwanted noise and to focus on important features only. Additionally, this allows for more efficient algorithms and lower memory footprint in computations. This is often performed as a pre-processing step: instead of training a large NN architecture to perform a specific task, it is possible to first reduce the dimensionality of data. A second, more compact network may then be trained as a downstream module. This is for example useful when data are represented by high dimensional objects (e.g. images), but the features required to fully characterize an instance are few and could be condensated into more compact vectors [6, 81]. This concept builds upon the 'manifold hypothesis' [6, 48] (see also section 2.1) and is at the core of latent-space-dynamics models (see section 3.2.2).

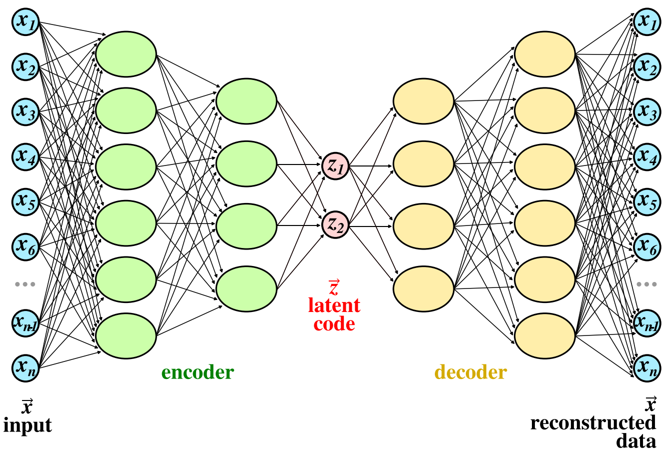

Figure 3. General scheme of an autoencoder, compressing the input data $\vec{x}$ to a two-dimensional space. Both the encoder and the decoder part are parameterized as two-layer perceptrons.

A traditional approach in this regard is the principal component analysis (PCA) [47], which aims at finding a linear combination of the input features which maximizes the explained variance of samples. In mathematical terms, this is equivalent to finding a set of linear operators $\overline{E}$ and $\overline{D}$ which minimize the 'reconstruction' error:

$$\mathcal{L} = ||\vec{x} - \overline{D} \cdot \left(\overline{E} \cdot \vec{x}\right)||^2 \tag{7}$$

Crucially, the rank of $\overline{E}$ is smaller than the dimensionality of the feature vector $\vec{x}$, so that $\overline{E} \cdot \vec{x}$ is the desired compact representation.

Autoencoders (AE) are NN architectures which generalize this idea. Similarly to PCA, the input vector $\vec{x} \in \mathcal{X}$ is compressed into a 'latent vector' or 'latent code' $\vec{z} \in \mathcal{Z}$, with $\dim(\mathcal{Z}) < \dim(\mathcal{X})$, as sketched in figure 3. In this case, the projection is nonlinear and it is performed through a function $e : \mathcal{X} \to \mathcal{Z}$ called 'encoder'. In order to translate back the latent representation in the original space, e should be invertible (or pseudo-invertible). To ensure this, the encoder is trained concurrently with the 'decoder' function d, such that the loss:

$$\mathcal{L}_{\mathrm{AE}} = |\vec{x} - d(\vec{z})|^2 = |\vec{x} - d(e(\vec{x}))|^2 \tag{8}$$

is minimized. Depending on the task, alternative loss functions may be preferable.

An AE is therefore obtained as the encoder/decoder pair that minimizes the reconstruction error. Once trained, e may be used to compress the information in instances of $\vec{x}$, while d can be used to re-project data into their original space. Importantly, the training of AE does not require a dataset of annotated samples, hence it may be regarded as a non-supervised approach and represents an automatic tool to extract from observations a suitable nonlinear projection to the underlying manifold.

If e and d are linear transformations, $\mathcal{X} = \mathbb{R}^n$ and $\mathcal{Z} = \mathbb{R}^m$ ($m < n$), one recovers the PCA. In deep learning, however, encoders and decoders are parameterized as NNs, possibly CNNs or RNNs (see section 2.4) if the dataset is composed of images or time sequences respectively. The advantage is that the nonlinear nature of NN increases the compression rate, at the expense of the higher interpretability of PCA. Of course, in the last years, variants of the simple AE architecture have been proposed. For example, denoising AE [82] (DAE) are trained using corrupted versions on the original dataset and once trained allow to recover partially corrupted information. Contractive AEs [83] (CAE), on the other hand, use specific regularization schemes to try to enforce geometric properties to the latent space $\mathcal{Z}$. Finally, Variational AE [84] (VAE) enforce an assigned probability distribution to the latent space, so that once trained, they can also be used to extract new, unobserved samples coherent with the same distribution of the training dataset, thus behaving as a generative model (i.e. a model capable of sampling from a learned distribution). An in-depth review of AEs and their applications can be found in [81].

2.3.2. U-Net.

U-Nets are a specialized architecture initially developed for image-to-image translation and image segmentation (i.e. pixel-level classification), obtained by composing convolutional layers and skip connections [85], as exemplified by the scheme in figure 4. At first glance, the structure of this NN resembles that of an AE: the input image is progressively reduced in resolution (contracting path), which is then increased again (expanding path) to match the initial one. The organization and size of the layers resemble a U shape, giving the architecture its name. The progressive compression in the contracting path recalls a sort of 'hierarchical' manifold hypothesis, with global (or large-scale features) living in a smaller space than local (short-scale) ones. At variance with AE, however, there is no independent latent code $\vec{z}$ in this case, as the presence of the skip connections makes the reconstruction from the lowest resolution features impossible without the intermediate stages too. Additionally, the input and output of a U-Net are not required to be the same. The main advantage of U-nets over standard CNNs comes from an easier integration of long-range dependencies while retaining high-resolution details thanks to the skip connections between the initial downscaling layers [85, 86]. This makes them particularly well-suited for applications in which long-range interactions cannot be neglected.

2.3.3. Deep Operator Networks (DeepONets).

DeepONets [87] are specialized architectures devised to approximate the mapping between infinite-dimensional function spaces, instead of vector-valued functions. The core idea is based on a specialized version of the UA theorem for operators [88]. Given the operator G acting on a function ν so that $u = G[\nu]$, $u(\vec{x})$ may be approximated as a function of $\vec{x}$ and a discretized version of ν. Several discretization strategies are possible, such as simple function evaluation on collocation points or projection on basis functions. Further details can be

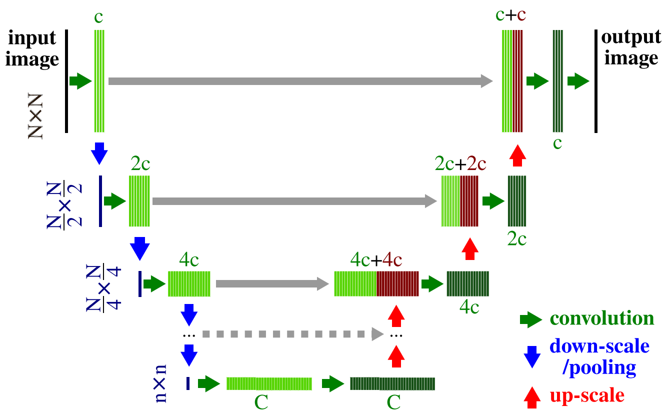

Figure 4. Simple scheme of a U-net architecture. On the left, the contractive path progressively reduces the resolution of the input image from $N \times N$ to $n \times n$. Correspondingly, the number of channels is increased, from c to C, as more complex 'semantic' information is extracted. On the right, the expanding path upscales the resolution back to the one required for segmentation. At the same time, the number of channels is progressively reduced. Information at intermediate resolutions is propagated through skip connections (grey arrows) and concatenated with the data coming from lower levels.

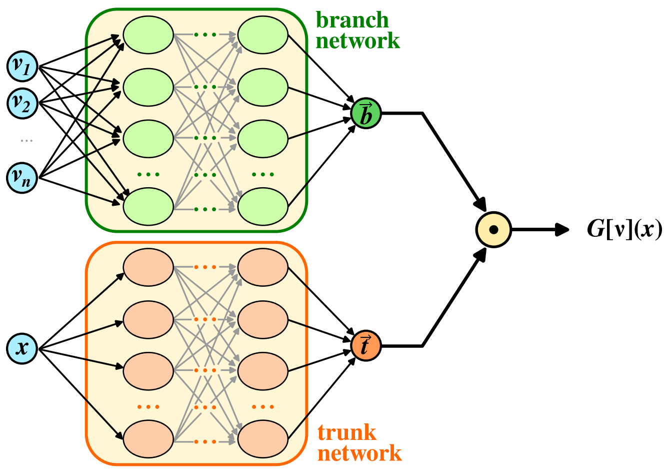

Figure 5. DeepONet scheme for an operator acting on one-dimensional functions. The NN architecture is composed of two sub-networks. The branch network takes a discretized version of the input function ν, i.e. the scalar components $\nu_1, \ldots, \nu_n$. The inputs to the trunk network, instead, are the coordinates at which the output function $u = G[\nu]$ should be evaluated. In this case, the single spatial coordinate x.

found in [89] and references therein. In practice, DeepONets have a structure suggested by the aforementioned theorem and are composed of two subunits, called 'trunk' and 'branch' networks (see figure 5). The role of the branch network is to perform a nonlinear projection of the discretized ν to $\vec{b} \in \mathbb{R}^p$. Similarly, the trunk net takes as input the coordinate point at which u is to be evaluated and also outputs a vector $\vec{t} \in \mathbb{R}^p$. The width p is a network hyperparameter. In both structures, the input may also be augmented with additional context information and specialized structures may be used to enforce desired properties, such as specific boundary conditions [89]. The value of $u(\vec{x})$ is then approximated as:

$$u(\vec{x}) \approx \vec{t}(\nu) \cdot \vec{b}(\vec{x}) + b_0 \tag{9}$$

being b_0 an additional bias parameter, $\cdot$ the regular scalar product between vectors and, with abuse of notation, ν is the set of discretized values of the input function.

2.4. Recurrent architectures

RNNs are NNs specialized in the manipulation of sequences, such as time series (or natural language as in the first applications). In this section, we will indicate the elements of an input sequence of a RNN as $\vec{x}_t$, with t a discrete index indicating the position in the sequence. Recurrent architectures are capable of modeling explicitly non-Markovian systems and input series which are not fixed in size by introducing a hidden state $\vec{H}_t$ at every time t. From a heuristic point of view, this acts as a memory for the NN. $\vec{H}_t$ is updated in time by means of an update function g, usually parameterized as a NN:

$$\vec{H}_t = g\left(\vec{x}_{t-1}, \vec{H}_{t-1}\right) \tag{10}$$

Different choices of g may lead to different performances and issues. If it is parameterized by a standard NN composed of affine-transformation/nonlinear activation, the resulting architecture may exhibit training stability problems, such as 'vanishing gradients' or the 'exploding' counterpart [90] (see section 2.5). This choice is sometimes referred to as RNN without further specifications (see figure 6(a)). The two most used alternatives are the long short-term memory (LSTM) [91] architecture and the gated recurrent unit (GRU) [92] (see figures 6(b) and (c)) respectively for a schematic representation). Both these structures make use of multiplicative gates involving the hyperbolic tangent and the logistic sigmoid σ. More concretely, based on the NN input, the specialized architecture selects which hidden state components should be updated, retained (i.e. multiplied by one) or 'forgotten', (i.e. multiplied by zero). This approach is proven to solve gradient problems [6, 48, 91, 92], making training stable and improving the information flow from distant times in the sequence. LSTM was the first one to be proposed, uses an extra gate and supplements $\vec{H}_t$ with an additional 'cell' state ($\vec{C}_t$ in figure 6(b)). The GRU, on the other hand, operates on the hidden state alone, thus being cheaper in terms of parameters. It should be pointed out, however, that there is no consensus on which architecture is consistently the most effective [93–95].

The hidden state $\vec{H}_t$ may be transformed into the required quantity $\vec{y}$ by means of a second subnetwork. Depending on the problem at hand, $\vec{y}$ may be a time-dependent property, so that $\vec{y} = \vec{y}_t$ for all timestamps t. A scheme depicting this application is shown in figure 6(d). In this picture, time dependency is 'rolled' and there is a connection exiting and entering the same computational graph node. The meaning of this is better explained in the 'unrolled' representation of figure 6(e),

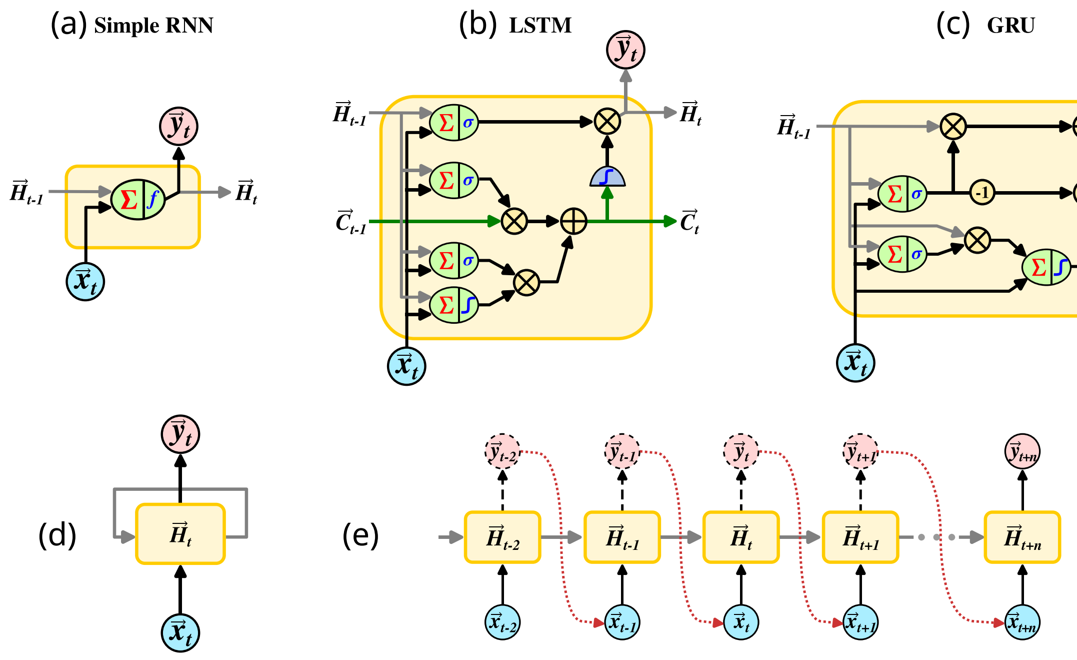

Figure 6. (a)–(c) Schematics of common RNN architectures. σ represents the logistic sigmoid, f a generic activation function, $\int$ is the hyperbolic tangent and $\otimes$ represent component wise (or Hadamard) product. (a) Simple RNN with no gating mechanism. (b) Long short-term memory (LSTM) recurrent network. (c) Gated recurrent unit (GRU). (d) Compact representation of a RNN working. (e) 'Unrolled' representation, in which time dependencies between subsequent time step predictions are shown explicitly. The conversion $\vec{H}_t \to \vec{y}_t$ at every timestep (dashed black lines) is performed only if the quantity of interest is a time series too. Connections of $\vec{y}_t$ and $\vec{x}_{t+1}$ (dashed red) are used only in closed-loop mode.

where the temporal dependence is explicitly shown. If the target $\vec{y}$ is a property characterizing the whole sequence, such as in time sequence classification, only the last hidden state $\vec{H}_t$ is instead considered for the output. This case is also particularly well suited when sequences of arbitrary length should be converted into a fixed-size representation or quantity, as in video classification.

RNNs not only may consume sequences as inputs but also are capable of sequentially generating them. Conceptually, this is done by setting $\vec{y}_t = \vec{x}_{t+1}$. A NN with this property may be obtained by feeding back the RNN output at time t as a new input (autoregression or 'closed-loop' prediction), a strategy which is commonly used also in text-to-text translation [6, 48, 92]. In practice, the input at time t, $\vec{x}_t$ is converted together with the hidden state $\vec{H}_t$ (which contains a summary of important features of the whole previous states of the system) by the RNN into $\vec{H}_{t+1}$. This is itself converted into the prediction of the next sequence element by means of a second subnet and passed to the RNN as the next input. A whole sequence is thus generated sequentially (i.e. using the dashed red connection in figure 6(e)). It is worth mentioning that particular care is necessary when training networks which re-use their own outputs: during training, in principle, the whole time sequence is available, hence it is possible to provide the RNN with the 'correct' input at every time step. Once the training is terminated, however, this is no longer the case. This mismatch between training and evaluation may be resolved, at least in part, by generating predictions in closed-loop at training time too, which however requires higher computational costs. Several, more advanced strategies are also available in textbooks [6, 48] and specialized publications [96, 97].

Recurrent architectures can be conveniently combined with any other NN structure. The most important case in the present context is that of convolutional recurrent networks (CRNNs), in which an LSTM [98] or a GRU [99] architecture substitute standard, vector-valued gating mechanisms with convolutional layers, allowing one to process sequences of images while retaining local spatial correlations and allowing pixel-wise predictions. A more sophisticated architecture for dealing with spatiotemporal data comes from PredRNN [100], which exploits zigzag connections between multiple stacked convolutional blocks to improve the NN predictive capabilities.

2.5. *Training generalities*

As discussed in section 2.1, a NN defines a parametric family of functions. Training may then be interpreted as an optimization problem, i.e. finding the set of parameters $\vec{\vartheta}^*$ which yields the 'best' function approximation. Given a dataset, it is straightforward to formulate the optimization problem through an error function commonly called 'loss function' $\mathcal{L}$. One of the most common classes of loss functions is defined as:

$$\mathcal{L}_{Lp}\left(\vec{\vartheta}\right) = \frac{1}{N_{\mathrm{TS}}} \sum_{i=1}^{N_{\mathrm{TS}}} ||\hat{\vec{y}}_i\left(\vec{x}_i|\vec{\vartheta}\right) - \vec{y}_i\left(\vec{x}_i\right)||_p \tag{11}$$

where the hat on top of $\vec{y}$ is used to distinguish between the NN prediction of the dependent variable $\hat{\vec{y}}$ and the training set instance $\vec{y}$, N_{TS} is the number of examples in the training set, $p \geq 1$ and $||\cdot||_p$ is the pth power of the Lp norm defined as

$$||\vec{x}||_p = \sum_{j=1}^{\dim(\vec{x})} |x_j|^p \tag{12}$$

The index j runs here on the vector components and not training instances. For the case $p = 1$ one recovers the mean absolute error (MAE), while for $p = 2$ one obtains the mean squared error (MSE) $\mathcal{L}_{L2}$, which is also commonly used in standard linear regression. The rationale behind equation (11) is clear: the loss function is zero if and only if the NN is reproducing all the correct $\vec{y}$ values for every given $\vec{x}$, i.e. it is perfectly reproducing the training set.

