A minimalistic approach to physics-informed machine learning using neighbour lists as physics-optimized convolutions for inverse problems involving particle systems

Alexiadis, Alessio

DOI:
10.1016/j.jcp.2022.111750

License:
Creative Commons: Attribution (CC BY)

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Alexiadis, A 2023, 'A minimalistic approach to physics-informed machine learning using neighbour lists as physics-optimized convolutions for inverse problems involving particle systems', *Journal of Computational Physics*, vol. 473, 111750. https://doi.org/10.1016/j.jcp.2022.111750

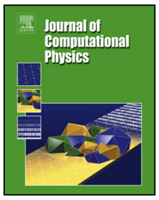
Link to publication on Research at Birmingham portal

Journal of Computational Physics 473 (2023) 111750

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp



A minimalistic approach to physics-informed machine learning using neighbour lists as physics-optimized convolutions for inverse problems involving particle systems

Alessio Alexiadis

School of Chemical Engineering, University of Birmingham, Birmingham B15 2TT, UK

ARTICLE INFO

Article history:
Received 3 December 2021
Received in revised form 20 May 2022
Accepted 30 October 2022
Available online 7 November 2022

Dataset link: edata.bham.ac.uk/744/

Keywords:
Physics-informed machine learning
Molecular dynamics
Smoothed particle hydrodynamics
Discrete element method

ABSTRACT

This study proposes a hybrid approach for combining mechanistic (first principle) and Machine Learning models. This approach applies to discrete (particle-based) systems and continuous systems that can be recast as a particle problem by a framework like Smoothed Particle Hydrodynamics. The governing equations are written as a set of equations describing the motion of the particle system. Artificial Neural Networks are used to derive from data the forces acting on the particles, while the system's path in the state-space is calculated with the equation of motion. This ensures that fundamental physical principles such as Newton's laws of motion are always satisfied in a strong sense. Neighbour lists automatically introduce dimensionality reduction into the system by functioning as physics-optimized convolutions. Therefore, the network can be smaller, simpler, and more easily trainable than other physics-informed machine learning models. The proposed technique is applied to three inverse modelling problems. The method is designed to learn the pairwise forces acting between particles without knowing these forces from the training data. In fact, the training data contains the total force acting on each particle, not the pairwise forces between pairs of particles. Data for Molecular Dynamics, Smoothed Particle Hydrodynamics and Discrete Element Method simulations are fed into the model that 'extracts' their physics and reproduces the simulations with a high degree of accuracy. The model's capability for generalization is noteworthy. As long as the underlying physics remains the same, the model can predict the dynamics of systems with geometries and boundary conditions very different from those of the training dataset. As a remarkable example, a model trained surface-flow data also correctly replicates channel flow.

1. Introduction

The last few years have witnessed a growing interest in computational methods that combine mechanistic (first principle) and Machine Learning (ML) models. A group of these methods, generally known as "physics-guided ML", "physics-informed ML" or "physics-aware AI", aims at integrating data analysis and mechanistic models to solve a variety of complex or ill-posed problems such as inverse modelling, model-order reduction and uncertainty quantification. The idea is to combine

E-mail address: a.alexiadis@bham.ac.uk.

https://doi.org/10.1016/j.jcp.2022.111750

the ability of first principles models to comply with the well-established laws of physics with the ability of ML to learn from data (see [1] and [2] for a review).

To achieve this goal, different techniques have been proposed. Physics-Informed neural networks (PINNs), for instance, incorporate physical knowledge into the ML model by adding constraints into the loss function [3], [4]. Other methods, such as Graph neural networks (GNNs) [5], design new ML architectures to capture physics-based dependences among variables. A third group, known as hybrid physics-ML models, replaces, with ML, one or more components of the mechanistic model that are poorly modelled using physics [6].

This study proposes a hybrid physics-ML approach that applies to discrete (particle-based) systems. If the system is continuous, it can be recast, at least in theory (in practice it is not always a trivial task), as a particle problem by a method like Smoothed Particle Hydrodynamics [7] or similar [8]. In this way, the system can be represented by N non-relativistic particles moving according to Newton's laws of motion. Mathematically, there are several equivalent ways to represent the equation of motion. One option is the Lagrange's equation of motion [9]

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = Q_i \tag{1}$$

where q_i are a set of generalized coordinates used to describe the motion of the system and $\dot{q}_i$ the corresponding momenta. $\mathscr{L}$ is the so-called Lagrangian, defined as the difference between the kinetic energy T and the potential energy V, and $\mathcal{Q}_i$ the generalized dissipative forces. All (physical) particle systems, from molecules to galaxies, must satisfy eq. (1) (or any equivalent equation of motion): the difference between one system and another lies in the functional form of V and $\mathcal{Q}_i$.

This study proposes a 'minimalistic' approach to physics-informed ML. The ML model, and specifically an Artificial Neural Network (ANN), will not learn eq. (1), but it will target specific terms like V (more precisely $-\nabla V$, the negative gradient of V) and $\mathcal{Q}_i$, which represent the forces acting on the particles. This has several implications that are gradually introduced, justified, and discussed in the rest of the paper.