Of course, Lp norms do not comprise all the explored loss functions. Another important class of losses comes from modifications and generalization of the binary cross entropy (BCE) loss used to compare scalar y representing a probability:

$$\begin{aligned}\mathcal{L}_{\mathrm{BCE}}\left(\vec{\vartheta}\right) = & \frac{1}{N_{\mathrm{TS}}}\sum_{i=1}^{N_{\mathrm{TS}}} y_i \log\left(\hat{y}_i\left(\vec{x}_i|\vec{\vartheta}\right)\right) \\ & + (1-y_i)\log\left(1-\hat{y}_i\left(\vec{x}_i|\vec{\vartheta}\right)\right).\end{aligned} \tag{13}$$

A comprehensive list and description of alternatives may be found in [101]. In general, $\mathcal{L}$ may be designed based on the specific training task.

For completeness, it should be mentioned that, in some contexts, loss functions may be justified or derived from *maximum likelihood estimation principles* or *maximum a posteriori estimations* [6, 8, 47, 48]. However, in the present Review, we will mainly follow a heuristic perspective and interpret the loss function as an object which can be adapted to the task at hand.

Through the loss function, the training problem is cast as an optimization one. An efficient algorithm to minimize $\mathcal{L}$ is therefore necessary. Indeed, we recall that the NN is intrinsically a nonlinear function and, at variance with linear models, there is no closed form solution for $\vec{\vartheta}^* = \mathrm{argmin}_{\vec{\vartheta}}\mathcal{L}(\vec{\vartheta})$. Optimization is therefore usually performed using some variant of gradient descent: given an initial set of parameters $\vec{\vartheta}$, the closest (local) minimum of $\mathcal{L}$ is found following the negative gradient $-\vec{\nabla}_{\vec{\vartheta}}\mathcal{L}$. Parameters are moved proportionally to this direction, with the proportionality constant used for the move called 'learning rate'. The study and implementation of gradient-based algorithms is still a very active area of research. Out of the many proposed approaches, the most common ones are probably the so-called stochastic gradient descent (SGD) [102], RMSProp [103] and especially ADAM [104]. All these algorithms are, to some extent, devised to modify standard steepest descent protocols to avoid trapping in high-loss local minima or to speed-up convergence. An important modification which is present in all these approaches is that, instead of calculating the gradient of $\mathcal{L}$ over the whole training set at each minimization step, it is customary to randomly select smaller samples, known as 'mini-batches'. This makes the algorithms stochastic, which returns advantages both in terms of convergence quality and speed [6, 48, 105]. Once the whole training set has been presented to the NN via mini-batches, one training 'epoch' has ended.

One of the motivations behind the NN success is that $\vec{\nabla}_{\vec{\vartheta}}\mathcal{L}$ may be efficiently obtained through the so-called 'back-propagation' approach: by repeatedly applying the chain rule for derivatives of composite functions, the NN layers are traversed from the output nodes to the input ones, calculating the required quantities leveraging dynamic programming approaches. Crucially, the gradient may be calculated exactly through this (reverse mode) automatic differentiation [106] and the result is not affected by approximation errors as in finite differences or similar schemes. Naturally, this means that well-behaved gradients are necessary to effectively perform training. Indeed, gradient-related issues are especially relevant in the case of very deep NN and when using saturating activation functions [6, 48], because of the repeated application of nonlinear maps. In such cases, $\vec{\nabla}_{\vec{\vartheta}}\mathcal{L}$ tends to vanish ('vanishing gradients'), thus making the training very slow and impractical. The opposite situation of divergent $\vec{\nabla}_{\vec{\vartheta}}\mathcal{L}$ is also possible and is referred to as 'exploding gradients'. To mitigate these risks, suitable architectures such as ResNets (section 2.2.4) and LSTMs (section 2.4) have been developed.

From a technical point of view, back-propagation can be very efficiently performed with the use of GPUs [6, 107, 108]. Fortunately, there is no need to implement back-propagation from scratch, as many frameworks are already available. The most used ones are Tensorflow [109], Pytorch [110] and JAX [111].

2.5.1. *Overfitting and regularization.*

Some aspects of training differ substantially from pure optimization: NNs are strongly nonlinear functions, hence $\mathcal{L}$ may have a plethora of local minima which may yield poor results. This problem is partly addressed by specialized training algorithms and may be alleviated by training multiple NN models with different random initial conditions. In reality, there is at least one additional subtlety: reaching the global minimum of $\mathcal{L}$ does not guarantee that predictions on new data are accurate, even though the NN perfectly reproduces the training set. The situation in which a NN has a small loss on the training set but fails dramatically for other $\vec{x}$ values is called 'overfitting' [112]. This risk is particularly present with very large networks, in which the number of parameters may even exceed the number of examples in the training set. A practical solution for this problem is based on using an additional dataset, called 'validation set': while $\mathcal{L}$ is being minimized based on the training set examples, its value is also monitored on the validation set. If overfitting occurs, the validation loss increases while the training value continues decreasing, and the training should end (early stopping). It is customary during the NN optimization procedure to select the model which yields the lowest validation loss. An example of a training loss plot involving overfitting is provided in figure 7. Notice the characteristic presence of oscillations, more evident for the validation loss, due to the stochastic nature of the optimization algorithm.

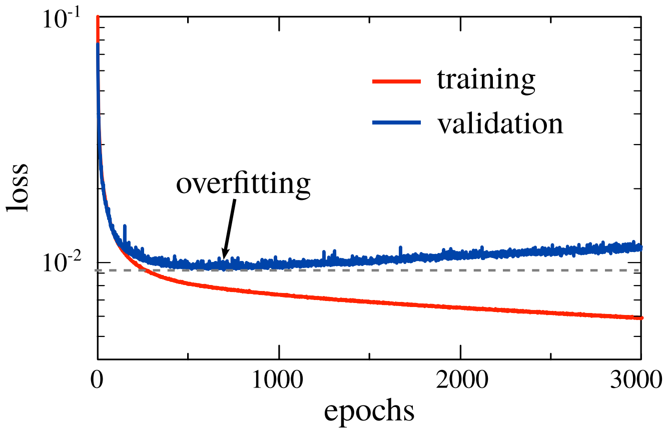

Figure 7. Example of a training and validation loss curves showing overfitting. The same CNN architecture of [113] is used but on a reduced version of the training set. The dashed gray line corresponds to the minimum of the validation loss. The presence of oscillations (stronger for the validation curve) is due to the use of stochastic optimization methods, here ADAM.

Another useful strategy is 'regularization': the mathematical expression of $\mathcal{L}$ is modified in order to have a trade-off between a desirable NN property and perfect adherence to the training values. This introduces an effective constraint between parameter values, mitigating the overfitting problem. As a simple example, we consider the so-called $L2$ regularization, in which the $L2$ norm of the set of parameters is penalized, enforcing small values for weights and biases. Once again, this apparently arbitrary choice may be linked to the statistical theory of *maximum a posteriori estimation* [6, 48].

Using the MSE loss function, the L2 regularized $\mathcal{L}$ reads:

$$\mathcal{L}\left(\vec{\vartheta}\right) = \frac{1}{N_{\mathrm{TS}}}\sum_{i=1}^{N_{\mathrm{TS}}} ||\hat{\vec{y}}_i\left(\vec{x}_i|\vec{\vartheta}\right) - \vec{y}\left(\vec{x}_i\right)||^2 + \lambda \sum_{j=1}^{\dim\left(\vec{\vartheta}\right)} \vartheta_j^2. \quad (14)$$

Notice that in equation (14) a new quantity, the scaling factor λ, controlling the regularization strength has been introduced. Since λ defines the training procedure itself, it is commonly referred to as a 'hyperparameter' (see section 2.5.2).

Another regularization possibility is 'data augmentation' [6, 48]. The core idea is that, instead of increasing the size of the training set with new data, one may generate suitable new examples from the ones already available. In practice, before passing inputs to the NN, symmetry operations (e.g. reflections, rotations, etc) are applied, effectively increasing the diversity of examples. While this strategy is useful for reducing overfitting issues, it has to be remembered that modified versions of input data are correlated, hence data augmentation cannot fully substitute the collection of new, independent data. A comprehensive survey on possible augmentation strategies may be found at [114].

A related idea is the injection of noise (or 'instance noise') into the input of a NN [6, 48, 115, 116]. The idea is that slightly perturbed inputs should provide the same output, making the NN less prone to catastrophic failure in the vicinity of training points. Indeed, these techniques penalize NN whose output is rapidly fluctuating and is proven to be equivalent to $L2$ regularization [117] under suitable conditions.

It is important to mention that in the context of approximating the evolution of physical systems, the loss function may be conveniently modified with terms which contain symmetries or conserved quantities. Similarly, data augmentation techniques involving symmetry operations may be exploited. One has to remember, however, that regularization implies a trade-off in the training objective, hence properties will be satisfied only approximately, i.e. they are enforced only in a weak sense. Another, more effective strategy to achieve this goal is to inscribe properties into the NN architecture.

2.5.2. Hyperparameters tuning.

In all the previously introduced architectures, the NN structure itself has to be defined before training starts. For example, when using CNNs, the receptive field (i.e. the dimension of the convolution kernel) and the number of channels have to be chosen. Indeed, for every deep NN architecture, the number of layers to be stacked has to be defined. Similarly, there may be additional quantities regulating the training process, such as the regularization strength λ (equation (14)) or the learning rate itself. All these are collectively called 'hyperparameters' [6], to distinguish them from values in $\vec{\vartheta}$ which regulate the input-output mapping of the NN.

The effectiveness of the trained model and its performances are strictly related to the specific choice of hyperparameters. Unfortunately, the inherent nonlinear nature of NNs and the huge number of parameters involved, makes hyperparameters selection and tuning a critical problem. Indeed, one often has to resort to trial and error to find a reasonable set of values. If the number of hyperparameters is small, it may be possible to perform a grid search, monitoring the training/validation loss to select the optimal combination or even obtaining the optimal values by gradient minimization [118]. A plethora of 'hyper-optimization' strategies have been developed and this is still an open area of research. Technical aspects are thoroughly discussed in [119]. Additionally, we remark that tools, such as Optuna [120], have been developed to automatically perform some of these operations.

2.6. System descriptors and performance metrics

In the context of using NNs to predict the (time-dependent) solution of PDE problems, the evaluation of the model performances and reliability can be assessed both at a local, 'pixel-wise' level and at a global level.

Since the NN is trained to minimize a training loss (e.g. MSE, MAE etc see section 2.5), it may be used as an estimator of the match between predicted and true values either at the local or global level according to its definition. This, however, has the possible issue that the loss function may be an abstract quantity, lacking a simple and direct physical interpretation. Different descriptors may therefore be preferable.

2.6.1. Local metrics. At the local level, the objective is to determine how well the NN model predicts the exact fields on the chosen domain, at the precise integration time of the ground-truth PDE solution. This sets the most strict fidelity metrics as evaluating how the NN model can be considered as a one-to-one replacement of the true solution.

The most obvious way to assess the one-to-one match between the predicted and true fields DNS is to measure their difference at each pixel/voxel on the grid. By constructing an error image it is possible to identify whether the NN faults are generalized or related to specific local features. In this latter case, an expansion of the training set to better represent such critical conditions could be sufficient to improve the NN performance.

A typical indicator of the match is the intersection-over-union (IOU) [48] and its generalizations [121]:

$$\mathrm{IOU} = \sum_{j=1}^{\dim(\vec{y})} \frac{\hat{y}_i \cdot y_i}{\hat{y}_i + y_i} \tag{15}$$

in a strict sense, the formula is defined for binary values only ($y_i \in \{0,1\}$), hence it is particularly fitting for segmentation tasks (e.g. pixel-level classification) or PF-like images.

Another indicator, which may be used for continuous variables, is the pixel-wise Pearson correlation coefficient [122]:

$$\rho_{y\hat{y}} = \frac{\sigma_{y\hat{y}}}{\sigma_y \sigma_{\hat{y}}} \tag{16}$$

where $\sigma_{y\hat{y}}$ is the covariance and $\sigma_y, \sigma_{\hat{y}}$ are the standard deviations for y and $\hat{y}$ respectively.

It must be however noticed that a strict pixel-by-pixel comparison can return large errors even if the prediction is overall satisfactory but subject to trivial mismatch with the original pixels, e.g. because of slight shifts or misalignments. Different indicators, mostly inherited from computer vision tasks, have been devised to evaluate the actual extent of similarity between two image-like objects, i.e. the true and predicted fields. We here mention two among the most commonly used [123], the peak signal-to-noise ratio (PSNR), set on a dB scale as:

$$\mathrm{PSNR} = 20 \cdot \log_{10}\left(\frac{\max_I}{\sqrt{\mathrm{MSE}}}\right) \tag{17}$$

where $\max_I$ is the maximum value of the image, and the structural similarity index metric (SSIM):

$$\mathrm{SSIM}(y,\hat{y}) = \frac{(2\mu_y\mu_{\hat{y}} + C_1)\cdot(2\sigma_{y\hat{y}} + C_2)}{\left(\mu_y^2 + \mu_{\hat{y}}^2 + C_1\right)\cdot\left(\sigma_y^2 + \sigma_{\hat{y}}^2 + C_2\right)} \tag{18}$$

where $\mu_y, \mu_{\hat{y}}$ are the mean values, $\sigma_y, \sigma_{\hat{y}}$ the standard deviations, and $\sigma_{y\hat{y}}$ the covariances and C_i stabilization constants.

2.6.2. Global metrics. Global indicators are to be consider whenever the focus is directed on the capability of the NN model to return the appropriate dynamics of the whole system, independently of how well it matches the PDE solution at the local scale. This is especially relevant in chaotic dynamics or in the presence of high noise levels that can lead to bifurcations in the dynamics which return different solution patterns even when they correspond to equivalent global properties.

Suitable descriptors are to be devised to evaluate the relevant properties qualifying the system state, and their time evolution is to be monitored to verify the expected dynamics.

On the one hand, for each class of problems, one can identify some known physical descriptors, e.g. energy, geometry-related quantities such as volumes, surface, profile curvatures and aspect ratios. Similarly, properties such as statistical parameters and autocorrelation functions, characterizing the state and evolution of the system, may be used if appropriate. Some examples of this type of analysis have been performed in the literature (e.g. [75, 124–126] to cite a few).

Global state features can also be inferred *a posteriori* from a dataset of possible solutions by extracting a low-dimensional set of latent-space variables, encoding the major traits distinguishing each configuration. Different techniques are exploited to determine such latent-space descriptors, from linear PCA to more refined nonlinear mappings, such as Isomap or UMAP [127, 128].

NN architectures are themselves developed to infer from the data some latent-space descriptors that better condensate the information. Among the alternatives, AEs are built with the target of reducing data on a lower-dimensional space (see section 2.3.1). Similarly, the hidden layers of CNN (section 2.2.2) can be thought of as maps of the relevant features in the data images based on the inferred kernels.

The data-driven identification of representative latent features is a physically relevant achievement on its own for understanding the main contributions impacting an observed phenomenon. Their use as metrics to evaluate the NN prediction performances is then conditioned by how effective they are as descriptors of the problem under scrutiny.

3. Data-driven models

In this section, we discuss the class of data-driven approaches, i.e. supervised learning methods which take profit of the approximation capabilities of NN to accelerate the simulation of a time-evolving physical system, starting from a dataset of examples generated by DNS [43]. Two conceptually different strategies can be considered.

In the first, the NN models are deployed for supporting the conventional DNS, e.g. by providing fast predictions of bottlenecking contributions in coupled PDE problems or corrective terms to improve model approximations. In this scenario, the time integration is still performed using classical numerical schemes (e.g. Euler method), but one or more terms are substituted/integrated by lightweight, yet accurate NN approximations. A discussion of literature studies pursuing this strategy is reported in section 3.1.

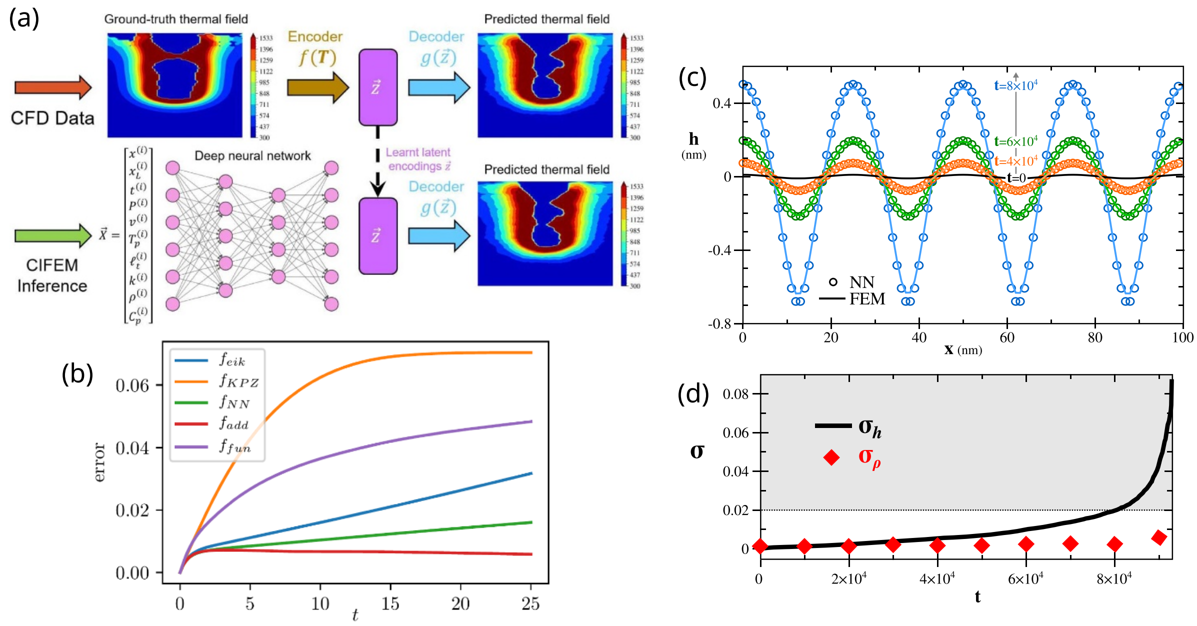

Figure 8. (a) Schematic representation of the NN approach from [131], exemplifying the thermal field prediction compared to the CFD calculation. Reprinted from [131], Copyright (2022), with permission from Elsevier. (b) Time-evolution of the spatially averaged absolute error with respect to the true Allen–Cahn solution when using DNS approximations (i.e. eikonal and small-slopes KPZ approximations) and NN models (full NN surrogate model, additive corrective term to KPZ and functional correction from KPZ equation). Reprinted figure with permission from [132], Copyright (2023) by the American Physical Society. (c) and (d) Profile evolution sequence (c) and error evaluation (d) for a simulation of ATG instability of a strained film, obtained by using a NN prediction of the elastic driving force rather than its explicit computation by FEM. σ is the root-mean-squared error for the NN prediction of both the elastic energy density ρ and the surface profile h during the time evolution. Reproduced with permission from [113]. Copyright © 2024 Authors. This is an open access article reproduced with permission under a CC BY-NC license (https://creativecommons.org/licenses/by-nc/4.0/) CC BY-NC 4.0.

In the second approach, investigated more in-depth in section 3.2, the NN model is constructed to directly produce the temporal sequence of system states for the given evolution problem, thus completely replacing the DNS integration scheme from which it is trained.

3.1. Enhancement of DNS approaches by NNs

The ability of NNs to produce high-fidelity solutions to 'static' PDE problems at a fraction of the computational cost required by DNS has been widely demonstrated (see e.g. [129, 130]). This can be conveniently exploited in multi-physics, coupled problems where the evaluation of such data can be part of the full solution process. The gain in replacing the explicit calculation of computationally intensive terms can be dramatic when considering time integration as their cost is to be multiplied by the number of time steps. In the case of severe bottlenecks, resorting to a NN model might then return substantial speed-ups in the simulation wall time or even be the only option enabling to perform calculations that would be otherwise too demanding for DNS.

An application of this idea can be found in the work by Strayer *et al* in [131] for simulating the dynamics of laser-induced local fusion in additive manufacturing. In that case, the accurate solution of the thermal field and the dynamics of the liquid pool can be achieved only by very demanding computational fluid dynamics (CFD) calculations, on highly refined meshes, thus making the cost of large-scale simulations almost prohibitive. In contrast, long-range thermal transport could be traced by finite element method (FEM) with lower computational effort. The idea, illustrated in figure 8(a), was then to replace the costly CFD calculations by training a NN model capable of predicting its solution within a critical region and then plug it into a FEM solver to extend the solution to the large-scale domain. To such a goal, the CFD data are first processed via a CAE to reduce the very dense mesh of CFD into a low-dimensional latent-space representation. Then, the encoded data are used as output for a deep NN which takes in input a simplified definition of the system state/variables from the CFD-informed FEM (CIFEM) so that, after training, it can predict the latent-space representation of the desired CFD solution for an arbitrary input. Last, the NN output is decoded back to a real-space thermal field to be embedded in the FEM solver. This latter is used to extend the solution to the whole domain by heat diffusion and then advance to the next time step. A speed-up of ≈28× is achieved compared to CFD-based simulations while preserving accuracy with a mean absolute relative error below 10%.

Another case is found in the work by Lanzoni *et al* in [113] for the study of the morphological evolution of strained films [133]. There, a 2D surface profile is evolved by surface diffusion dynamics following local gradients of chemical potential. This latter includes a local contribution from surface energy, conveniently computed by a low-cost finite difference scheme, and an elastic energy contribution which involves the more demanding solution of the mechanical equilibrium problem via FEM. Then, a CNN is trained to predict such elastic contribution directly from the surface profile, so that it could be used as input to solve the diffusion equation. As reported in figure 8(c) for a benchmark simulation of Asaro–Tiller–Grinfeld film instability [133], the time-integration using the NN prediction of the elastic term closely matches the ground-truth FEM evolution yielding a one-to-one reproduction with root-MSE of few % up to the latest diverging stages. A 4-orders-of-magnitude shrinkage of simulation wall time is reported while retaining numerical stability for millions of integration time steps. The study also highlights the generalization capabilities of the trained model towards diverse morphologies and the possible application to domains much larger than those used for the training, thanks to the use of a fully CNN architecture. An in-depth analysis of prediction errors and extrapolation limits for a similar study but using a Green's function approximation of the elastic problem is reported in [134].

A different target is the one proposed by Kemeth *et al* in [132] where a feed-forward fully-connected NN model is devised to learn the correction to a known analytic approximation of a PDE. More precisely, the Allen–Cahn equation [41] is taken as the targeted numerical solution of interface dynamics in 2D. For small slopes, it can be approximated by the deterministic Kardar–Parisi–Zhang (KPZ) equation [135], which is its analytical reduction to a lower-dimensional front-tracking problem. The NN is then tasked to find the correction that applied to the KPZ solution gives back the Allen–Cahn result, even beyond the limits of validity of the KPZ approximation itself. To such a goal, the NN is trained to learn the difference $(\partial h/\partial t)_{\mathrm{PF}} - (\partial h/\partial t)_{\mathrm{KPZ}}$ with $\partial h/\partial t$ the local interface velocity obtained respectively from the motion of the PF $\varphi = 0$ iso-line and from the analytical expression $f_{\mathrm{KPZ}}(h, \partial h/\partial x, \partial^2 h/\partial x^2)$ set by the KPZ equation. The alternative strategy of learning the correction directly from the KPZ approximated solution is also considered but yielded less accurate predictions. The efficacy of this strategy is made evident in figure 8(b) where the time evolution of the absolute error obtained for different DNS approximations and using NN models are compared.

It is worth noticing how, for all of these approaches, the NN model is clearly identified with some physical term, retaining its connection and role within the overall problem to be solved. This ensures a full transferability of the NN models to any other PDE system containing the same contributions, without having to repeat the training for every variant of the physical equations.

3.2. NN surrogate model for dynamics

The potential of NNs as universal approximators can be fully leveraged to train models capable of predicting the actual evolution sequence for an assigned time-dependent problem, initial conditions and eventual parameters.

From the perspective of time-dependent problems, the NN can be generally considered as a 'time-stepper' or 'time-solver' $\mathcal{S}$ capable of updating the system state, described by a feature vector $\vec{x}(t)$ at a time t to its new configuration at a subsequent time step $t+\tau$, i.e.

$$\vec{x}(t+\tau) = \mathcal{S}_t(\vec{x}(t)) \tag{19}$$

where the operator $\mathcal{S}_t$ can also be time-dependent.

More precisely, the NN can be trained to directly predict the new state vector $\vec{x}(t+\tau)$ or its variation with respect to the previous one $\delta\vec{x} = \vec{x}(t+\tau) - \vec{x}(t)$ (residual learning). Other choices can be devised to enforce the peculiar properties of the solutions. For instance, in [125], the NN output is set as the flux vector $\vec{J}$ such that the variation of the scalar quantity x is given by $\delta x = \vec{\nabla}\cdot\vec{J}$, enforcing a local conservation law by construction.

Applying iteratively the procedure in equation (19), feeding the NN with the previously predicted system states (see figure 6(e)), allows in principle to reconstruct the entire evolution sequence as

$$\vec{x}(t) = \mathcal{S}_t(\mathcal{S}_{t-\tau}(\ldots\mathcal{S}_0(\vec{x}(0)))) \tag{20}$$

As for any dynamics to be solved in the forward time direction, this iterative approach leads to error accumulation. Indeed, new predictions build on top of the previously approximated ones, so that the accuracy of the solution at a time t will depend on the accuracy of previous time steps. This limits the extent of the sequence that can be predicted by the NN model as well as its extrapolation. For example, most of the studies show that, after some time, one-to-one accordance with the NN prediction is lost due to error accumulation [76, 136]. An improvement can be expected by exploiting recurrent schemes to better capture time correlations (see section 2.4). Several works have indeed shown that the use of RNN allows to obtain time evolutions with small error accumulation for sequences which are several times longer than those used in training [75, 124, 126, 137], and in some cases even reaching the correct stationary state [125]. Still, the search for effective strategies to obtain stable time evolution schemes is one of the main directions in the field. In figure 9, comparisons between NN predictions and the solution obtained by DNS are exemplified for different physical problems: (a) grain coarsening; (b) domain dynamics in ferroelectric materials; (c) directional solidification of polycrystals. These highlight how trained NN models can achieve a compelling reproduction of the true fields, especially on the training-set time scale. Typical prediction errors can however be observed as local discrepancies possibly related to bifurcating events in the dynamics.

The NN training takes data sequences as input and aims at the minimization of the overall loss function on all sequence frames, as here given for MSE:

$$\mathcal{L}\left(\vec{\vartheta}\right) = \frac{1}{N_{\mathrm{TS}}}\frac{1}{T}\sum_{i=1}^{N_{\mathrm{TS}}}\sum_{t=1}^{T}\left[\hat{\vec{y}}\left(\vec{x},t|\vec{\vartheta}\right) - \vec{y}(\vec{x},t)\right]^2 \tag{21}$$

with T the number of time-frames in the NN sequence.

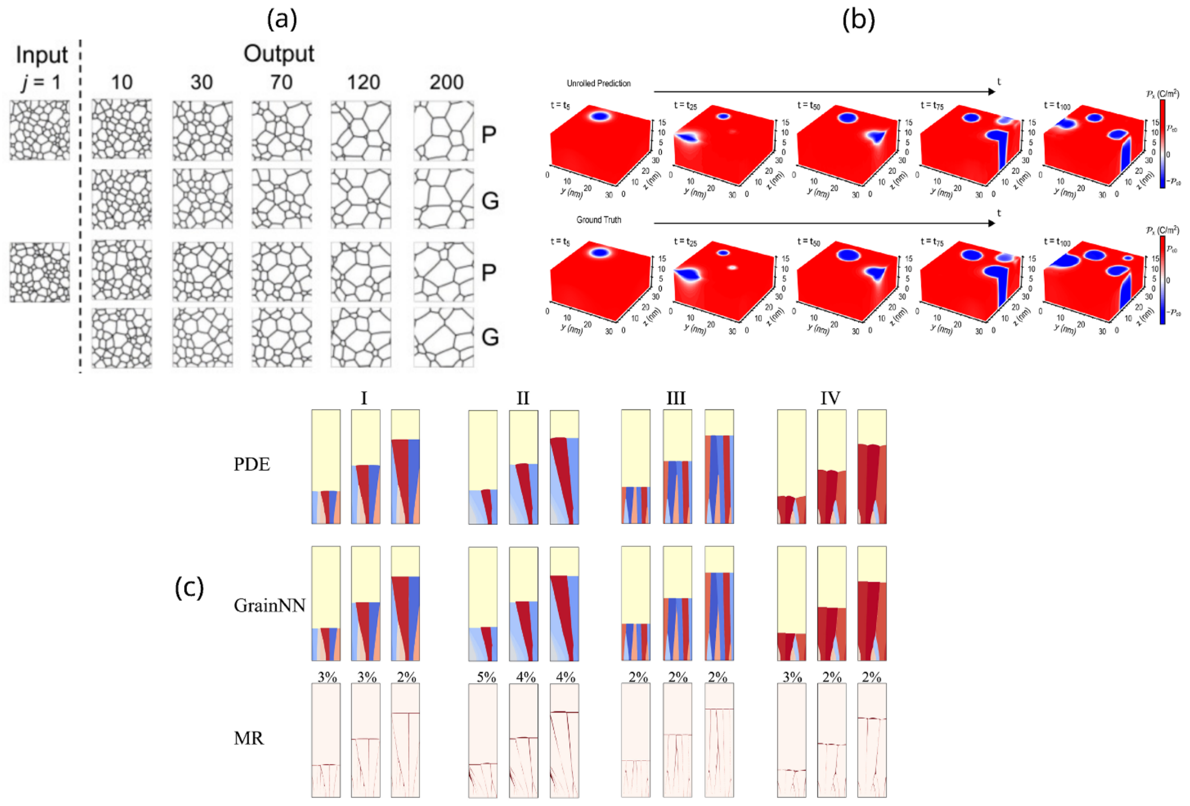

Figure 9. Examples of NN predictions and comparison with the corresponding ground truth evolution from DNS. In well-trained models, local deviations may be observed, also due to error accumulation, but the overall dynamics are consistent with the underlying physical process. (a) Examples of grain microstructure evolutions as predicted by NN predictions (P) and obtained via DNS (ground truth—G) in 2D. Due to small error accumulations, the evolutions tend to lose one-to-one accordance as the time sequence advances. Predictions are still consistent with the general underlying physical process. Reprinted from [126], Copyright (2021), with permission from Elsevier. (b) Comparison of the polarization field in ferroelectrics as predicted by NN (using a U-Net architecture) and DNS. Reproduced from [76], with permission from Springer Nature. (c) Epitaxial grain growth simulations generated by a specialized LSTM architecture using geometrical features as input. The Misclassification Rate (MR) is reported on top of error maps in the third line. Reprinted from [138], Copyright (2023), with permission from Elsevier.

The training data are produced by DNS using conventional computational methods [43]. This of course requires that such a model exists, that it provides a sufficiently accurate representation of the physical problem, and that its solution times are short enough to perform the large number of simulations needed to produce the dataset. A fair evaluation of the cost-benefit balance of developing a NN model should carefully account for the time and resources needed for its training, which can become very demanding if trying to accurately map a wide range of behaviors and cases. This puts forward the need to develop DNS as well as the importance of constructing NN architectures that minimize the dataset dimension to make it feasible (see section 6.3).

After training, the NN model is meant to behave as an exact surrogate of the DNS from which it derives. Then, it becomes possible to obtain a high-fidelity time evolution just starting from a short input sequence or even from an initial frame only, without any explicit solution step. This yields a great advantage as the NN time step τ is generally larger (say at least one or two orders of magnitude) than a typical integration step in DNS. A speed-up of several orders of magnitude can be achieved if replacing explicit time-integration schemes [75, 125, 126], since the NN solution is not subject to the same time-step limitations imposed by stability constraints. As demonstrated in several studies [126, 139], NN surrogates can be faster than semi-implicit integration schemes too, thanks to their simplicity and efficient, highly parallelized algorithms. Typical speed-up of factors of $\approx 100\times$ in 2D [126, 139] have been reported and even stronger accelerations of more than $1000\times$ were achieved in the 3D case [75, 76, 125]. This way, the NN model can be used to produce fast predictions, e.g. for 'real-time' response, or to perform simulations for longer times and larger domains (within the limits of extrapolation) too costly for the DNS. One has however to remember that the acceleration gained in this way is at the price of a more opaque model, in which separating different contributions and error sources may be difficult.

Another important aspect is that the NN surrogate can preserve or regenerate data consistently with inputs, so as to be promptly transferable back into the DNS. This opens the way to possible hybrid schemes [140], using the fast NN for time-jumping between time-consuming stages of a DNS, reverting to it whenever critical events or loss of accuracy occur.

Learning the time evolution of a PDE solution essentially requires a NN to capture both spatial and time correlations as set by the PDE differential operators in both space and time coordinates. Concerning time-handling, literature studies can be divided between those predicting a new evolution stage on the only basis of the current state and those accounting for memory effects over multiple time steps, eventually via a recurrent architecture (see section 2.4).

Following Dingreville *et al* [141], a further categorization can be based on the representation of DNS input data, distinguishing between NN approaches directly processing the DNS data and those extracting latent-space features from them. We will here call the first category 'data-space' methods (generalizing the 'pixel-space' concept discussed in [141]) typically exploiting convolutional, graph or DeepOnet architectures. The second class will instead be referred to as 'latent-space' approaches. If one has to make an analogy, data-space approaches are akin to finite difference/finite volume methods, in which one evolves values at discretization points. Instead, latent-space approaches are affine to (nonlinear and truncated) spectral methods: the state of the system is expanded into a reduced set of coefficients, whose evolution allows for the reconstruction of the dynamics of the system.

In the following, we present the key concepts for each of these categories and inspect the related literature.

3.2.1. Data-space approaches.

Processing DNS data as pixel values in a multi-channel image is maybe the most straightforward data-space approach. In this case, pixels represent the system variables at each evolution step, and the NN is trained to reconstruct such an image sequence corresponding to the temporal evolution. Dealing with images allows one to take full profit from the latest advancements in the field of computer vision, relying on CNN architectures as a basis. The recognition of spatial features is achieved through the fitting of kernel parameters which extract features in the hidden layers.

CNN or U-Net architectures have been successfully employed to predict the time sequence for several problems, such as microstructural evolution in polycrystalline materials [136, 139, 142], spinodal decomposition [143–145], domain formation in ferroelectrics [76, 146] and dendritic growth [147, 148]. In [126], an eidetic 3D LSTM architecture, applying convolutions to both space and time dimensions has also been exploited to tackle several time-dependent problems in 2D, such as wave equations, spinodal decomposition, dendritic growth and grain microstructure evolution. An example of a typical evolution of the latter case is reported in figure 9(a), showing that, while one-to-one accordance with the DNS is lost, typical physical features are still recovered. This is also verified by a quantitative analysis, comparing grain statistics (e.g. grain area and radius).

The relevance of using a recurrent scheme is especially highlighted by Kazemzadeh Farizhandi *et al* in [145] where the use of a ConvLSTM [98] is indicated as the key to training a NN model capable of tracing both the fast phase-separation and the slow coarsening dynamics proper of spinodal decomposition. Moreover, by learning deeper time correlations, a general improvement of time extrapolation can be achieved, as demonstrated by Lanzoni *et al* in [125] where a 3D model trained on relatively short sequences of a few tens of time steps is found to yield consistent predictions on a 3-orders-of-magnitude longer timescale. This is made evident in figure 10(a) where a one-to-one comparison between the ground-truth simulation sequence and the corresponding NN predictions is reported. The match remains compelling well beyond the training time domain even if local faults due to bifurcating events (splittings or coalescence) become evident when exceeding a time that is $100\times$ the training duration. Notably, the NN predicts the correct final equilibrium state with complete phase separation. Moreover, while the one-to-one overlap is not possible at the late times, it is however found that the global dynamical behavior predicted by the NN, defined by the average over multiple NN sequences, remains consistent with the expectations of the physical model. A quantitative check is also provided: on average, the free-energy volumetric density F/V is expected to decay in time as $t^{-1/3}$ [149] in long-time coarsening, due to scaling laws. This is readily verified by averaging the result of 10 independent NN predictions, as reported in figure 10(b). Notice that in this case, a comparison with the corresponding ground truth would be computationally very expensive, if compared to the training set generation time, due to the requirement of very long simulations on large computational domains.