1. It guarantees the path of the system is always consistent with Newton's laws of motion (see, for instance, Sections 4.6 and 5.6): even at the beginning of the training phase, when $-\nabla V$ and $\mathcal{Q}_i$ are random functions.
2. We do not need to incorporate constraints into the loss function to progressively steer predictions towards physically consistent outputs. Physical consistency is enforced in a 'strong' sense (see Section 4.6).
3. The ML model does not reproduce the next frame of the simulation knowing the previous one: this is done by the equation of motion. Thus, the ANN can be smaller and more easily trainable (see Sections 4.4, 5.4 and 6.4).
4. In the minimalistic method, neighbour lists play the same role that convolutional layers play in Convolutional Neural Networks (CNNs) (see Sections 2, 4.2, 4.3 and 7).
5. The input of the ANN is not the full variable space, but a subset determined by the size of the neighbour lists. This reduces the dimensionality of the system but requires a tweak to the loss function (see Sections 4.3, 5.3, 6.3 and 7).
6. The use of neighbour lists ensures that $-\nabla V$ and $\mathcal{Q}_i$ are space and time-invariant. The ML model does not see the system as a series of image-like inputs, but as a series of neighbour lists that evolve under the action of forces, whose functional form remains invariant during the evolution of the system (see Section 7).

In this article, the minimalistic approach is applied to inverse problems involving three different physical models. We know the output of a particle simulation, and we want to discover the unknown physics that originated that output. These three problems are used as case studies to progressively introduce the methodology. Neighbour lists, which play a fundamental role in the method, are discussed in the first case study. The issue of noisy data is tackled in the second case study, while a technique for handling unknown boundary conditions is presented in the third. To facilitate the explanation, the article does not follow a traditional subdivision into 'Methods', 'Results' and 'Discussion'. Each case study has its own 'Methods', 'Results' and 'Discussion'. In this way, the ideas behind the minimalistic approach are gradually introduced together with the numerical methods used for each specific inverse problem. Finally, the 'Conclusions' section links the three case studies into a general view of the minimalist method.

2. Relation to other work

A large amount of work has been performed in the last five years on the topic of learning the dynamics of discrete systems. Neural networks were trained to reproduce the Hamiltonian [10,11], Lagrangian [12], [13], or potential [14] of discrete systems. The size of the network is proportional to the degrees of freedom of the system and these studies are limited to a few degrees of freedom. As the number of particles increases (discrete systems can easily involve millions of particles), the complexity of the network makes training more difficult and computationally expensive. For structured data, convolutional layers are often used to reduce data dimensionality improving training performance. However, CNNs cannot be used for unstructured data such as particle systems. Therefore, new types of networks have been proposed for these systems. Examples are Behler-Parrinello neural networks [15], Gradient-domain machine learning [16], Deep Potential Molecular Dynamics [14], Hamiltonian neural networks [10], Lagrangian neural networks [13], Graph neural networks [5], SympNets [11], SplineCNNs [17], MoNets [18], GMLS-Nets [19] and SpiderCNN [20].

Instead of proposing a new type of network, this paper takes a different direction.

Since similarities between neural networks and particle methods have been noted [22], [23], is it possible that data structures commonly used in particle methods can achieve dimensionality reduction in a way similar to convolutions in structured data? This study answers positively to this question and identifies these structures with neighbour lists, first proposed in 1967 [21]. Convolutional layers extract local features that are translational invariant; similarly, neighbour lists capture features in a given neighbourhood that are permutation invariant. However, contrary to convolutional layers, neighbour lists do not need training: their only adjustable parameter is the cut-off radius, which is dictated by the physics of the system. In a way, we can consider neighbour lists as a 'physics-optimized' substitute of convolutional layers for particle systems.

Based on these considerations, this paper shows that, by looking at the system as a collection of neighbour lists rather than particles, special network architectures are not required, and feedforward neural networks are perfectly adequate. However, as explained in Section 4.3, this will require a modification of the loss function.

3. Inverse problems and case studies

Let us assume, we have a typical output file of a particle simulation that records positions **r**, velocities **v** and accelerations **a** of all N particles in the system at different times t. Sometimes, **a** or **v** are not stored in the simulation output and must be recalculated from **r**. In this case, depending on the difference between the simulation timestep and the frequency with which data are recoded, noise could be introduced into the data in the form of numerical errors. Noise will be discussed in Case Study 2. In the following sections, unless otherwise specified, the dataset is assumed noise-free.

The goal of the inverse problem is to calculate the factors that produced the simulations only knowing the output. To achieve this aim, we employ an ANN that 'learns' the physics of the systems, which, in our case, means the pairwise forces $-\nabla V$ and $\mathcal{Q}_i$. However, as explained in Section 4.3, there is a mismatch between the available training data (i.e. the total forces acting on each particle) and the output of the ANN (i.e. the pairwise forces exchanged by pairs of particles), which requires a tweak to the loss function.

In Case Study 1, the output comes from a Molecular Dynamics (MD) simulation; in Case Study 2, from Smoothed Particle Hydrodynamics (SPH); and in Case Study 3, from the Discrete Element Method (DEM). All simulations are two-dimensional but can be easily extended to the three-dimensional case. The systematic optimization of the network architecture and hyperparameters is beyond the scope of this work. A reasonable network was found by trial-and-error for Case Study 1 and applied to the other case studies by changing the input size when necessary.