Even if a generic CNN (as any NN) can in principle be made an effective approximator for the solution of arbitrary problems, a large dataset could be needed to let the NN model infer the full problem physics. On the other hand, in most cases, physical constraints and variable relationships to which the solution must conform are known for the specific problem at hand. These can then be encoded as weak inductive biases by conditioning the training to satisfy such requirements. This way, the parameter space is reduced thus favoring a faster convergence of the training process. A simple implementation of this concept can be found in [124] where a loss penalty term is included for the training of a CRNN model for surface diffusion dynamics, to enforce global volume conservation.

A more extended blending of physical constraints into a training loss is exploited by Alhada-Lahbabi *et al* in [76, 146] for the prediction of the polarization and mechanical displacement vector fields and of electrostatic potential during domain evolution in ferroelectric materials. To this goal, a U-Net is trained on a dataset of evolution sequences obtained by the numerical solution of time-dependent Landau–Ginzburg equations via PF. A composite loss function is considered, including both the supervised contribution comparing the field predictions with the ground-truth data with unsupervised terms enforcing the minimization of energy and of the residual from computing the Poisson equation by the predicted electrostatic field. Notably, even in the more challenging 3D case of [76], the prediction error is reported to be below 10% with respect to the DNS, as made evident by the close match of NN vs. PF simulation, although with local failures, reported in figure 9(b). In this case, an impressive speed up factor of $2500\times$ is achieved.

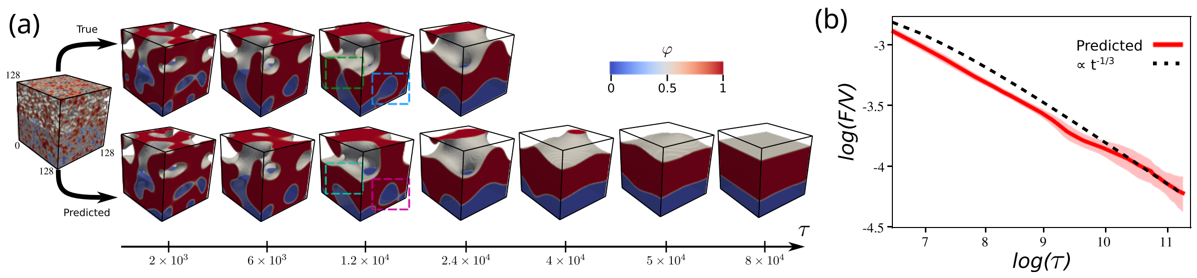

Figure 10. (a) Comparison between the ground truth (top) and the CRNN prediction (bottom) of a 3D spinodal decomposition process. Notice that, similarly to approaches discussed before, local differences, although consistent with the underlying physical process, may arise due to error accumulation. (b) Comparison of the mean free-energy volumetric density F/V decay from NN predictions (red line) and the analytical expected long-time behavior (dashed black line). The shaded area corresponds to one standard deviation. Statistics is collected over 10 independent simulations. Reproduced from [125]. CC BY 4.0.

In coupled problems, an alternative approach to add available physical information is to use DNS for the less computationally expensive contributions, similar in spirit to the approaches described in section 3.1. Then, the NN is responsible for the evolution of the remaining terms, conditioned on the already available DNS. A concrete example may be found in the work by Choi *et al* in [136]. There, the authors use a U-Net architecture for predicting the microstructural evolution of grains in a laser powder bed fusion process. The problem is coupled with that of thermal transport, which is solved by DNS. At every time step, the NN is provided with the current microstructure (represented via PF), together with the current and the next DNS temperature profiles. This approach yields a computational speed-up of $20-1200\times$, compared to full DNS.

An emerging alternative to CNN in data-space methods is the use of GNN architectures. One of the applications in which this class of approaches has seen particular success is the modeling of 3D microstructures as graphs are particularly effective in mimicking complex grain interconnections. Depending on the application, this may be done in several ways.

In [75], Fan *et al* apply a MeshGraphNet approach [150] to the evolution of grains under a simple isotropic approximation. Specifically, in this approach a graph representation of the mesh for the PF description of the microstructure is used, allowing non-uniform discretization. This, in turn, allows for further compressing the computational costs with respect to CNN. Importantly, despite not coupling the architecture with recurrent units, an increase in performances with respect to the convolutional recurrent baseline is reported. Long time sequences up to ten times those used in training are shown, with a high accuracy reproduction of statistical distribution of grain sizes. Additionally, an adaptive time stepping procedure is proposed: several independent NN are trained on DNS data with different time steps between the input and predicted microstructure. At evaluation time, the predictions of models are compared and, if this is below a certain threshold, a switch to the longer-stepping model is performed.

A different way of exploiting the representation capabilities of graphs may be found in [151], with an application to hysteresis curve prediction in Hafnium oxide, a ferroelectric material. There, Alhada-Lahbabi *et al* show that the polycrystalline microstructure can be conveniently embedded into a graph structure which represents the connectivity between grains. Node features, containing the grain volume and orientation, are supplemented with edge information containing the contact angle between them. The GNN is then used to extract from a given morphology the corresponding hysteresis curve, thus bypassing the explicit evolution of the microstructure, yielding an impressive acceleration of ≈ 6 orders of magnitude with respect to the PF solver.

A second possibility relies upon exploiting more geometric representations, as in [138]. There, the problem of epitaxial growth of grains, obtained from multi-PF simulations, is tackled. Instead of leveraging CNNs or graph architectures, grains are described in terms of their dynamical geometric features (e.g. the grain sizes and the positions of their interfaces). While no convolutions are used, a specialized LSTM architecture, accounting for possibly non-local interactions, is also proposed. The reported speedup is of at least a factor $1000\times$ with respect to the DNS simulation, with prediction errors localized at interfaces and being typically in the order of 5%–10%. In figure 9(c) we report a comparison between DNS and NN simulations of growth, together with the misclassification error.

DeepONets represent yet another possible NN architecture choice, shown to provide similar advantages in terms of computational speed-ups.

In the very recent work of [152], Ciesielski *et al* show how a DeepONet architecture may be trained in a data-driven fashion to approximate grain growth dynamics of Uranium crystals as predicted from the Allen–Cahn equation. Importantly, the authors exploit a Convolutional architecture as a preprocessing step in the branch network, achieving a prediction error below 0.5%. Another very interesting feature is that the amount of data required to train the DeepONet model is very small: only 119 time sequences were used, while the typical training set is composed of at least one order of magnitude more examples.

In [153], Huang *et al* show a data-driven application of a DeepONet for the prediction of reactive transport equations in porous media. Also in this case, a CNN is used in the branch network to find a suitable, high-level representation of the porous microstructure. Additionally, a second branch subnetwork is added, containing parameters for the specific case considered (e.g. the diffusion coefficient). The method is shown to provide a computational speed-up of three orders of magnitude with respect to the corresponding FEM solution.

3.2.2. Latent-space approaches.

In order to reduce the complexity of the problem and variability in the dataset, possibly clearing redundant information, input data from DNS are first embedded in a low-dimensional manifold to produce a compact representation of the system state. Indeed, this is once again motivated by the 'manifold hypothesis' [48, 154] (see section 2.1): the set of physical trajectories in data-space is, in reality, concentrated into a smaller-dimensional, 'latent' manifold, thus making it possible to obtain reliable approximations by considering a subset of important variables. Generally, the mapping from data space to latent space is nonlinear. If such mapping is discovered and new descriptors, which better track the dataset variability, are extracted from the dataset, this enables the training of NN models with fewer parameters, i.e. from smaller datasets. Moreover, the NN model may be trained to reconstruct the dynamics of latent-space variables, returning to the real-space representation only for the output, by a suitable 'decoding' procedure. The success of these approaches is critically dependent on the definition of the tracked latent-space features and the method used to extract them. These are typically obtained through PCA or more advanced nonlinear mappings or generated by an AE (see section 2.3.1).

As an example, in the work by Montes De Oca Zapiain in [137], the morphological evolution of spinodal decomposition patterns is statistically characterized via two-point spatial autocorrelations which are then reduced to latent space by PCA. The trajectories in the reduced space formed by the first (10) PC, i.e. the time sequences of the respective scores, supplemented by a list of the scalar input parameters generating each simulation case, namely mean composition and phase mobilities, are then submitted as input for an LSTM NN. Noticeably, the microstructure recovered by the AE may be directly passed to the DNS, allowing for seamless switching between the methods. This is at variance with an alternative phase reconstruction algorithm, discussed in the same study. A direct comparison of the same microstructure for the two approaches is shown in figures 11(a) and (b). The NN prediction error is non-uniform, with the highest values in correspondence with critical regions in the true dynamics. The corresponding error map is shown in figure 11(c).

Given the nonlinear nature of the Cahn–Hilliard equation ruling the spinodal decomposition, as well as of most relevant PDEs, using (linear) PCA may cause a loss of information when considering the whole stages of evolution so a nonlinear embedding could be preferable. The comparative study of [128], exploiting ISOMAP and UMAP in alternative to PCA shows how the simplicity of PCA could still return better results in the model training, at least when the dynamics is in the quasi-linear coarsening regime. A similar conclusion is obtained also in [155] where a combination of auto-encoding and PCA is shown to deliver better performance with respect to a dual auto-encoding strategy. In [128], different RNN architectures are also used for propagating the latent state in time. As figures of merit, the autocorrelation function and an average characteristic feature size error (AFSE) are considered. Additionally, an analysis of the dependence on the number of frames used in training is also inspected. The results, collected in figure 11(b), show that LSTM and GRU provide better results with respect to standard RNN.

In [156], Oommen *et al* show that it is possible to merge latent-space approaches with DeepOnet architectures, once again for the prototypical case of spinodal decomposition. An autoencoder is used to compress the state of the system for each case in the dataset and a DeepONet is used to propagate trajectories in the latent space directly. This approach is shown to be more accurate than previous variants exploiting RNN [137]. Additionally, the authors explicitly show the possibility of data exchange between the surrogate model and a traditional, high-fidelity PF DNS. This is an important feature for ML models, as it allows for finding the best compromise between the acceleration and accuracy of the solution. The same approach is extended in a subsequent work [140], where applications to other microstructure evolution problems, such as dendritic growth and vapor phase deposition, are considered.

3.3. Benchmarks and available datasets

The success of data-driven approaches is possible only if large, validated datasets are available providing clear benchmarks for the development, testing and experimentation of new NN schemes. Many of the works discussed in this section start with the definition of a physical problem of interest and the construction by DNS of a suitable dataset. While for specific problems this may be the only option, generating a new dataset for each work may involve a significant computational surcharge with respect to the NN workflow and frequently requires experience with DNS. Moreover, a quantitative comparison between competitive methods is very difficult if based on different data.

These problems may be mitigated by using some of the already available datasets and benchmarking frameworks [157]. Open data and accessibility policies (e.g. FAIR—Findable, Accessible, Interoperable, and Reusable principles [158]) have a central role to foster the development of reliable data-driven approaches. Many of the publications here cited offer their dataset fully available through public repositories (see e.g. [113, 126, 159]).

Several projects have recently started with the purpose of gathering community-based data into standardized open-access repositories. An example is the 'Mechanical MNIST'(www.nist.gov/mechanical-properties) [160] dataset of mechanical properties, including microstructural data derived from the well-known MNIST digit dataset [63]. A collection of PF models can instead be found in the 'PFHub' at

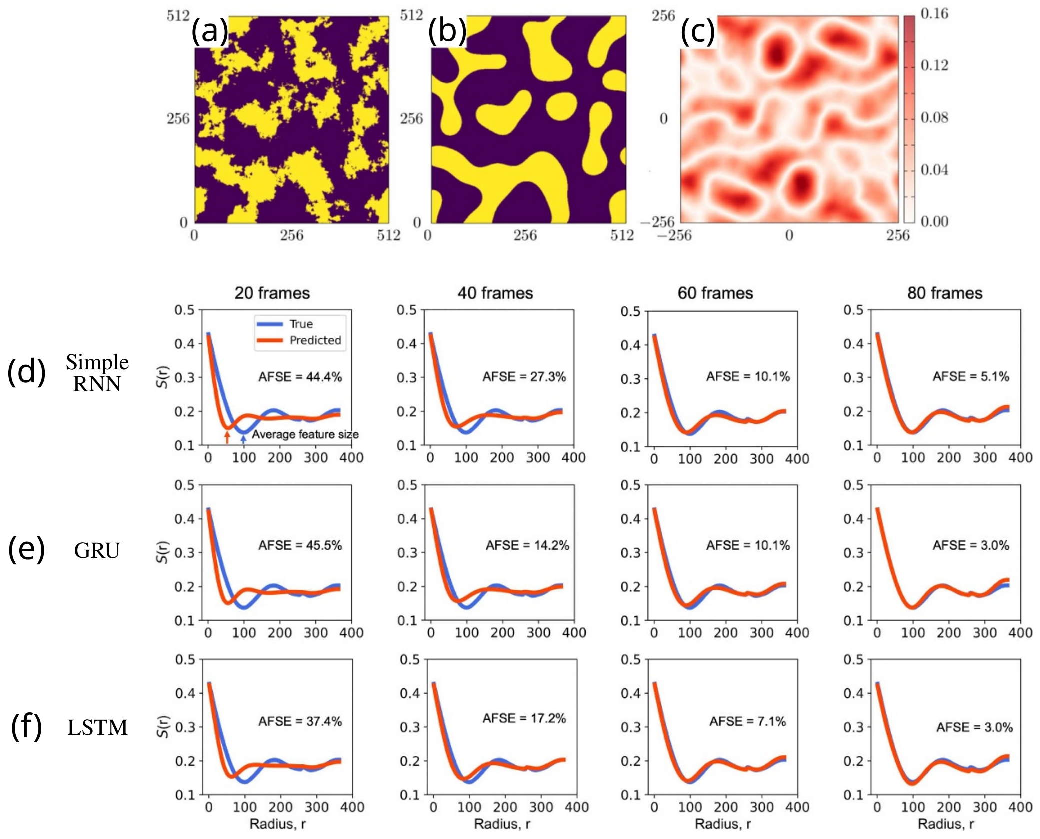

Figure 11. NN prediction of the profile produced by spinodal decomposition following the latent-space approach of [137] (a) recovered by a phase-reconstruction algorithm and (b) obtained by a suitably-trained encoder/decoder structure. Reproduced from [137]. CC BY 4.0. (c) Pointwise prediction error with respect to the DNS evolution at the same time as in panel (b). (d)–(f) Microstructural autocorrelation function $S(r)$ during spinodal decomposition as obtained from a latent-space model (red line) and the corresponding one obtained by DNS (blue line), along with the corresponding absolute relative errors on the average characteristic feature size (AFSE) identified as the minimum of $S(r)$. The plots compare different possible recurrent architectures evolving the latent vector, depending on the number of frames used in training. Reprinted from [128], Copyright (2022), with permission from Elsevier.

https://pages.nist.gov/pfhub/, developed as a benchmark platform to standardize the method [161].

Similar projects have just appeared also for 'general purpose' physics-oriented deep learning applications, targeting standardized benchmarking and performance quantification. An example is the 'Pdebench' [162] dataset at https://github.com/pdebench/PDEBench, proposed in 2022, comprising several examples from different continuum models, such as advection and Burgers' equations or reaction-diffusion systems. Another example is the one on fluid dynamics developed by Hassan *et al* [163] and focusing on liquid-gas phase transitions under various conditions. An in-depth benchmarking framework for testing approaches approximating PF evolutions has been recently developed by Dingreville *et al* [141], also providing extensive datasets in the open PF microstructure database (OPMD) https://microstructures.net/. Similarly, in the work by Rieger *et al* [164] a systematic benchmarking of a U-Net architecture is explored providing full access to their dataset for comparison and re-use.

While, at the time of this writing, no clear benchmark and/or metric has emerged as a standard in the community, this issue will become central in the coming years as more works build along this subject.

4. Physics-driven network architectures

In the previous section, we focused on data-driven NN approaches, in which the necessary step is the construction of a dataset of time sequences, usually by means of more expensive DNS. Fully data-driven methods, in principle, require no previous knowledge of the underlying physical processes, which may be an advantage when dealing with very complex systems, where physical contributions are difficult to model or to identify. On the other hand, recent studies show that the addition of physically-motivated architectures, descriptors or regularizations not only allows for less opaque behavior but is also critical to achieve superior performances. This is particularly critical in view of possible applications to experimental data, where the collection of examples is typically even more expensive and time-consuming.

In this section, we therefore focus on the opposite side of the spectrum of NN models, i.e. on the possibility of training models capable of solving PDEs in the small to no-data regimes. This is the scope of methods where one uses a NN as a solution ansatz for a PDE problem [165–169], providing useful alternatives with respect to classical solvers. One example is dealing with very high dimensional systems [168], where computational costs of PDE discretization are prohibitive. Another possibility is the straightforward assimilation of experimental or scarce data coming from external sources [167]. Physics-encoding architectures are important also because they may be effectively incorporated into data-driven approaches, leading to a ‘best of the two worlds’ scenario.

The idea of using NNs as an ansatz for the solution of differential equations traces back to the pioneering works of Dissanayake [166] and Lagaris [165] in the 90 s. In recent years, however, thanks to the tremendous increase in computational power, this class of approaches is becoming more and more widespread. The most used frameworks in applications to materials science and condensed matter physics are represented by PINNs [167, 170], which will be the main focus of the present section. Notice, however, that alternative schemes are also possible. Notable examples are the Deep Galerkin method [168, 171] (focusing on the weak form of the differential problem) and the Deep Ritz method [169] (focusing on its variational formulation).

4.1. PINNs

PINNs [167, 170] represent an approach to PDEs in which the UA capabilities of NNs and automatic differentiation [106] possibilities are leveraged to solve a differential problem directly. Importantly, at least in principle, there is no need to collect data to train a PINN, as the loss function may be constructed solely based on the knowledge of the underlying differential equation.