4. Case study 1, Molecular Dynamics: conservative, ergodic systems

The first case study deals with systems of atoms/molecules simulated with MD. These systems are conservative meaning that $\mathcal{Q}_i = 0$ and $-\nabla V$ is a function of the molecular positions only. They are also ergodic: provided enough time, the trajectory will eventually visit all parts of the state-space. This helps training since the dataset covers uniformly the state-space. The next section provides a brief overview of MD; the reader can refer to Allen and Tildesley [24] for more details.

4.1. Molecular Dynamics background

In Molecular Dynamics (MD), the trajectory of particles representing molecules or atoms is calculated by solving eq. (1), where interactions between particles are defined by intermolecular forces. In this case study, we consider N particle moving in a two-dimensional space contained in a computational box of width L_x and height L_y with periodic boundary conditions. The generalized coordinates $q_1, \ldots, q_{2N}$ are given by the coordinates $\mathbf{r}_1(x_1, y_1), \ldots, \mathbf{r}_N\,(x_N,\, y_N)$ of each particle. The system is conservative ($\mathcal{Q}_i = 0$), and the simulations run in the microcanonical ensemble so that the total energy of the system remains constant. Conservative forces acting on the ith particle are derived from the gradient of a scalar potential $V(\mathbf{r}_1, \ldots, \mathbf{r}_N)$

$$\mathbf{F}_i = -\frac{\partial V\left(\mathbf{r}_i\right)}{\partial \mathbf{r}_i} = -\nabla V, \tag{2}$$

which is a function of the particle positions $\mathbf{r}_1, \ldots, \mathbf{r}_N$ only.

In general, V can be divided into different terms depending on the coordinates of individual molecules, pairs of molecules, triplets, etc.

$$V = \sum_{i}^{N} V_1\left(\mathbf{r}_i\right) + \sum_{i}^{N}\sum_{j>i}^{N} V_2\left(\mathbf{r}_i, \mathbf{r}_j\right) + \sum_{i}^{N}\sum_{j>i}^{N}\sum_{j>k}^{N} V_3\left(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k\right) + \cdots . \tag{3}$$

V_1 represents the effect on the system of an external field such as gravity, but it may also include interactions at the boundaries such as the box walls. In Case Study 1, we do not consider external fields and we apply periodic boundary conditions, therefore $V_1 = 0$. V_2 represents the pair potential, i.e. the forces exchanged by pairs of molecules. V_3 is the

Table 1
MD reduced units assuming σ^*=1, ε^*=1, m^*=1.

Reduce Unit	Formula
Length r^*	$r^* = \frac{r}{\sigma}$
Time t^*	$t^* = \sqrt{\frac{\varepsilon}{m\sigma^2}}$
Temperature T^*	$T^* = \frac{k_B T}{\varepsilon}$
Force F^*	$F^* = \frac{F\sigma}{\varepsilon}$
Energy U^*	$U^* = \frac{U}{\varepsilon}$
Pressure p^*	$p^* = \frac{p\sigma^3}{\varepsilon}$
Density r^*	$\rho^* = \rho\sigma^3$

Table 2
MD simulation parameters in reduced units.

Width simulation box L^*x	20
Height simulation box L^*y	20
Density r^*	0.8
Number of molecules N	320
Timestep Dt^*	0.005
Temperature T^*	1
Cut-off r_c^*	2.5

three-bodies potential, i.e. the forces exchanged by triplets of molecules and so on. V_2 is usually the dominant term and, most of the time, the only accounted for in MD simulations. Here, we assume $V \approx V_2$ and neglect the multi-body terms in eq. (3).

To conserve angular momentum, the force between particle i and j in the pair potential V_2 must depends only on the magnitude of the pair separation r_{ij} = $||\mathbf{r}_j - \mathbf{r}_i||$. Thus, the force excerpted to particle i from particle j can be written as

$$\mathbf{f}_{ij} = \mathbf{f}\left(\mathbf{r}_{ij}\right) = -\frac{\partial V}{\partial \mathbf{r}_{ij}} = -\frac{\partial \mathrm{V}}{\partial r_{ij}} \frac{\mathbf{r}_{ij}}{r_{ij}}, \tag{4}$$

where $\mathbf{r}_{ij}$ = $\mathbf{r}_j - \mathbf{r}_i$. To conserve linear momentum, it must also be $\mathbf{f}_{ji}$ = $-\mathbf{f}_{ij}$.

The simulation is carried out with the classic Lennard-Jones (LJ) pair potential

$$V(r_{ij}) = \mathbf{4}\varepsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{\mathbf{12}} - \left(\frac{\sigma}{r_{ij}}\right)^{\mathbf{6}}\right], \tag{5}$$

where ε is the so-called dispersion energy and σ a measure of the size of the molecules. From eq. (5), we can first derive the magnitude of the force

$$f_{ij} = -\frac{\partial V}{\partial r_{ij}}, \tag{6}$$

and then, calculating the direction cosines $\mathbf{r}_{ij}/r_{ij}$ from the particle positions, we determine the force in eq. (4). In the simulation, we use reduced variables, i.e. dimensionless variables reduced on the basis of σ, ε, m (see Table 1).

4.2. Simulation details

Details of the simulation in reduced units are given in Table 2. The simulation is run for 100,000 timesteps and results written to the output file every 10,000 timesteps. This output will provide the dataset for training the ANN.

In general, the force f_{ij} decays with r_{ij}. Therefore, to reduce computational time, numerical codes use neighbour lists: data structures that maintains a list of all particles within a given cut-off distance of each other. The force f_{ij} exchanged between pairs is not calculated for all possible i-j pairs, but only for particles whose distance is within the cut-off radius r_c. Any particle j with $r_{ij} < r_c$ belongs the neighbour list of particle i, while those with $r_{ij} > r_c$ are ignored when calculating the particle's total force $\mathbf{F}_i$ (Fig. 1).

The input of the ANN can be the neighbour list rather than the whole system of particles reducing the dimensionality of the problem from N (total number of particles) to N_ℓ (particles in the neighbour list). In this way, the neighbour list works as a sort of Lagrangian convolutional layer that is constructed, at each timestep, based on the relative distances among particles to optimally represent the physics of the system. Moreover, if the forces are calculated from a pair-potential, we can further reduce the size of the input layer from N_ℓ to the inputs of the potential. This requires some modifications to the loss function as explained in the next section.

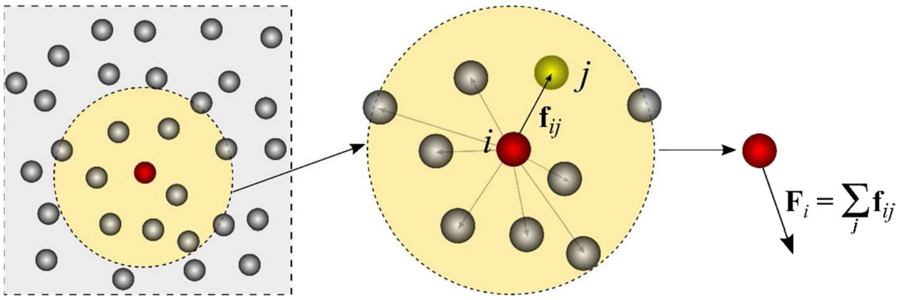


Fig. 1. Neighbour list of particle i. Particle i exchanges pairwise forces $\mathbf{f}_{ij}$ with all other j particles in its neighbour list. The final force $\mathbf{F}_i$ on particle i is given by the sum of all $\mathbf{f}_{ij}$ forces.

4.3. The minimalistic approach for solving the inverse MD problem

The goal of the inverse problem is to train an ANN to replicate the pairwise forces f_{ij}. However, there are still two obstacles before the network can be trained. The first is that the dataset provides accelerations rather than forces and the particle mass m is not necessarily known. We can solve this problem by using reduced units (Table 1), which scale forces with respect to m making the accelerations $\mathbf{a}_i$ numerically equivalent to the forces $\mathbf{F}_i$. The second obstacle requires changes to the loss function. The output of the ANN is the pairwise force f_{ij}, however, the dataset only provides the total force $\mathbf{F}_i$ acting on particle i. In other words, there is a mismatch between $f_{ij}{}^{\text{TARGET}}$, the target we would like to have, and $\mathbf{F}_i{}^{\text{TARGET}}$, the target we actually have. Nevertheless, the total force acting on particle i is related with the pairwise forces between i and the neighbour particles j

$$\mathbf{F}_i = \sum_{j \neq i}^{N_\ell} \mathbf{f}_{ij} = \sum_{j \neq i}^{N_\ell} f_{ij} \frac{\mathbf{r}_{ij}}{r_{ij}}. \tag{7}$$

Therefore, we can solve the problem by modifying the training loss function. Since $f_{ij}{}^{\text{TARGET}}$ is unknown, instead of using a traditional loss function based on $\|\ f_{ij}{}^{\text{TARGET}} - f_{ij}{}^{\text{ANN}}\ \|^2$, we redefine the loss as

$$L_i = \left\| \mathbf{F}_i^{\boldsymbol{TARGET}} - \sum_{j \neq i}^{N_{\ell,i,}} f_{ij}^{ANN} \frac{\mathbf{r}_{ij}}{r_{ij}} \right\|^2 . \tag{8}$$

This is the loss for a single neighbour list for a single timestep. The total loss is obtained by adding the losses L_i of all the N neighbour lists for all N_t time steps

$$L = \sum_t^{N_t} \frac{\sum_i^N \left\| \mathbf{F}_{i,t}^{\boldsymbol{TARGET}} - \sum_{j \neq i}^{N_{\ell,i,t}} f_{ij,t}^{ANN} \frac{\mathbf{r}_{ij,t}}{r_{ij,t}} \right\|^2}{N N_t}. \tag{9}$$

In eq. (9), it is $N_{\ell,i,t}$ rather than N_ℓ because the number of particles in the neighbour list can be different for each particle i and each timestep t.