To explain this point, we will closely follow the continuous-time version of PINNs proposed in the original work of Raissi *et al* [167]. Consider for example a generic PDE problem for the function u defined by

$$\begin{cases} \mathcal{G}\left[u\right]\left(\vec{x}\right) = 0 & \forall \vec{x} \in \Omega \\ u\left(\vec{x}\right) = u_{\mathrm{D}}\left(\vec{x}\right) & \forall \vec{x} \in \partial\Omega_{\mathrm{D}} \\ \vec{\nabla} u \cdot \hat{n} = u_{\mathrm{N}}\left(\vec{x}\right) & \forall \vec{x} \in \partial\Omega_{\mathrm{N}} \end{cases} \tag{22}$$

where $\mathcal{G}$ is a differential operator (possibly nonlinear), Ω is the domain of u, $\partial\Omega_{\mathrm{D}}$ and $\partial\Omega_{\mathrm{N}}$ are Dirichlet and Neumann boundaries defined by values u_{D} and u_{N} respectively. Invoking the UA capabilities, we can approximate u with a NN $w(\vec{x}|\vec{\vartheta})$ and then construct the loss function as the residual of w with respect to the PDE:

$$\begin{aligned} \mathcal{L}\left(\vec{\vartheta}\right) = & \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{G}\left[w\left(\vec{x}_i|\vec{\vartheta}\right)\right]\right)^2 \\ & + \frac{1}{N_{\mathrm{D}}}\sum_{i=1}^{N_{\mathrm{D}}}\left(w\left(\vec{x}_i|\vec{\vartheta}\right) - u_{\mathrm{D}}\left(\vec{x}_i\right)\right)^2 \\ & + \frac{1}{N_{\mathrm{N}}}\sum_{i=1}^{N_{\mathrm{N}}}\left(w\left(\vec{x}_i|\vec{\vartheta}\right) - u_{\mathrm{N}}\left(\vec{x}_i\right)\right)^2 \end{aligned} \tag{23}$$

where $\vec{x}_i$ are N evaluation points sampled from the domain, N_{D} are sampled on Dirichlet boundaries and N_{N} on Neumann boundaries. We report a sketch representing the role of the NN

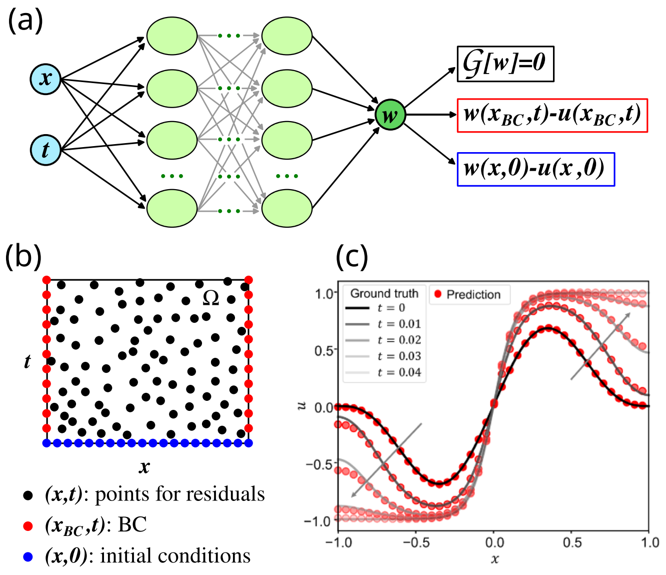

Figure 12. (a) Schematics of a PINN architecture: a NN is used as the ansatz for the solution of the differential equation and the loss function terms are constructed using automatic differentiation without the need of a training set of example. (b) Schematics of the evaluation points for training a PINN. Different colors correspond to different terms in the loss function (black: residual evaluation; red: boundary conditions; blue: initial condition). (c) Comparison of the solution of the Allen–Cahn equation as obtained by PINNs and finite difference. Despite local deviations, the accordance is remarkable. Reproduced from [173]. CC BY 4.0.

ansatz in figure 12(a). A schematics showing the typical distribution of evaluation points in (x,t) space for a 1D, time-dependent problem is instead shown in figure 12(b). Several strategies, beyond uniform random selection, are possible, such as Latin hypercube sampling [167] and active sampling techniques [172]. Additionally, alternative error measures may be used instead of the reported MSE. The critical point of PINNs is that, since w is continuous by construction, the value $\mathcal{G}[w](\vec{x}_i)$ can be obtained exactly and efficiently through automatic differentiation. The price to pay is that training a PINN is more computationally expensive with respect to training a data-driven NN due to the calculation of (possibly high-order) derivatives.

To illustrate the degree of precision achievable with PINN architectures, we report in figure 12(c) a comparison between the solution of the Allen–Cahn equation and the finite difference one, as performed by Li *et al* in [173]. Zero-flux boundary conditions are used in this case. Minor local differences may be observed, mainly close to domain edges. However, the accuracy of the PINN solution is remarkable and the NN architecture can capture the correct phase separation behavior.

Another strong advantage with respect to classical solvers is that, while a finite collection of $\vec{x}_i$ values is required at training time, there is no need for an explicit, fixed mesh and, once a PINN is trained, there is no remaining memory footprint of the evaluation points used.

As previously mentioned, to perform training there is no need for an external dataset of examples: once the PDE problem and the NN architecture have been defined, the loss may be directly calculated. These approaches may therefore be used as alternatives to classical solvers [43], such as FEM, finite difference and finite volume methods. Indeed, several works use PINNs and related ideas to train NN to approximate PDEs in a fully data-free way, e.g. see [173, 174].

As an example, [175] reports the application of a PINN to the evolution of a system undergoing a thermochemical curing procedure, described by a system of coupled nonlinear PDEs, i.e. the heat equation and the time evolution of the material cure degree. This work also shows that this framework retains the flexibility of NN approaches: the authors adapt the original PINN architecture to handle discontinuous domains in the case of bi-material systems, where material properties abruptly change across the interface. This may be done straightforwardly by splitting the network into two subnets, each of which is multiplied by a corresponding Heaviside step function. The relative absolute error in the temperature prediction with respect to the expensive FEM solver is less than 1%, showing the high effectiveness of this approach.

Despite their potential, care is needed when applying PINNs to PDEs which exhibit complex behavior. For instance, in [172], Wight and Zhao performed an in-depth, comprehensive analysis of numerous training strategies for PINNs applied to Cahn–Hilliard and Allen–Cahn flows showing how the choice of sampling techniques for evaluation points and loss term weights, together with techniques imported from data-driven contexts such as mini-batches, is crucial for obtaining a successful solution of these PDE problems, even in the simple 1D case. Indeed, the authors explicitly show that without these modifications, the PINN approach may fail dramatically for both models. In the same work, the use of adaptive sampling in time is also discussed, suggesting two strategies to limit error propagation across consecutive time steps. The first is to progressively expand the time window learned by the PINN (an approach resembling curriculum learning [96]). The second one splits the full evolution into time intervals, each of which is learned by a dedicated NN.

A related approach to PINNs, which also uses the possibility of representing complex microstructures as graphs, has been proposed in [74], called Physics-embedded graph network (PEGN). The material domain is discretized using a graph structure, instead of a standard meshing procedure (see figure 13(a)). Then, the evolution of the system is based on the flow of properties (temperature field, grain orientation, liquid-solid fraction) following free-energy gradient flow. Moreover, the free-energy function $\mathcal{F}$ is conveniently defined directly on the graph and backpropagation is used to calculate gradients efficiently and simulate the dynamics of the system. More specifically, $\mathcal{F}$ is computed using the message-passing mechanism on the graph [67]: each grain (node in the graph) communicates with neighbor ones, causing the update of node features. A scheme showing the described workflow is shown in figure 13(b). While this method may suffer from resolution issues, it is capable of tackling very large systems at a fraction of the memory requirement, yielding an acceleration of at least $\approx 50\times$ with respect to explicit PF modeling.

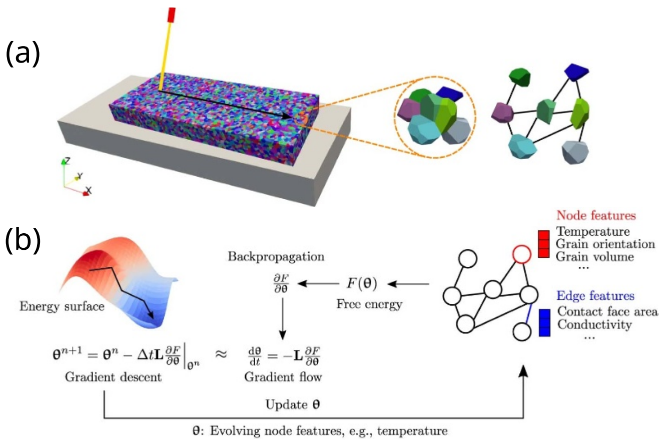
(a)
(b)
Energy surface
Backpropagation
$\frac{\partial F}{\partial \theta}$
$F(\theta)$
Free energy
Node features
Temperature
Grain orientation
Grain volume
...
Edge features
Contact face area
Conductivity
...
$\theta^{n+1} = \theta^n - \Delta t \mathbf{L} \frac{\partial F}{\partial \theta}\Big|_{\theta^n}$
Gradient descent
$\approx$
$\frac{d\theta}{dt} = -\mathbf{L}\frac{\partial F}{\partial \theta}$
Gradient flow
Update θ
θ: Evolving node features, e.g., temperature

Figure 13. NN physics-driven approaches. (a) Representation of grain microstructures as a graph in Physics-embedded graph network (PEGN). Each grain in the computational domain is represented as a node in an undirected graph. (b) Schematics of the PEGN procedure: physical properties are defined on nodes/edges. A free-energy function is defined in terms of such properties and the system evolution is based on gradient flow, efficiently obtained via backpropagation. Reproduced from [74]. CC BY 4.0.

Symmetrically to data-driven approaches, in recent years, multiple instances of 'hybridization' have been suggested and, often, the loss function is augmented with a dataset-based part. From the moment of their proposal [167], PINNs have been formulated with the possibility of integrating PDE information with examples from a dataset. This may be performed straightforwardly by adding to the loss function as defined in equation (23) an additional term in the form of equation (11). In this form, PINNs can be thought of as a strongly regularized NN approach that may be applied in data-scarce contexts. The risk of overfitting is greatly reduced as the NN has governing equation guidance in spatio-temporal regions lacking experimental values. Indeed, several applications in fluid dynamics are currently exploiting this possibility for sparse data assimilation [45, 46], while applications to solids are lagging behind.

We find it instructive to discuss the example of [176] from a methodological point of view. The work shows that it is possible to reconstruct the flow, temperature and pressure field of air on top of a coffee cup, using both a PDE-residual loss, based on the coupled Navier–Stokes and heat transport problem, and a data loss term, based on experimental temperature measurements. Additionally, boundary conditions for the problem are not specified, resulting in an ill-posed setting which could not have been tackled using traditional solver schemes. The PINN approach is in this case capable of obtaining realistic velocity fields, both in terms of magnitude and directionality, as compared with an independent experimental setup. Moreover, as NNs provide continuous functions both in space and in time, it is possible to interpolate between collected sparse data, allowing the inspection of potentially important but unobserved details.

4.2. PINNs and deep neural operators

An emerging approach is to rely on DeepONets [87] as an alternative to 'coordinate' networks (i.e. NN taking spatio-temporal coordinates as input) in the context of PINNs for condensed matter problems. Indeed, DeepONets have proven to be very effective in the approximation of general parametric PDEs [177, 178] using the PINNs scheme: while the original formulation in [87] was used in a data-driven fashion, i.e. exploiting a dataset of input-output examples, it is still possible to take profit of automatic differentiation and use the PDE of interest as a loss function also with a DeepONet architecture.

An additional advantage of DeepONets over the standard PINN approach lies in the possibility of making the learned NN independent of initial and boundary conditions. While coordinate networks need to be retrained for every set of boundaries, as required from the loss function in equation (23), one may use DeepONets to directly learn the operator mapping the initial/boundary condition to the corresponding solution, dramatically increasing the transferability of the approach.

In [179], Koric and Abueidda show that DeepONets can effectively produce the solution of the heat diffusion equation with various source terms. In the same work, a comparison between the performances of data-driven and PINN approaches is also shown. Interestingly, findings suggest that, at least for the specific problem considered, data-driven approaches may achieve a marginally lower error rate, although it has to be considered that, in contrast with 'pure' Physics-Informed approaches, there is a dataset construction computational overhead. This is a key consideration, as for complex PDEs the construction of a dataset may indeed represent the computational bottleneck.

In recent works, it is shown that a DeepONet architecture may be successfully employed in the solution of prototypical PF models such as Cahn–Hilliard and Allen–Cahn equations [173, 180] as well as of microstructure evolution dynamics, such as crack propagation [181].

In [180] the free energy of the system is combined with a function-space distance to learn Allen–Cahn and Cahn–Hilliard dynamics. More specifically, the DeepONet is used as a time stepper, providing the next state of the system given the current one. To this aim, the authors exploit both a data-driven loss term, based on numerical simulation of the evolution of random Gaussian process, and a physics-based loss term. The latter is based on a minimizing movement scheme: instead of training the NN to approximate the residual of the result of an explicit integration scheme, the joint minimization of the free energy and a suitable distance metric between functions is used, allowing for much larger time steps. These are shown to be equivalent to those available in implicit integration schemes, increasing the efficiency of the approach. Computational speed-ups are in the order of 10 $\times$. A tutorial for the approach, guiding the reader to implementing the method in 1D may be found at [173]. Moreover, a comparison between coordinate PINN approaches and using DeepONets as time steppers is provided in the same work.

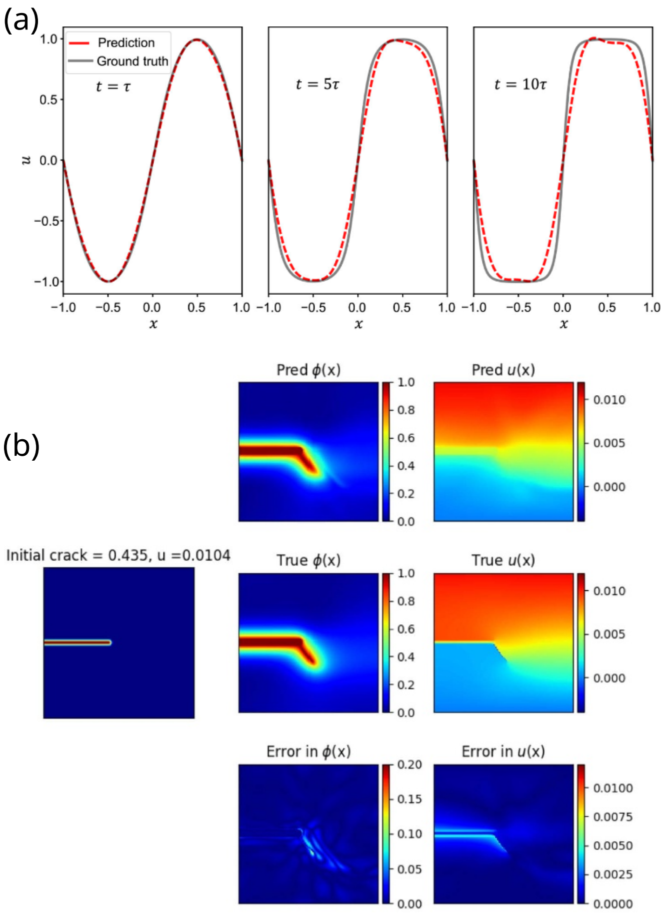

Figure 14. (a) Comparison between the ground truth and the prediction of the 1D Allen–Cahn equation using DeepONets. Phase separation is well captured and no need of re-training the architecture for new initial conditions is required. Reproduced from [173]. CC BY 4.0. (b) Example of a crack propagation prediction based on a PINN approach leveraging DeepONet architecture. Almost one-to-one accordance may be seen in this case for the crack configuration (ϕ values), while a higher average error may be observed for displacement fields $\vec{u}$. Reprinted from [181], Copyright (2022), with permission from Elsevier.

An example of Allen–Cahn dynamics obtained with the latter method is reported in figure 14(a).

In [181], a variant of DeepONet (Variational DeepONet or V-DeepONet) is trained using both a dataset obtained by high accuracy DNS (i.e. a data-driven loss contribution) and a term coming from the variational formulation of the problem (i.e. a physics-based loss contribution). In this respect, this is similar to what was already described in the previous section about data assimilation, with numerical solutions taking the role of experimental data. The task of the NN is to provide the PF and the displacement field representing the most probable crack propagation path, both as a function of the initial crack location and applied displacement. The V-DeepONet manages to provide the solution to the problem to a very high degree of accuracy, especially in terms of the PF representing the crack configuration, as exemplified in figure 14(b). Simulation times are compressed by a factor of $\approx 1.5\times$. More optimized architectures are however possible promising a further acceleration for this class of approaches.

5. Other approaches and architectures

In this section we briefly review some novel methods and approaches that are not yet widely deployed for the continuum scale modeling but are expected to raise in interest in the coming years, eventually opening the way to new research areas.

5.1. Attention and transformers

'Transformers' have become very popular since their introduction in 2017 [182–184], replacing RNNs in the natural language processing community, and, ultimately, leading to the recent impressive results in LLMs [185, 186]. Generalizations and applications to image data has also been developed [187–189] thus making transformers a promising candidate also to tackle spatio-temporal evolutions.

While at the time of writing of this Review, only a few applications have been reported in the field of continuum, condensed matter systems, e.g. addressing solidification of polycrystals [138, 190] and stress prediction in fracture [191], a fast development of this research area is expected in the coming years. For this reason, we here briefly discuss its underlying idea and possible exploitation in the context of continuum modeling. Further details are left to the dedicated literature [48, 183].

The core of a transformer consists in using the 'attention' mechanism [192] to infer long-range space and time correlations. Attention is an approach to extract representations from data initially proposed as an alternative to recurrent schemes for handling sequential data [48, 182, 192]. Formally, for each element in a sequence, the attention a is obtained as the weighted sum:

$$a\left(\vec{q},\left\{\vec{k}_i\right\},\left\{\vec{v}_i\right\}\right)=\sum_{i=1}^{m}\alpha\left(\vec{q},\vec{k}_i\right)\vec{v}_i \qquad (24)$$

where $\vec{q}$, $\vec{k}_i$ and $\vec{v}_i$ are called query, key and value, respectively. $\vec{q}$ is calculated from the input element of the sequence, while a $(\vec{k}_i,\vec{v}_i)$ pair is obtained for all the m elements in the sequence. $\vec{q}$, $\vec{k}$, and $\vec{v}$ values are calculated via suitable learnable (and possibly nonlinear) projections [182, 192] (e.g. fully-connected subnetworks). The function α is chosen to be positive and normalized to one [48, 182].

One of the key advantages of the attention mechanism compared to the RNN architecture is that long-range dependencies are not mediated by the hidden state $\vec{H}_t$, but are directly encapsulated by large values of α. This limits the vanishing gradient problem (see section 2.4) and explains transformers' capabilities in capturing non-local dependencies. Indeed, this is explicitly discussed in [191], where the attention layer is exploited to capture the global features necessary to correctly predict crack propagation patterns. Superior performances of architectures combining attention and CNNs has already been proven in the broader deep learning community for tasks where non-local behaviors are present [193, 194]. Moreover, it is possible to completely get rid of convolutions and opt for vision transformers [187]. These early developments offer a clear proof of concept of the potential applications to the simulation of condensed matter systems.