4.4. Training of the ANN

The network architecture and training hyperparameters are reported in Table 3. For the ANN input, we use $1/r_{ij}$ rather than r_{ij}. Strictly speaking, this is not necessary (it is not used in Case Study 3, for instance) but provides a practical benefit. Not all the neighbour lists have the same size but fixing the size of the array storing the list improves coding efficiency. Therefore, certain entries will be empty. These missing data can be indicated by a zero that, if we use $1/r_{ij}$, corresponds to a particle at infinity and, therefore, outside the neighbour list.

We train the network with 11 timesteps and 320 particles, which corresponds to 3520 training data: 80% is used for training and the remaining for validation. The loss function during training is shown in Fig. 2a. By using neighbour lists, the actual number of training data is augmented by a factor of N and 11 timesteps are enough to train the network.

The dataset provides the particle positions not the neighbour lists that must be recalculated before training. In the inverse problem, we do not know what r_c was used to build the neighbour lists during the simulation. However, when recalculating the lists from data, we can use an arbitrary cut-off $r_c{}^{\text{ANN}}$. As long as $r_c{}^{\text{ANN}} > r_c$, the ANN automatically understands that data above a certain distance do not affect the output. Typically, r_c is estimated from the physics of the problem. In MD, for instance, it usually varies in the range $[2\sigma, 3\sigma]$. Therefore, assuming $r_c{}^{\text{ANN}} = 4\sigma$ can be safe choice.

Table 3
Network Architecture and Training hyperparameters for the MD inverse problem.

ANN architecture		
	Neurons	Activation function
Input layer	1 representing $1/r_{ij}$	–
Hidden Layer 1	8	Elu
Hidden Layer 2	16	Elu
Hidden Layer 3	8	Elu
Output layer	1 representing f_{ij}	none
Training hyperparameters		
Loss function	Eq. (9)	
Optimizer	Adaptive Moment Estimation (Adam)	
Initial Learning rate	10^{-4}	

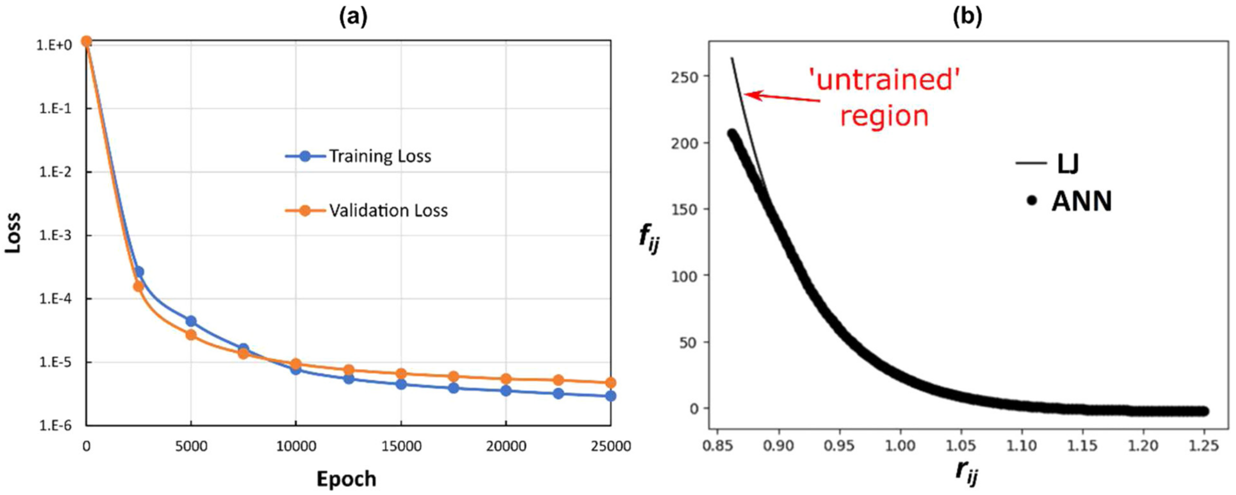


Fig. 2. (a) Loss function during training of the MD inverse problem; (b) comparison between the ANN output and the actual f_{ij} function from the LJ potential. There are not data for training the network when $r_{ij} < 0.88$.

4.5. Results, validation, and discussion

Fig. 2b shows that f_{ij}^{ANN} is a good approximant of the real inter-particle forces f_{ij} except for small r_{ij}. A system with a given temperature T^* and density ρ^* does not visit, in a finite amount of time, the entire state-space. In the original simulation ($T^* = 1$, $\rho^* = 0.8$), states corresponding to $r_{ij} < 0.88$ never occur. Therefore, there are not data available for training the network below this threshold, and $f_{ij}{}^{\text{ANN}}$ deviates from f_{ij}.

A further validation comes from running the original simulation with $f_{ij}{}^{\text{ANN}}$ instead of f_{ij}. In Fig. 3a, K is the average kinetic energy of the system, U the total energy and P the virial pressure defined as

$$P = \frac{1}{2L_x L_y N} \sum_{i}^{N} \sum_{i<j}^{N} \mathbf{f}_{ij} \cdot \mathbf{r}_{ij}, \tag{10}$$

In MD simulations particle velocities are initialized randomly based on the Maxwell–Boltzmann distribution. Therefore, we do not expect identical outputs, but statistically equivalent results (Fig. 3a). Note that U can be shifted by a constant because it is integrated from eq. (6) with respect to an arbitrary constant. In the ANN data, we use $U(r_c) = 0$, but the LJ potential is defined with respect to $U(\infty) = 0$, which explains the small difference between U_{ANN} and U_{LJ}.