Transformers also offer several advantages from a computational perspective. Indeed, a can be calculated for all elements concurrently via matrix multiplication, removing the need for sequential generation of predictions at training time, thus leveraging full parallelization. Additionally, these architectures exhibit favorable scaling properties with training data and number of parameters [195]. However, the construction of very large datasets might still pose a severe bottleneck limiting applications. Another issue is that attention requires the retention of the whole previous sequence to predict future states [196]: in contrast with RNNs, the memory requirements and computational costs increase with the length of the generated sequence, albeit this is mitigated by parallelization and may be partially addressed by specific strategies [197, 198].

Finally, it must be noted that attention is by construction invariant with respect to the order of elements. Specialized modifications, such as 'positional encoding' [182, 199], must be implemented if sequence order is important, as natural in time-evolution of physical systems.

5.2. Reinforcement learning

Reinforcement learning is another successful paradigm in ML [200], where an autonomous agent learns how to take actions in a given dynamic environment in order to perform an assigned task as effectively as possible. In general, such 'most effective' action is formalized in terms of a reward function which has to be maximized. NNs, thanks to their expressive power, are one of the possible tools to define the agent behavior. As a prototypical example, deep Q-learning, introduced in [201], uses a NN to approximate the so-called 'value function', i.e. the function which returns the expected reward given the current state of the system. This and more refined techniques may be used in the context of materials science and condensed matter research, opening the way toward disruptive applications, especially in experimental workflows, where the automation of specific tasks could ensure reproducibility and higher-quality results.

As a proof of concept, in [202], Yang and Demkowicz show that a NN may be trained, based on Allen–Cahn PF simulations, to find the optimal control parameters (temperature and bias field) to obtain a target microstructure.

A more complete example of the possibility of microstructure control via NN in ferroelectric materials is given in the very recent work by Alhada-Lahbabi *et al* in [203]. There, the NN model presented in [76] (and here discussed in section 3.2.1) is used to surrogate the dynamics of the system. A second NN model is then trained on surrogate dynamics data to learn the optimal tip trajectory in order to reach a target microstructure, showing the potential for the use of these approaches in nanoelectronic applications.

The possibility of transferring reinforcement learning concepts to experimental setups, still in the context of ferroelectric materials, is discussed in [204]. There, Smith *et al* trained a NN model to predict the dynamics of ferroelectric domain walls from piezo-response force microscopy images. This model is then used to train a deep learning agent capable of automatically determining the voltage of pulses required to obtain a target wall microstructure. While in this study the reinforcement learning approach is not deployed into the experimental setup directly, it is a clear demonstration of the method potentialities.

Another example, which, however, involves an atomistic description of the system, may be found in [205, 206]. There, the authors demonstrate that a reinforcement learning approach can be exploited to obtain the target shape of a nanocluster by changing the temperature of the system. This result testifies the strength of these approaches in handling the dynamics of systems evolving also in the presence of random thermal fluctuations.

5.3. Ensemble methods and prediction uncertainty estimation

One of the main limitations of NN approaches is that it may be difficult to identify when the trained model is in an interpolation or extrapolation regime. Indeed, the development of detection tools is essential to avoid possible catastrophic failures in model predictions, especially in view of possible real-time applications in experimental settings. Additionally, a prediction uncertainty estimation is particularly useful in hybrid schemes such as the one proposed in [140], in which NN outputs may be seamlessly exchanged with the DNS. A last advantage is that this may enable active learning strategies [207], in which new elements are added to the training set and the model is refined based on inaccurate predictions.

A possibility in this sense, which is applicable in a straightforward manner to any training workflow without the need for extensive overhauls, is the use of ensemble methods [208–210]. The core idea is that, given a dataset, multiple NNs are trained. Due to the non-convexity of the loss function and the use of stochastic optimization tools, each trained model will converge to a different set of parameters [6]. The advantage of this procedure is twofold. First, individual predictions may be aggregated (e.g. by taking a simple average). Second, the spread of outputs provides an estimation of prediction uncertainty. The reason is that close to dataset configurations (i.e. in interpolation), all models are constrained to behave similarly, while in extrapolation this requirement is lifted. This approach has been used in some active training workflows in the context of interatomic potentials [211], and recently applied to continuum models.

The [124] provides a proof of concept of ensemble methods for CRNNs predicting the morphological evolution of material domains under surface diffusion, as from PF simulations. No active learning strategies have been implemented there, but the prediction uncertainty is shown to correlate well with prediction errors. In figure 15(a), the average prediction of the ensemble on an extrapolation case is shown, together with the corresponding standard deviation. High uncertainty regions are marked with correspondingly high predicted standard deviation. Noticeably, these are in correspondence with areas where material domain splitting is possible, which indeed correspond to the most critical regions in the system dynamics.

In the work by Jiang *et al* in [212], instead, a similar strategy is applied to PINNs for heat transport problems. More in detail, the NN is used to identify the unknown source/boundary term in the stationary state of the heat equation. The combined prediction of the ensemble is shown to be more accurate than individual ones. Concurrently, the prediction uncertainty is used for an active sampling procedure of new evaluation points. In figure 15(b), confidence intervals extracted from the ensemble method are reported with the blue shaded area. In steps 1-8, evaluation points are progressively added based on high-variance regions in an active sampling strategy. In step-8, the approach has converged to the correct solution.

5.4. Learning governing equations with PINNs from spatiotemporal data

An additional advantage of PINNs, not mentioned in section 4.1, stems from the possibility of using this framework not only for PDE approximation but also in the discovering of governing equations. In this sense, they also offer the possibility of solving inverse problems from observations [213, 214].

To illustrate how this problem may be tackled, we follow the original proposal in [213]. The starting point is analogous to the one discussed in the previous section, with the difference that the differential operator $\mathcal{G}$ now contains N unknown parameters $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_N\}$. Approximating the solution u with a NN, a PINN is therefore defined. The main difference with respect to the minimization of equation (23), is that during training the parameters λ (initialized at random) are now learned as well, leveraging once again automatic differentiation. At convergence, the PINN both approximates the solution of the PDE and has identified the most probable set of parameters.

In this respect, some applications are present, although in static contexts. In [215], the authors apply PINNs to tackle consistent field theory calculations of the monomer distribution in copolymers. In particular, deep learning is used to both approximate the partition function and the chemical potential terms required to estimate the equilibrium distribution of monomers in 1D. Additionally, it is shown that PINNs may be successfully used for inverse problems too, i.e. to infer self-consistent field parameters from a known monomer distribution, recovering qualitative accordance.

As a second example, in [216], Zhang *et al* apply PINNs to the identification of mechanical parameters in non-homogeneous materials. Specifically, the task considered is the identification of the spatial distribution of shear moduli based on simulated displacement fields in a quasi-static regime. The NN shows very high prediction capabilities, with relative errors below 1%, promising interesting applications in experimental elasticity imaging.

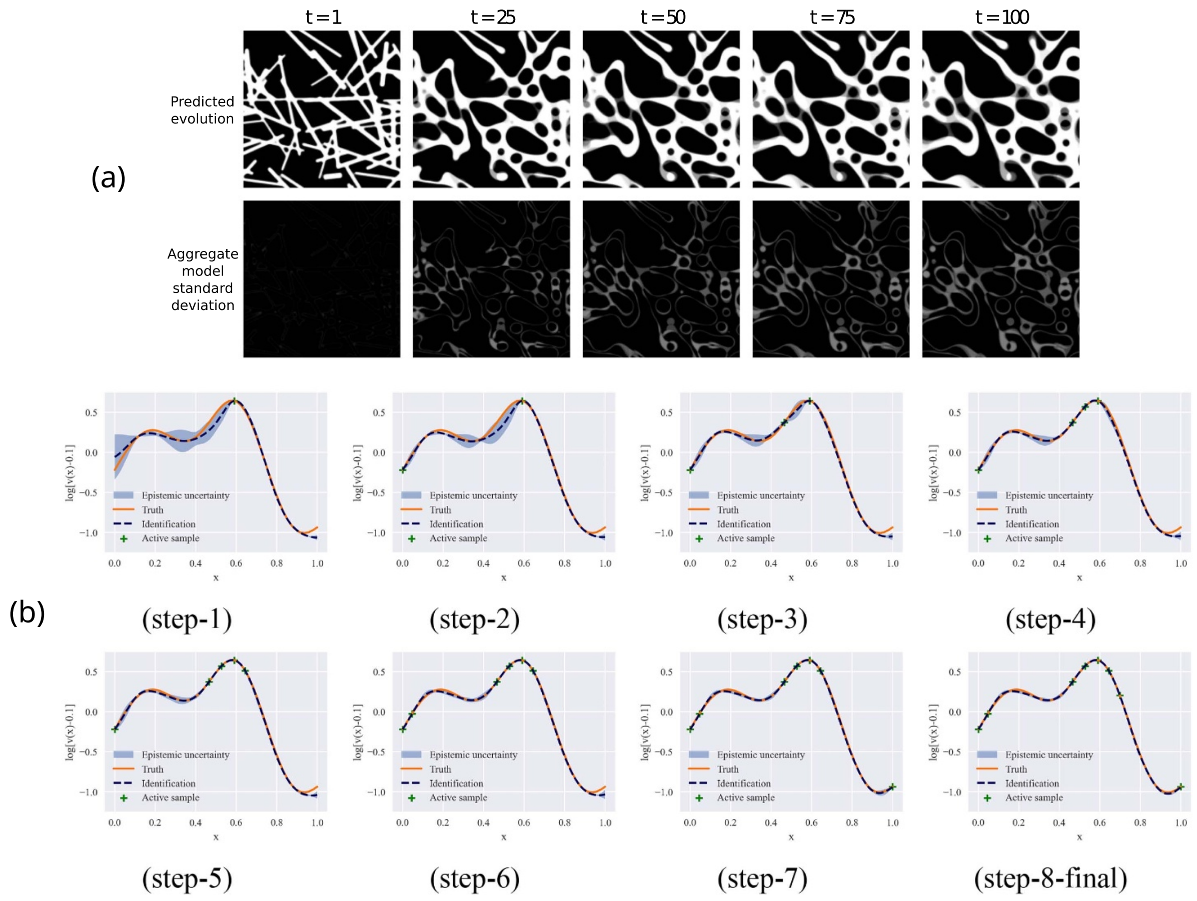

Figure 15. Prediction uncertainty obtained from two different ensemble methods. In panel (a), the aggregate model standard deviation highlights regions in which predictions are diverse. Reprinted figure with permission from [124], Copyright (2022) by the American Physical Society. In panel (b), the ensemble method is applied to a PINN predicting the source term for the heat equation. The shaded area corresponds to the uncertainty extracted from the ensemble. Additionally, high uncertainty points are subsequently added to the evaluation points (green crosses), actively sampling more important regions in the domain. Reprinted from [212], Copyright (2023), with permission from Elsevier.

To our knowledge, however, most of the applications to time-dependent problems come once again from the fluid dynamics community. For instance, in [217], the authors show that it is possible to train a PINN to infer unknown material properties, together with the velocity field. Additional applications and a more in-depth discussion may be found in the review at [45, 46] and references therein. These promising results suggest that the application to inverse problems may provide a useful analysis technique also for experimental condensed matter and materials science.

5.5. *Domain decomposition and equation coarse graining*

Other possibilities of leveraging ML stem from more subtle integration with classical approaches.

One of these is the possibility of exploiting deep learning approaches to make more efficient domain decomposition when using DNS. For example, the classification capabilities of NN may be used to select which edges in discretization meshes require the solution of an eigenvalue problem. A review of some of these approaches may be found in [218] and references therein.

An equally interesting idea is based on the possibility of learning optimized coarse-grained versions of differential problems in order to reduce the fine mesh requirements often present in the numerical solution of PDEs. In [219], an application to 1D problems is shown. A NN is used to find equation-specific coefficients which define a new, effective discretization scheme. The compression rate in mesh requirements is up to a factor 8 $\times$.

6. Challenges, perspectives and open questions

In this section, we discuss the actual state and potential of NN approaches for applications in continuum models for condensed matter physics and materials science. Their strengths and weaknesses are critically considered so as to define their actual perimeter of use and the directions of improvement. Additionally, we discuss briefly about possible future trends and open problems.

6.1. Stability and long time evolution

One of the most critical aspects of NN approaches at the moment is the possibility of obtaining reliable evolution for sequences which are much longer than those provided in training. Despite the possible tremendous computational speed-ups, as stated in section 3, the typical sequence reported in published work is $\approx 10\times$ the length of sequences in training. While this may be enough for smaller-scale problems in dissipative dynamics, it may be still a critical problem when generalization to larger scales is considered, as the time required to reach the stationary state may get considerably longer. This is a particularly critical issue, as most ML models may dramatically fail when reaching their extrapolation limit.

Strategies for achieving such stable surrogate models are still an open question. Based on the current state of published work, it seems that recurrent strategies may help in this direction. A definitive answer, however, is not currently available.

Indeed, very few quantitative and comprehensive studies comparing the performances of different architectures are reported, especially if compared with the multitude of the proposed approaches. Some exceptions are present, such as [141], comparing different approaches for PF simulations of spinodal decomposition. In general, performing such systematic analyses is not trivial, since NN architectures come in several different flavors and variants and it is not always clear how a fair comparison between two competing architectures should be performed. Resorting to standardized benchmarks and generally available datasets is then crucial to develop reliable frameworks (see section 3.3).

A trend that seems to be more consistent is that the introduction of physics-inspired terms and architectures helps with the stability and overall results of the NN predictions. As a concrete example, the introduction of a specialized output layer mimicking the underlying diffusive dynamics in [125] allowed for a reduction in the number of parameters of one order of magnitude with respect to the previous version of the model proposed in [124]. At the same time, the stability of the predictions has increased. Similarly, the addition of physics-based loss terms seems to be particularly effective. Of course, this effect is paramount in PINN approaches, where strong physics-based regularization alone may be sufficient to allow time integration. A stronger integration between data-driven and physics-driven approaches may therefore be expected to provide key contributions in this aspect.

6.2. Global/local schemes

Another critical issue which is currently limiting the generalization possibilities of NN approaches stems from the difficulties in integrating global and local properties in the same architecture. For example, in problems involving elastic contributions, one often assumes instantaneous relaxation to the mechanical equilibrium. In this case, the deformation state of the solid is a global property, i.e. it cannot be deduced only based on local microstructural information. Similar considerations may apply to electrostatic interactions and, in general, to integrodifferential problems.

A possible solution to this problem is to resort to equivalent, non-local NN architectures. This may not be straightforward, however. For example, CNNs encode two main characteristics, i.e. locality, and translational equivariance, from which generalization to arbitrary domain size follows. Removing locality, e.g. by introducing fully connected layers, may easily break these properties, introducing possible spurious effects. A promising alternative is the transformer architectures [182] (see also section 5.1), exploiting the attention mechanism. As already mentioned, it permits the modeling of global dependencies while still allowing for the processing of sequences of arbitrary length. Indeed, in [138] attention mechanism is used to account for non-local grain interactions in epitaxial growth simulations. Similarly, Chen *et al* [191] have shown that self-attention layers help NN to capture global properties in the prediction of stress fields in fractured materials.

6.3. Strategies to reduce dataset size requirements

By definition, data-driven approaches are greedy for training set examples. For instance, the typical size of a dataset is in the order of several hundred to a few thousands sequences, see e.g. [75, 76, 125, 126, 137]. It is safe to say that, based on the trends in the broader ML community, producing a larger amount of data and increasing the number of parameters in NNs will automatically produce more accurate models. As a consequence, the introduction of more efficient DNS for physical sciences will also automatically benefit the development of deep learning approaches.

On the other hand, the collection of extensive datasets is still one of the main challenges. Finding effective strategies to reduce the number of required examples is therefore of utmost importance. If the problem being addressed is a standard one, or if the goal is just the development/testing of the NN approach itself, the availability of standardized, public datasets may alleviate the requirement of generating DNS solutions (see also section 3.3)

An alternative possible strategy to mitigate data availability problems is ‘transfer learning’. The idea is that a NN model is first trained on cheaper simulations of a simpler but related problem, where obtaining abundant data is possible. Then fine tuning is done on the reduced dataset for the real task, possibly freezing part of the parameters. The idea is that starting from a pre-trained parameter configuration would yield convergence to a better NN.

A second possibility comes from ‘multi-fidelity’ schemes. In this latter case, the relative importance of data from either source (expensive or cheap) can be controlled by splitting the total training loss into separate contributions, with a relative weight w: $\mathcal{L} = \mathcal{L}_{\text{cheap}} + w\mathcal{L}_{\text{exp}}$. Attributing a higher weight to expensive data would push the training to be consistent with them, which would possibly improve the model prediction performances in the proximity of those points. Other, more refined schemes are also possible [220].

An application showcasing this possibility may be found in [129], authors use a multi-fidelity approach to learn the free energy of a precipitate based on FEM calculations. The low-fidelity model, in this case, is a second NN. Once trained, the model is used to find minimum energy configurations for a given inclusion volume, leveraging the lower computational costs with respect to a full FEM study.

6.4. Bridging continuum and atomistic scales

A promising direction in which deep learning approaches may lead to significant advancement for continuum models is the possibility of using NN to parametrize functions and terms coming from atomistic details, enabling the construction of a true multi-scale paradigm. In this case, the focus is not necessarily the acceleration of numerical schemes, but the injection of multiscale information, a notoriously complex task.

In [221], Teichert *et al* show that a suitable NN architecture may be used to parametrize the free-energy density from first principle calculations and then use it to integrate PF models. The same approach is proven to be effective also in a large-scale study on electrochemical Li_xCoO_2 systems [222]. There, density functional theory calculations were used to estimate atomistic energies. Free-energy terms are then fitted using deep learning techniques, allowing one to perform simulations of electrochemical cycles.

An application similar in spirit, but focusing on mechanical properties, may be found in [223]. The authors train a model capable of detecting mechanical failure and the stress response from molecular dynamics and then employ it to run continuum simulations, efficiently incorporating atomistic information on dislocations and crack formation and propagation.

Bypassing the time integration of continuum models altogether and training data-driven models on atomistic data directly is another fascinating perspective. For instance, in [224], a VAE is used to find a more compact (ten-dimensional) representation of molecular dynamics trajectories for polycrystalline Ni simulations. The microstructure is then evolved in latent space using a suitably trained LSTM network, similar in spirit to the latent-space approaches discussed in section 3.2.2. The speed-up provided is dramatic, albeit with limited time extrapolation capabilities.

6.5. Stochastic dynamics

Another critical issue, which we have so far not discussed, is that continuum models described by PDE are inherently deterministic (although possibly chaotic). Stochastic fluctuations, however, play an important role in many physical systems. The learning task, in this case, is much more complex, as the trajectories of the system are stochastic and the NN approach should learn the distribution of possible future states. Advancing the system in time is equivalent to sampling from such distribution. NNs, in this sense, offer several approaches in the form of generative models. The three most used possibilities are currently variational autoencoders (VAEs) [225], generative adversarial networks (GANs) [154] and diffusion models [226].

In particular, GANs have proven quite successful in learning the dynamics of low-dimensional systems. Some examples have recently appeared in literature: in [227] a prototypical system in 1D, based on kinetic Monte Carlo simulations, have been considered, [228] tackles more complex, three-dimensional dynamics and [229] introduces a proper regularization scheme for RNN applications in stochastic time series generation.

Another example involving more realistic physical systems, although with an atomistic description, is in [230], where an application to the dynamics of small molecules in solution is discussed. The NN approach, based on a diffusion model, is trained on a dataset generated from molecular dynamics simulations. Convergence to statistical properties is shown, providing at the same time an impressive speed-up of $80-180\times$.

A modified version of the original scheme of PINNs has also been proposed to tackle stochastic systems [231]. The approach is based on GANs, and inserts physical information into the NN architectures as a stochastic differential equation.

6.6. Training on experimental data

Another potentially groundbreaking application of NN for modeling condensed matter and materials science systems is related to the possibility of exporting the approaches discussed in this Review to experimental data directly. This is particularly appealing for complex systems, where modeling attempts are difficult and/or the identification of all the driving forces is still lacking. Indeed, some applications regarding real-time and automatic control of growth processes in semiconductor manufacturing have already been proposed [232, 233]. On the other hand, it has to be remembered that, without physics-guided procedures, there is the risk that NN methods may become complete black boxes, providing accurate predictions without yielding any understanding of the underlying processes.

Deep learning methods are agnostic with respect to the origin of the data, hence they can be applied to simulations and experiments with little to no modification, depending on the specific approach. In some sense, applying NN approaches to ‘synthetic’ data coming from DNS may be considered an essential boot camp to understand which architectures and tools are the most effective, in view for experimental applications. On the other hand, however, one should remember that collecting extensive datasets of laboratory observation may be particularly expensive and/or time-consuming.

Due to the limitations mentioned above, the works which actually exploit this possibility are very few. A notable example is in [234], where a specialized AE architecture is used to reconstruct the dynamics of Ge pore reorganization in a latent-space model.

Another promising direction, which has already been mentioned in section 4.1 is related to data assimilation in PINNs. This may represent a very interesting opportunity, especially when dealing with experimental techniques which provide only partial information (e.g. due to the impossibility of non-destructively inspecting the bulk of a solid) on the system under study.

As many problems of learning from experiments may be traced back to the requirement of obtaining large datasets, possible solutions are similar to those already discussed in section 6.3, i.e. transfer learning, the use of multi-fidelity schemes and data augmentation. For instance, multi-fidelity approaches involving experimental data, may be found in [235, 236], where the density functional theory band gap estimations are improved via ML approaches.

Another scheme potentially bypassing such dataset dimension issues is based on the possibility of training on synthetic data which are suitably modified to resemble experimental ones via data augmentation (see section 2.5.1). This latter techniques have been successfully applied in [237]. There, suitable noise and modifiers were added to PF simulations so to replicate typical experimental transmission electron microscopy images. A model is then trained to provide image segmentation of nanostructures. Once trained, the NN provides high accuracy (>99%), as tested on real data.

Data availability statement

No new data were created or analysed in this study.

Acknowledgment

F M and R B acknowledge financial support from ICSC-Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing, funded by European Union-NextGenerationEU.

ORCID iDs

Daniele Lanzoni 0000-0002-1557-6411
Francesco Montalenti 0000-0001-7854-8269
Roberto Bergamaschini 0000-0002-3686-2273

References

[1] Jordan M I and Mitchell T M 2015 *Science* **349** 255–60
[2] Dimiduk D M, Holm E A and Niezgoda S R 2018 *Integr. Mater. Manuf. Innov.* **7** 157–72
[3] Doya K, Ema A, Kitano H, Sakagami M and Russell S 2022 *Neural Netw.* **152** 542–54
[4] Wang H *et al* 2023 *Nature* **620** 47–60
[5] Zheng Y, Xu Z and Xiao A 2023 *Artif. Intell. Rev.* **56** 9497–539
[6] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (MIT Press)
[7] Carleo G, Cirac I, Cranmer K, Daudet L, Schuld M, Tishby N, Vogt-Maranto L and Zdeborová L 2019 *Rev. Mod. Phys.* **91** 045002
[8] Mehta P, Bukov M, Wang C-H, Day A G, Richardson C, Fisher C K and Schwab D J 2019 *Phys. Rep.* **810** 1–124
[9] Rosenblatt F 1958 *Psychol. Rev.* **65** 386
[10] Clark J W and Gazula S 1991 Artificial neural networks that learn many-body physics *Condensed Matter Theories* ed S Fantoni and S Rosati (Springer) pp 1–24
[11] Yagawa G and Okuda H 1996 *Arch. Comput. Methods Eng.* **3** 435–512
[12] Bhadeshia H K D H 1999 *ISIJ Int.* **39** 966–79
[13] Morgan D and Jacobs R 2020 *Annu. Rev. Mater. Res.* **50** 71–103
[14] Wiewel S, Becher M and Thuerey N 2019 *Comput. Graph. Forum* **38** 71–82
[15] Kim B, Azevedo V C, Thuerey N, Kim T, Gross M and Solenthaler B 2019 *Comput. Graph. Forum* **38** 59–70
[16] Kochkov D, Smith J A, Alieva A, Wang Q, Brenner M P and Hoyer S 2021 *Proc. Natl Acad. Sci.* **118** e2101784118
[17] Herrmann L and Kollmannsberger S 2024 *Comput. Mech.* **74** 281–331
[18] Freire P, Manuylovich E, Prilepsky J E and Turitsyn S K 2023 *Adv. Opt. Photon.* **15** 739–834
[19] Ma W, Liu Z, Kudyshev Z A, Boltasseva A, Cai W and Liu Y 2021 *Nat. Photon.* **15** 77–90
[20] Scheinker A and Pokharel R 2023 *APL Mach. Learn.* **1** 026109
[21] Bedolla E, Padierna L C and Castañeda-Priego R 2020 *J. Phys.: Condens. Matter* **33** 053001
[22] Kutsukake K 2024 *J. Cryst. Growth* **630** 127598
[23] Behler J and Parrinello M 2007 *Phys. Rev. Lett.* **98** 146401
[24] Mueller T, Hernandez A and Wang C 2020 *J. Chem. Phys.* **152** 050902
[25] Friederich P, Häse F, Proppe J and Aspuru-Guzik A 2021 *Nat. Mater.* **20** 750–61
[26] Imbalzano G and Ceriotti M 2021 *Phys. Rev. Mater.* **5** 063804
[27] Fantasia A *et al* 2024 *J. Chem. Phys.* **161** 014110
[28] Abou El Kheir O, Bonati L, Parrinello M and Bernasconi M 2024 *npj Comput. Mater.* **10** 33
[29] Butler K T, Davies D W, Cartwright H, Isayev O and Walsh A 2018 *Nature* **559** 547–55
[30] Himanen L, Geurts A, Foster A S and Rinke P 2019 *Adv. Sci.* **6** 1900808
[31] Dai D, Xu T, Wei X, Ding G, Xu Y, Zhang J and Zhang H 2020 *Comput. Mater. Sci.* **175** 109618
[32] Fiedler L, Shah K, Bussmann M and Cangi A 2022 *Phys. Rev. Mater.* **6** 040301
[33] Linton N and Aidhy D S 2023 *APL Mach. Learn.* **1** 016109
[34] Jin H, Zhang E and Espinosa H D 2023 *Appl. Mech. Rev.* **75** 061001
[35] Lee E H, Jiang W, Alsalman H, Low T and Cherkassky V 2022 *Phys. Rev. Mater.* **6** 043802
[36] Zuccarini C, Ramachandran K and Jayaseelan D D 2024 *APL Mater.* **12** 090601
[37] Chen C-T and Gu G X 2020 *Adv. Sci.* **7** 1902607
[38] Hiraide K, Hirayama K, Endo K and Muramatsu M 2021 *Comput. Mater. Sci.* **190** 110278
[39] Wang J, Kwon H, Kim H S and Lee B-J 2023 *npj Comput. Mater.* **9** 60
[40] Zhang K, Guo Y, Liu X, Hong F, Hou X and Deng Z 2024 *Extreme Mech. Lett.* **69** 102165
[41] Li B, Lowengrub J, Ratz A and Voigt A 2009 *Commun. Comput. Phys.* **6** 433–82 (available at: https://global-sci.com/article/81163/geometric-evolution-laws-for-thin-crystalline-films-modeling-and-numerics)
[42] Provatas N and Elder K 2010 *Phase-Field Methods in Materials Science and Engineering* (Wiley) (https://doi.org/10.1002/9783527631520)
[43] Quarteroni A 2017 *Numerical Models for Differential Problems* (Springer) (https://doi.org/10.1007/978-3-319-49316-9)
[44] Brunton S L, Noack B R and Koumoutsakos P 2020 *Ann. Rev. Fluid Mech.* **52** 477–508
[45] Cai S, Mao Z, Wang Z, Yin M and Karniadakis G E 2021 *Acta Mech. Sin.* **37** 1727–38
[46] Zhao C, Zhang F, Lou W, Wang X and Yang J 2024 *Phys. Fluids* **36** 101301
[47] Bishop C M and Nasrabadi N M 2006 *Pattern Recognition and Machine Learning* vol 4 (Springer)
[48] Zhang A, Lipton Z C, Li M and Smola A J 2023 *Dive Into Deep Learning* (Cambridge University Press)
[49] Schmidhuber J 2015 *Neural Netw.* **61** 85–117
[50] Cybenko G 1989 *Math. Control Signals Syst.* **2** 303–14

[51] Hornik K, Stinchcombe M and White H 1989 *Neural Netw.* **2** 359–66
[52] Gripenberg G 2003 *J. Approx. Theory* **122** 260–6
[53] Csáji B C *et al* 2001 *Approximation with Artificial Neural* (Faculty of Sciences, Etvs Lornd University) vol 24 p 7
[54] DeVore R, Hanin B and Petrova G 2021 *Acta Numer.* **30** 327–444
[55] Brahma P P, Wu D and She Y 2016 *IEEE Trans. Neural Netw. Learn. Syst.* **27** 1997–2008
[56] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *2016 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (IEEE) pp 770–8
[57] Xiao L, Bahri Y, Sohl-Dickstein J, Schoenholz S and Pennington J 2018 Dynamical isometry and a mean field theory of CNNS: how to train 10,000-layer vanilla convolutional neural networks *Proc. 35th Int. Conf. on Machine Learning (ICML'18)* vol 80 pp 5393–402
[58] Hopfield J J 1999 *Rev. Mod. Phys.* **71** S431–7
[59] Zador A M 2019 *Nat. Commun.* **10** 3770
[60] Lillicrap T P, Santoro A, Marris L, Akerman C J and Hinton G 2020 *Nat. Rev. Neurosci.* **21** 335–46
[61] Rasamoelina A D, Adjailia F and Sincák P 2020 A review of activation function for artificial neural network *2020 IEEE 18th World Symp. on Applied Machine Intelligence and Informatics (SAMI)* pp 281–6
[62] Le Cun Y, Jackel L, Boser B, Denker J, Graf H, Guyon I, Henderson D, Howard R and Hubbard W 1989 *IEEE Commun. Mag.* **27** 41–46
[63] Le Cun Y, Boser B, Denker J S, Henderson D, Howard R E, Hubbard W and Jackel L D 1989 *Neural Comput.* **1** 541–51
[64] Gholamalinezhad H and Khosravi H 2020 Pooling methods in deep neural networks, a review (arXiv:2009.07485)
[65] Jia Y, Huang C and Darrell T 2012 Beyond spatial pyramids: Receptive field learning for pooled image features *2012 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* pp 3370–7
[66] Kayhan O S and van Gemert J C 2020 On translation invariance in CNNs: convolutional layers can exploit absolute spatial location *2020 IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)* (IEEE) pp 14262–73
[67] Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, Wang L, Li C and Sun M 2020 *AI Open* **1** 57–81
[68] Reiser P *et al* 2022 *Commun. Mater.* **3** 93
[69] Dong Z, Feng J, Ji Y and Li Y 2023 *J. Phys. Chem. A* **127** 5921–9
[70] Batzner S, Musaelian A, Sun L, Geiger M, Mailoa J P, Kornbluth M, Molinari N, Smidt T E and Kozinsky B 2022 *Nat. Commun.* **13** 2453
[71] Bertin N, Bulatov V V and Zhou F 2024 *npj Comput. Mater.* **10** 192
[72] Tian Y, Bagchi S, Myhill L, Po G, Martinez E, Lin Y T, Mathew N and Perez D 2024 *npj Comput. Mater.* **10** 219
[73] Queen O, McCarver G A, Thatigotla S, Abolins B P, Brown C L, Maroulas V and Vogiatzis K D 2023 *npj Comput. Mater.* **9** 90
[74] Xue T, Gan Z, Liao S and Cao J 2022 *npj Comput. Mater.* **8** 201
[75] Fan S, Hitt A L, Tang M, Sadigh B and Zhou F 2024 *Mach. Learn.: Sci. Technol.* **5** 025027
[76] Alhada–Lahbabi K, Deleruyelle D and Gautier B 2024 *npj Comput. Mater.* **10** 197
[77] Horie M and Mitsume N 2022 Physics-embedded neural networks: graph neural PDE solvers with mixed boundary conditions *Proc. 36th Int. Conf. on Neural Information Processing Systems (NIPS'22)* (Curran Associates, Inc.) pp 23218–29
[78] Jin Z, Zheng B, Kim C and Gu G X 2023 *APL Mach. Learn.* **1** 046109
[79] Xiang Z, Peng W, Yao W, Liu X and Zhang X 2024 *Appl. Soft Comput.* **155** 111437
[80] Wu Z, Pan S, Chen F, Long G, Zhang C and Yu P S 2021 *IEEE Trans. Neural Netw. Learn. Syst.* **32** 4–24
[81] Li P, Pei Y and Li J 2023 *Appl. Soft Comput.* **138** 110176
[82] Alain G and Bengio Y 2014 *J. Mach.: Learn. Res.* **15** 3743–73
[83] Rifai S, Vincent P, Muller X, Glorot X and Bengio Y 2011 Contractive auto-encoders: Explicit invariance during feature extraction *Proc. 28th Int. Conf. on Int. Conf. on Machine Learning (ICML'11)* pp 833–40
[84] Kingma D P and Welling M 2013 Auto-encoding variational Bayes (arXiv:1312.6114)
[85] Ronneberger O, Fischer P and Brox T 2015 U-net: convolutional networks for biomedical image segmentation *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* ed N Navab, J Hornegger, W M Wells and A F Frangi (Springer) pp 234–41
[86] Majumdar S, Sau A, Biswas M and Sarkar R 2024 *Comput. Mater. Sci.* **244** 113199
[87] Lu L, Jin P, Pang G, Zhang Z and Karniadakis G E 2021 *Nat. Mach. Intell.* **3** 218–29
[88] Chen T and Chen H 1995 *IEEE Trans. Neural Netw.* **6** 911–7
[89] Lu L, Meng X, Cai S, Mao Z, Goswami S, Zhang Z and Karniadakis G E 2022 *Comput. Methods Appl. Mech. Eng.* **393** 114778
[90] Hochreiter S, Bengio Y, Frasconi P and Schmidhuber J 2001 Gradient flow in recurrent nets: the difficulty of learning longterm dependencies *A Field Guide to Dynamical Recurrent Networks* (IEEE) pp 237–43
[91] Hochreiter S and Schmidhuber J 1997 *Neural Comput.* **9** 1735–80
[92] Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H and Bengio Y 2014 Learning phrase representations using RNN encoder-decoder for statistical machine translation (arXiv:1406.1078)
[93] Chung J, Gulcehre C, Cho K and Bengio Y 2014 Empirical evaluation of gated recurrent neural networks on sequence modeling (arXiv:1412.3555)
[94] Shewalkar A, Nyavanandi D and Ludwig S A 2019 *J. Artif. Intell. Soft Comput. Res.* **9** 235–45
[95] Gao Y, Wang R and Zhou E 2021 *Sci. Program.* **2021** 4055281
[96] Bengio Y, Louradour J, Collobert R and Weston J 2009 Curriculum learning *Proc. 26th Annual Int. Conf. on Machine Learning (ICML'09)* pp 41–48
[97] Goyal A, Lamb A, Zhang Y, Zhang S, Courville A and Bengio Y 2016 Professor forcing: a new algorithm for training recurrent networks *Proc. 30th Int. Conf. on Neural Information Processing Systems (NIPS'16)* (Curran Associates, Inc.) pp 4608–16
[98] Shi X, Chen Z, Wang H, Yeung D Y, Wong W k and Woo W c 2015 Convolutional LSTM network: a machine learning approach for precipitation nowcasting (arXiv:1506.04214)
[99] Ballas N, Yao L, Pal C and Courville A 2016 Delving deeper into convolutional networks for learning video representations (arXiv:1511.06432)
[100] Wang Y, Long M, Wang J, Gao Z and Yu P S 2017 PredRNN: Recurrent neural networks for predictive learning using spatiotemporal LSTMs *Proc 31st Int. Conf. on Neural Information Processing Systems (NIPS'17)* (Curran Associates, Inc.) pp 879–88
[101] Wang Q, Ma Y, Zhao K and Tian Y 2022 *Ann. Data Sci.* **9** 187–212