4.6. Generalization of the ML model

The model was trained with a simulation carried out with $\rho^* = 0.8$ and $T^* = 1$. How generalizable is the model to other values of ρ^* and T^*? We ran several simulations at different values of ρ^* and T^* with both $f_{ij}{}^{\text{ANN}}$ and f_{ij}. Fig. 3b shows the model performance based on the average error of the virial pressure. The error is within 10% also for values of ρ^* and T^* that are significantly different from the original ones. The more the system differs from the one used for training, the more frequently it will visit the 'untrained region'. The pair force will differ from the real one, but, overall, the system always complies with eq. (1). Energy and momentum conservation, for instance, are always guaranteed in a strong sense.

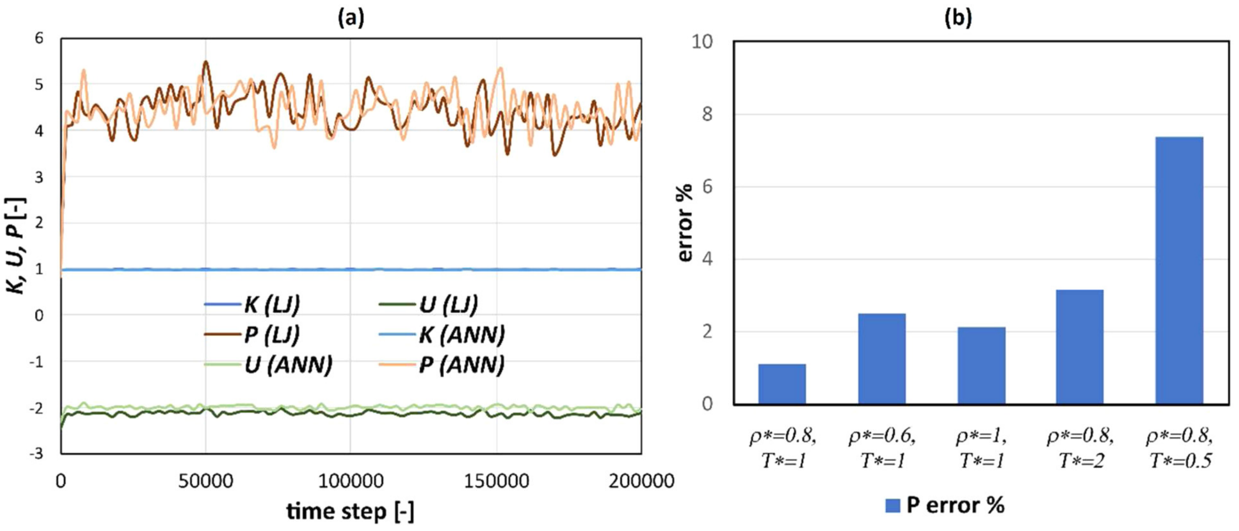


Fig. 3. (a) Comparison between simulations calculated with the LJ potential or with the ANN approximant: K is the average kinetic energy of the system, U the total energy, and P the virial pressure; (b) performance of the ANN model (percentage of error on the virial pressure), when conditions different from those of the training dataset.

5. Case study 2, fluid dynamics: continuous, non-conservative systems

The second case study refers to a classic fluid dynamics problem (dam break) simulated with SPH. The method designed in Case Study 1 is extended to dissipative systems where $\mathcal{Q}_i \neq 0$. As before, we consider conservative forces that only depend on the particle positions. But now, we also account for dissipative forces that depend on the particle velocities. The system is not ergodic; dissipation drives the system towards lower energy regions of the state-space. This implies that, unless new energy is introduced into the system, not all timesteps have the same training value. Particles will eventually settle into low energy states that do not provide additional benefit in terms of ANN training. The next section provides a brief overview of SPH; the reader can refer to Liu and Liu [25] for more details.

5.1. Smooth particle hydrodynamics background

The mechanics of fluids is described by the Navier-Stokes equation that the SPH method recasts as a particle system, where the forces acting on each particle mimic the pressure and viscosity forces occurring on computational 'fluid particles'

$$m_i \frac{d\mathbf{v}_i}{dt} = \sum_j m_i m_j \left(\mathrm{P}_{ij} + \Pi_{ij}\right) \nabla W_{ij} + \sum \mathbf{f}_E \tag{11}$$

where P_{ij} is the pressure tensor, Π_{ij} is the viscosity tensor, $\mathbf{f}_\mathrm{E}$ represents external body forces such as gravity. W is the so-called kernel. A bell-shaped function that weights the contributions of each j particle in the neighbour list based on their distance from i. In this work, we use the Lucy kernel defined as

$$W_{ij}\left(r_{ij}, h\right) = \frac{5}{\pi h^2} \begin{cases} (1+3R)\left(1-R^3\right) & R \le 1 \\ 0 & R > 1 \end{cases}, \tag{12}$$

where $R = r_{ij}/h$ and h is the so-called smoothing length that determines the radius of the neighbour list and plays a similar role to r_c in MD.

The pressure tensor is defined by

$$\mathrm{P}_{ij} = \frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \tag{13}$$

where p_i and p_j is the pressure associated respectively with particles i and j, and ρ_i and ρ_j their densities. In MD, particles possess mass m_i, position $\mathbf{r}_i$ and velocity $\mathbf{v}_i$. In SPH, particles are given additional properties such as pressure p_i and density ρ_i that depend on the position of the surrounding particles. The density is calculated as

$$\rho_i = \sum_j m_j W_{ij}. \tag{14}$$

Table 4
Dimensionless parameters scaled with respect the particle mass m and a reference length ΔL (initial distance between particles).

Dimensionless variable	Formula
Smoothing Length h^*	$h^* = \frac{h}{\Delta L}$
Time t^*	$t^* = t\sqrt{\frac{g}{\Delta L}}$
Force F^*	$F^* = \frac{F}{mg}$
Density r^*	$\rho^* = \frac{(\Delta L)^3 \rho}{m}$
Pressure p^*	$p^* = \frac{pL}{mg}$
Dynamic viscosity m^*	$\mu^* = \frac{\mu (\Delta L)^{\frac{3}{2}}}{m\sqrt{g}}$

Table 5
SPH simulation parameters in dimensionless numbers.

Width simulation box L^*x	80
Height simulation box L^*y	80
Density r_0^* (eq. (14))	1
Number of particles N	1200
Timestep Dt^*	0.005
gravity g^*	10
Smoothing length h^*	1.6
Constant c^* (eq. (14))	100
Constant α (eq. (15))	0.4

The concept of density for incompressible fluids like water can be misleading to the reader not familiar with SPH. A simpler way to interpret the SPH density is as a measure of the concentration of particles within the smoothing length. This density is used as a device to calculate the pressure forces in the fluid by means of an equation of state like

$$p_i = c^2 (\rho_i - \rho_0), \tag{15}$$

where c is a proportionality constant and ρ_0 a density reference. P$_{ij}$ represents the contribution of particle j to the total conservative forces acting on particle i (similar to f_{ij} in eq. (6)). Finally, the viscosity tensor can be defined as

$$\Pi_{ij} = -\frac{\alpha c}{\rho_i + \rho_j} \frac{\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}}{r_{ij} + \epsilon h^2}, \tag{16}$$

where $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$ is the relative velocity of the two particles, α a dimensionless factor controlling, the dissipation strength, and $\epsilon = 0.01$ a constant. Π_{ij} represents the contribution of each particle j to the total dissipation force $\mathcal{Q}_i$ in eq. (1). In principle, it should only depend on the normal (in the direction of $\mathbf{r}_{ij}$) velocity v_{ij}^n between i and j

$$v_{ij}^n = \frac{\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}}{r_{ij}}, \tag{17}$$

but the extra term ϵh^2 in eq. (16) is added to avoid singularities in the case particles get very close to each other. The viscosity tensor can be related to the real dynamic viscosity μ:

$$\mu = \frac{\alpha h c \rho_0}{8}. \tag{18}$$

There are different versions of the SPH method based on different definitions of eqs. (12)–(16). In this work, we use a 'vanilla' version of SPH; more complex versions can be found in the literature [25,7,26].

As for Case Study 1, we use dimensionless numbers (in MD, they are usually called reduced units; in fluid mechanics dimensionless numbers) (Table 4)

5.2. Simulation details

The simulation consists of a typical SPH benchmark case known as dam break: a column of water collapsing under the effect of gravity g. Details of the simulation in dimensionless numbers are given in Table 5. The simulation is run for 4,000 timesteps and the results written to the output file every 400 timesteps.

The box has reflective boundary conditions meaning that the velocity component perpendicular to the wall is simply reflected when the particle reaches the boundary. These are not realistic conditions and rarely employed in SPH. They are

Table 6
Network Architecture and Training hyperparameters for the SPH problem.

ANN architecture		
	Neurons	Activation function
Input layer	4 representing $1/r_{ij}$, v^n_{ij},ρ_i, ρ_j	–
Hidden Layer 1	8	Elu
Hidden Layer 2	16	Elu
Hidden Layer 3	8	Elu
Output layer	1 representing $(\text{P}_{ij} + \Pi_{ij})\nabla W_{ij}$	none
Training hyperparameters		
Loss function	Eq. (20)	
Optimizer	Adaptive Moment Estimation (Adam)	
Initial Learning rate	10^{-4}	

used here because they do not interfere with the forces acting on the particles. The technique for extracting boundary conditions is introduced in Case Study 3.