[102] Newton D, Yousefian F and Pasupathy R 2018 Stochastic Gradient Descent: Recent Trends *Recent Advances in Optimization and Modeling of Contemporary Problems* (INFORMS) pp 193–220
[103] Tieleman T and Hinton G 2012 Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude *Coursera: Neural networks for machine learning*
[104] Kingma D P and Ba J 2014 Adam: a method for stochastic optimization (arXiv:1412.6980)
[105] Hardt M, Recht B and Singer Y 2016 Train faster, generalize better: stability of stochastic gradient descent *Proc. 33rd Int. Conf. on Machine Learning* (*Proc. Machine Learning Research*) vol 48, ed M F Balcan and K Q Weinberger (PMLR) pp 1225–34
[106] van Merrienboer B, Breuleux O, Bergeron A and Lamblin P 2018 Automatic differentiation in ML: where we are and where we should be going *Proc. 32nd Int. Conf. on Neural Information Processing Systems (NIPS'18)* , pp 8771–81 (Curran Associates, Inc.)
[107] Wythoff B J 1993 *Chemometr. Intell. Lab. Syst.* **18** 115–55
[108] Baydin A G, Pearlmutter B A, Radul A A and Siskind J M 2018 *J. Mach.: Learn. Res.* **18** 1–43
[109] Abadi M *et al* 2015 TensorFlow: large-scale machine learning on heterogeneous systems (available at: www.tensorflow.org)
[110] Paszke A *et al* 2019 PyTorch: an imperative style, high-performance deep learning library *Proc. 33rd Int. Conf. on Neural Information Processing Systems (NIPS'19)*, ed H Wallach, H Larochelle, A Beygelzimer, F d Alché-Buc, E Fox and R Garnett (Curran Associates, Inc.) pp 8026–37
[111] Bradbury J, Frostig R, Hawkins P, Johnson M J, Leary C, Maclaurin D, Necula G, Paszke A, VanderPlas J, Wanderman-Milne S and Zhang Q 2018 JAX: composable transformations of Python+NumPy programs (available at: http://github.com/google/jax)
[112] Ying X 2019 *J. Phys.: Conf. Ser.* **1168** 022022
[113] Lanzoni D, Rovaris F, Martín-Encinar L, Fantasia A, Bergamaschini R and Montalenti F 2024 *APL Mach. Learn.* **2** 036108
[114] Shorten C and Khoshgoftaar T M 2019 *J. Big Data* **6** 1–48
[115] Yin S, Liu C, Zhang Z, Lin Y, Wang D, Tejedor J, Zheng T F and Li Y 2015 *EURASIP J. Audio Speech Music Process.* **2015** 2
[116] Zur R M, Jiang Y, Pesce L L and Drukker K 2009 *Med. Phys.* **36** 4810–8
[117] Bishop C M 1995 *Neural Comput.* **7** 108–16
[118] Bengio Y 2000 *Neural Comput.* **12** 1889–900
[119] Yang L and Shami A 2020 *Neurocomputing* **415** 295–316
[120] Akiba T, Sano S, Yanase T, Ohta T and Koyama M 2019 Optuna: a next-generation hyperparameter optimization framework *Proc. 25th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (ACM) pp 2623–31
[121] Rezatofighi H, Tsoi N, Gwak J, Sadeghian A, Reid I and Savarese S 2019 Generalized intersection over union: a metric and a loss for bounding box regression *2019 IEEE/CVF Conf. on Computer Vision and Pattern Recognition* pp 658–66 (IEEE)
[122] Penney G, Weese J, Little J, Desmedt P, Hill D and Hawkes D 1998 *IEEE Trans. Med. Imaging* **17** 586–95
[123] Wang Z, Bovik A, Sheikh H and Simoncelli E 2004 *IEEE Trans. Image Process.* **13** 600–12
[124] Lanzoni D, Albani M, Bergamaschini R and Montalenti F 2022 *Phys. Rev. Mater.* **6** 103801
[125] Lanzoni D, Fantasia A, Bergamaschini R, Pierre-Louis O and Montalenti F 2024 *Mach. Learn.: Sci. Technol.* **5** 045017
[126] Yang K, Cao Y, Zhang Y, Fan S, Tang M, Aberg D, Sadigh B and Zhou F 2021 *Patterns* **2** 100243
[127] Yabansu Y C, Steinmetz P, Hötzer J, Kalidindi S R and Nestler B 2017 *Acta Mater.* **124** 182–94
[128] Hu C, Martin S and Dingreville R 2022 *Comput. Methods Appl. Mech. Eng.* **397** 115128
[129] Teichert G H and Garikipati K 2019 *Comput. Methods Appl. Mech. Eng.* **344** 666–93
[130] Zhang X and Garikipati K 2020 *Comput. Methods Appl. Mech. Eng.* **372** 1–24
[131] Strayer S T, Templeton W J F, Dugast F X, Narra S P and To A C 2022 *Add. Manuf. Lett.* **3** 100081
[132] Kemeth F P, Alonso S, Echebarria B, Moldenhawer T, Beta C and Kevrekidis I G 2023 *Phys. Rev. E* **107** 025305
[133] Bergamaschini R, Salvalaglio M, Backofen R, Voigt A and Montalenti F 2016 *Adv. Phys. X* **1** 331–67
[134] Martín-Encinar L, Lanzoni D, Fantasia A, Rovaris F, Bergamaschini R and Montalenti F 2025 *Comput. Mater. Sci.* **249** 113657
[135] Kardar M, Parisi G and Zhang Y-C 1986 *Phys. Rev. Lett.* **56** 889–92
[136] Choi J Y, Xue T, Liao S and Cao J 2024 *Add. Manuf.* **79** 103938
[137] Montes De Oca Zapiain D, Stewart J A and Dingreville R 2021 *npj Comput. Mater.* **7** 3
[138] Qin Y, DeWitt S, Radhakrishnan B and Biros G 2023 *Comput. Mater. Sci.* **218** 111927
[139] Peivaste I, Siboni N H, Alahyarizadeh G, Ghaderi R, Svendsen B, Raabe D and Mianroodi J R 2022 *Comput. Mater. Sci.* **214** 111750
[140] Oommen V, Shukla K, Desai S, Dingreville R and Karniadakis G E 2024 *npj Comput. Mater.* **10** 145
[141] Dingreville R, Roberston A E, Attari V, Greenwood M, Ofori-Opoku N, Ramesh M, Voorhees P W and Zhang Q 2024 *Modelling Simul. Mater. Sci. Eng.* **32** 065019
[142] Liu J, Huang M, Li Z, Zhao L and Zhu Y 2022 *Eng. Fract. Mech.* **264** 108332
[143] Wu P, Iquebal A S and Ankit K 2023 *Comput. Mater. Sci.* **224** 112187
[144] Wang Z, Dabaja R, Chen L and Banu M 2023 *Sci. Rep.* **13** 5414
[145] Kazemzadeh Farizhandi A A and Mamivand M 2023 *Comput. Mater. Sci.* **223** 112110
[146] Alhada-Lahbabi K, Deleruyelle D and Gautier B 2023 *ACS Appl. Electr. Mater.* **5** 3894–907
[147] Tseng B-Y, Guo C W C, Chien Y-C, Wang J-P and Yu C-H 2023 *Adv. Sci.* **10** 2207731
[148] Pinto D, Greenwood M and Provatas N 2024 arXiv:2406.15326
[149] Kwon Y, Thornton K and Voorhees P 2010 *Phil. Mag.* **90** 317–35
[150] Pfaff T, Fortunato M, Sanchez-Gonzalez A and Battaglia P W 2020 Learning mesh-based simulation with graph networks (arXiv:2010.03409)
[151] Alhada-Lahbabi K, Deleruyelle D and Gautier B 2024 *Nanoscale Adv.* **6** 2350–62
[152] Ciesielski D, Li Y, Hu S, King E, Corbey J and Stinis P 2025 *Comput. Mater. Sci.* **246** 113417
[153] Huang P, Leng Y, Lian C and Liu H 2024 *Engineering* **39** 94–103
[154] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A and Bengio Y 2020 *Commun. ACM* **63** 139–44
[155] Fetni S, Pham T Q D, Hoang T V, Tran H S, Duchêne L, Tran X-V and Habraken A M 2023 *Comput. Mater. Sci.* **216** 111820
[156] Oommen V, Shukla K, Goswami S, Dingreville R and Karniadakis G E 2022 *npj Comput. Mater.* **8** 190
[157] Friedrich S and Friede T 2024 *Biometr. J.* **66** 2200212
[158] Wilkinson M D *et al* 2016 *Sci. Data* **3** 160018

[159] Nguyen B D, Potapenko P, Demirci A, Govind K, Bompas S and Sandfeld S 2024 *Mach. Learn. Appl.* **16** 100544
[160] Lejeune E 2020 *Extreme Mech. Lett.* **36** 100659
[161] Wheeler D *et al* 2019 *J. Open Res. Softw.* **7** 29
[162] Takamoto M, Praditia T, Leiteritz R, MacKinlay D, Alesiani F, Pflüger D and Niepert M 2022 Proc. 36th Int. Conf. on Neural Information Processing Systems (NIPS'22) (Curran Associates, Inc.) pp 1596–611
[163] Hassan S M S, Feeney A, Dhruv A, Kim J, Suh Y, Ryu J, Won Y and Chandramowlishwaran A 2023 BubbleML: a multiphase multiphysics dataset and benchmarks for machine learning *Proc. 37th Int. Conf. on Neural Information Processing Systems* (*NIPS'23*) (Curran Associates, Inc.) pp 418–49
[164] Rieger L H, Zelic K, Mele I, Katrasnik T and Bhowmik A 2024 *Sci. Data* **11** 1275
[165] Lagaris I, Likas A and Fotiadis D 1998 *IEEE Trans. Neural Netw.* **9** 987–1000
[166] Dissanayake M W M G and Phan-Thien N 1994 *Commun. Numer. Methods Eng.* **10** 195–201
[167] Raissi M, Perdikaris P and Karniadakis G E 2017 Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations (arXiv:1711.10561)
[168] Sirignano J and Spiliopoulos K 2018 *J. Comput. Phys.* **375** 1339–64
[169] Weinan E and Yu B 2018 *Commun. Math. Stat.* **6** 1–12
[170] Karniadakis G E, Kevrekidis I G, Lu L, Perdikaris P, Wang S and Yang L 2021 *Nat. Rev. Phys.* **3** 422–40
[171] Al-Aradi A, Correia A, Jardim G, de Freitas Naiff D and Saporito Y 2022 *Appl. Math. Comput.* **430** 127287
[172] Wight C L and Zhao J 2021 *Commun. Comput. Phys.* **29** 930–54
[173] Li W, Fang R, Jiao J, Vassilakis G N and Zhu J 2024 *APL Mach. Learn.* **2** 031101
[174] Chen X, Chen R, Wan Q, Xu R and Liu J 2021 *Sci. Rep.* **11** 19507
[175] Niaki S A, Haghighat E, Campbell T, Poursartip A and Vaziri R 2021 *Comput. Methods Appl. Mech. Eng.* **384** 113959
[176] Cai S, Wang Z, Fuest F, Jeon Y J, Gray C and Karniadakis G E 2021 *J. Fluid Mech.* **915** A102
[177] Wang S, Wang H and Perdikaris P 2021 *Sci. Adv.* **7** eabi8605
[178] Kovachki N, Li Z, Liu B, Azizzadenesheli K, Bhattacharya K, Stuart A and Anandkumar A 2023 *J. Mach.: Learn. Res.* **24** 1–97
[179] Koric S and Abueidda D W 2023 *Int. J. Heat Mass Transfer* **203** 123809
[180] Li W, Bazant M Z and Zhu J 2023 *Comput. Methods Appl. Mech. Eng.* **416** 116299
[181] Goswami S, Yin M, Yu Y and Karniadakis G E 2022 *Comput. Methods Appl. Mech. Eng.* **391** 114587
[182] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser L and Polosukhin I 2017 Attention is all you need *Proc. 31st Int. Conf. on Neural Information Processing Systems (NIPS'17)* (Curran Associates, Inc.) pp 6000–10
[183] Niu Z, Zhong G and Yu H 2021 *Neurocomputing* **452** 48–62
[184] Brauwers G and Frasincar F 2023 *IEEE Trans. Knowl. Data Eng.* **35** 3279–98
[185] Banik D, Pati N and Sharma A 2024 *Artif. Intell. Rev.* **57** 257
[186] Zhou C *et al* 2024 *Int. J. Mach. Learn. Cybern.* (https://doi.org/10.1007/s13042-024-02443-6)
[187] Dosovitskiy A *et al* 2021 An image is worth 16 × 16 words: transformers for image recognition at scale (arXiv:2010.11929)
[188] Hassanin M, Anwar S, Radwan I, Khan F S and Mian A 2024 *Inf. Fusion* **108** 102417
[189] Khan S, Naseer M, Hayat M, Zamir S W, Khan F S and Shah M 2022 *ACM Comput. Surv.* **54** 200
[190] Gao Z, Zhu C, Shu Y and Wang C 2024 *Mater. Today Commun.* **40** 109477
[191] Chen Y, Dodwell T, Chuaqui T and Butler R 2023 *Eng. Fract. Mech.* **286** 109314
[192] Bahdanau D, Cho K and Bengio Y 2014 arXiv:1409.0473
[193] Wang X, Girshick R, Gupta A and He K 2018 Non-local neural networks *2018 IEEE/CVF Conf. on Computer Vision and Pattern Recognition* (IEEE) pp 7794–803
[194] Huang Z, Wang X, Huang L, Huang C, Wei Y and Liu W 2019 CCNet: criss-cross attention for semantic segmentation *2019 IEEE/CVF Int. Conf. on Computer Vision (ICCV)* pp 603–12 (IEEE)
[195] Kaplan J, McCandlish S, Henighan T, Brown T B, Chess B, Child R, Gray S, Radford A, Wu J and Amodei D 2020 arXiv:2001.08361
[196] Kerg G, Kanuparthi B, Goyal A, Goyette K, Bengio Y and Lajoie G 2020 19443–54
[197] Dai Z, Yang Z, Yang Y, Carbonell J, Le Q V and Salakhutdinov R 2019 arXiv:1901.02860
[198] Kitaev N, Ł K and Levskaya A 2020 arXiv:2001.04451
[199] Gehring J, Auli M, Grangier D, Yarats D and Dauphin Y N 2017 Convolutional sequence to sequence learning *Proc. 34th Int. Conf. on Machine Learning* (ICML'17) vol 70 pp 1243–52
[200] Arulkumaran K, Deisenroth M P, Brundage M and Bharath A A 2017 *IEEE Signal Process. Mag.* **34** 26–38
[201] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D and Riedmiller M 2013 arXiv:1312.5602
[202] Yang H and Demkowicz M J 2024 *Comput. Mater. Sci.* **231** 112577
[203] Alhada-Lahbabi K, Deleruyelle D and Gautier B 2025 *ACS Appl. Electr. Mater.* **7** 1130–41
[204] Smith B R, Pant B, Liu Y, Liu Y-C, Yang J-C, Jesse S, Khojandi A, Kalinin S V, Cao Y and Vasudevan R K 2024 *Digit. Discovery* **3** 456–66
[205] Boccardo F and Pierre-Louis O 2022 *Phys. Rev. Lett.* **128** 256102
[206] Boccardo F and Pierre-Louis O 2024 *Phys. Rev. E* **110** L023301
[207] Ren P, Xiao Y, Chang X, Huang P Y, Li Z, Gupta B B, Chen X and Wang X 2021 *ACM Comput. Surv.* **54** 180:1–180:40
[208] Dong X, Yu Z, Cao W, Shi Y and Ma Q 2020 *Front. Comput. Sci.* **14** 241–58
[209] Ganaie M A, Hu M, Malik A K, Tanveer M and Suganthan P N 2022 *Eng. Appl. Artif. Intell.* **115** 105151
[210] Zhou Z H 2021 Ensemble learning *Machine Learning* (Springer) pp 181–210
[211] Schran C, Brezina K and Marsalek O 2020 *J. Chem. Phys.* **153** 104105
[212] Jiang X, Wang X, Wen Z, Li E and Wang H 2023 *Int. Commun. Heat Mass Transfer* **147** 106940
[213] Raissi M, Perdikaris P and Karniadakis G E 2017 Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations (arXiv:1711.10566)
[214] Chen Z, Liu Y and Sun H 2021 *Nat. Commun.* **12** 6136
[215] Lin D and Yu H-Y 2022 *Phys. Rev. E* **106** 014503
[216] Zhang E, Yin M and Karniadakis G E 2020 Physics-informed neural networks for nonhomogeneous material identification in elasticity imaging (arXiv:2009.04525)
[217] Yin M, Zheng X, Humphrey J D and Karniadakis G E 2021 *Comput. Methods Appl. Mech. Eng.* **375** 113603
[218] Heinlein A, Klawonn A, Lanser M and Weber J 2021 *GAMM-Mitteilungen* **44** e202100001
[219] Bar-Sinai Y, Hoyer S, Hickey J and Brenner M P 2019 *Proc. Natl Acad. Sci.* **116** 15344–9
[220] Fernández-Godino M G 2023 *Adv. Comput. Sci. Eng.* **1** 351–400

[221] Teichert G H, Natarajan A R, Van der Ven A and Garikipati K 2019 *Comput. Methods Appl. Mech. Eng.* **353** 201–16
[222] Shojaei M F, Holber J, Das S, Teichert G H, Mueller T, Hung L, Gavini V and Garikipati K 2024 *J. Mech. Phys. Solids* **190** 105726
[223] Xiao S, Hu R, Li Z, Attarian S, Björk K M and Lendasse A 2020 *Neural Comput. Appl.* **32** 14359–73
[224] Sase K and Shibuta Y 2023 *Acta Mater.* **259** 119295
[225] Kingma D P and Welling M 2022 Auto-encoding variational bayes (arXiv:1312.6114)
[226] Ho J, Jain A and Abbeel P 2020 Denoising diffusion probabilistic models *Proc. 34th Int. Conf. on Neural Information Processing Systems* (*NIPS'20*) (Curran Associates, Inc.)
[227] Lanzoni D, Pierre-Louis O and Montalenti F 2023 *J. Chem. Phys.* **159** 307
[228] Stinis P, Daskalakis C and Atzberger P J 2024 *J. Comput. Phys.* **519** 113442
[229] Yeo K, Li Z and Gifford W 2022 *SIAM J. Sci. Comput.* **44** A2150–75
[230] Hsu T, Sadigh B, Bulatov V and Zhou F 2024 *J. Chem. Theory Comput.* **20** 2335–48
[231] Yang L, Zhang D and Karniadakis G E 2020 *SIAM J. Sci. Comput.* **42** A292–317
[232] Shen C *et al* 2024 *Nat. Commun.* **15** 2724
[233] Shen C *et al* 2024 *In-situ* self-optimization of quantum dot emission for lasers by machine-learning assisted epitaxy (arXiv:2411.00332)
[234] Jeong J, Kim T and Lee J 2022 *Micro Nano Syst. Lett.* **10** 22
[235] Pilania G, Gubernatis J and Lookman T 2017 *Comput. Mater. Sci.* **129** 156–63
[236] Chen C, Zuo Y, Ye W, Li X and Ong S P 2021 *Nat. Comput. Sci.* **1** 46–53
[237] Yeom J, Stan T, Hong S and Voorhees P W 2021 *Acta Mater.* **214** 116990