5.3. The minimalistic approach for solving the inverse SPH problem

Both conservatives $\text{P}_{ij}(r_{ij})$ and dissipative $\Pi_{ij}(v^n_{ij})$ forces are lumped into a single function

$$\Phi_{ij} = \left(\text{P}_{ij} + \Pi_{ij}\right)\nabla W_{ij}. \tag{19}$$

The Loss function is the same of eq. (9) with the difference that Φ_{ij} replaces f_{ij}

$$L = \sum_{t}^{N_t} \frac{\sum_i^N \left\| \mathbf{F}^{\boldsymbol{TARGET}}_{i,t} - \sum_{j\neq i}^{N_{\ell,i,t}} \Phi^{ANN}_{ij,t} \frac{\mathbf{r}_{ij,t}}{r_{ij,t}} \right\|^2}{N N_t}. \tag{20}$$

In theory, Φ_{ij} should only depend on r_{ij} and v^n_{ij} since, according to eq. (14) and (15), p and ρ are functions of r_{ij} and v^n_{ij}. However, while eq. (20) is designed for pair forces, the density formulation of eq. (14) adds a multi-body element to the force. The extension of eq. (20) to multi-body forces requires a different approach that is left for future work. Here, to preserve the pair-based structure of the loss function, we simply include ρ_i and ρ_j as separate inputs of $\Phi^{ANN}_{ij}(r_{ij}, v^n_{ij},\rho_i, \rho_j)$. In the case of SPH, this is not a problem because, usually, simulations also store particle densities. However, this issue must be addressed when extending the method beyond SPH.

5.4. Training of the ANN

The network architecture and training hyperparameters are the same of Case Study 1. The only difference is in the input layer that has 4 neurons instead of 1 (Table 6).

We train the network with 100 timesteps and 1200 particles, which corresponds to 132,000 training data. We need more data points than Case Study 1, because the system is not ergodic and requires an adequate sampling of the transient, before the system settles into a low energy state. Gravity forces are not known but learned from data and incorporated into Φ^{ANN}_{ij}. The Loss function during training is shown in Fig. 4a.

5.5. Results, validation, and discussion

Fig. 4b compares Φ^{ANN}_{ij} with the real $(\text{P}_{ij} + \Pi_{ij})\nabla W_{ij}$. The approximation is very good except for the case of high positive velocities and small r_{ij}: more details on the untrained region are given in the next section.

We also ran the original simulation with Φ^{ANN}_{ij} instead $(\text{P}_{ij} + \Pi_{ij})\nabla W_{ij}$. Fig. 5 compares the results (see also Video 1). The behaviour is very similar. There are a few differences occurring around the droplets. Statistically, there are less particles in the droplets and, therefore, the ANN has, proportionally, fewer training data in these regions. If necessary, this issue could be mitigated by giving higher statistical weights to the particles near free surfaces.

5.6. Generalization of the ML model

The Φ^{ANN}_{ij} trained with the dam break simulation (an example of free-surface flow) is tested against channel flow (Fig. 6a). The walls are represented by two frozen layers of particles on each side; the flow is generated by a gravity-like body force ($\mathbf{f}^*_{\text{E}}$=10) in the (periodic) x-direction. Fig. 6 compares the results obtained with the ANN and with the SPH

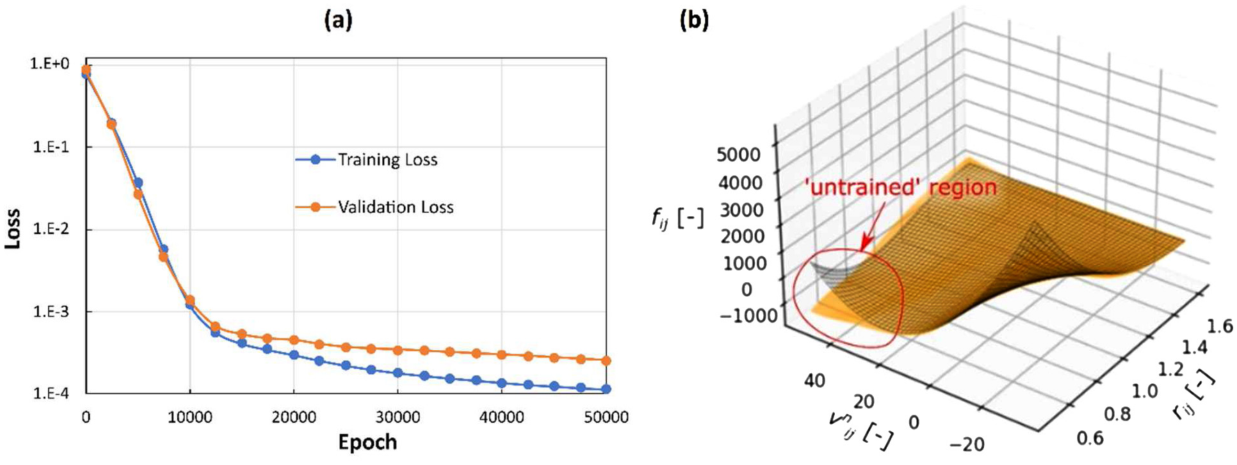


Fig. 4. (a) Loss function during training of the SPH inverse problem, (b) real pair force $(P_{ij} + \Pi_{ij})\nabla W_{ij}$ (wireframe) versus the ANN approximation Φ^{ANN}_{ij} (orange surface) at different values of v^n_{ij} and r_{ij} for the case of ρ_i = 1 and ρ_j = 1.01. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

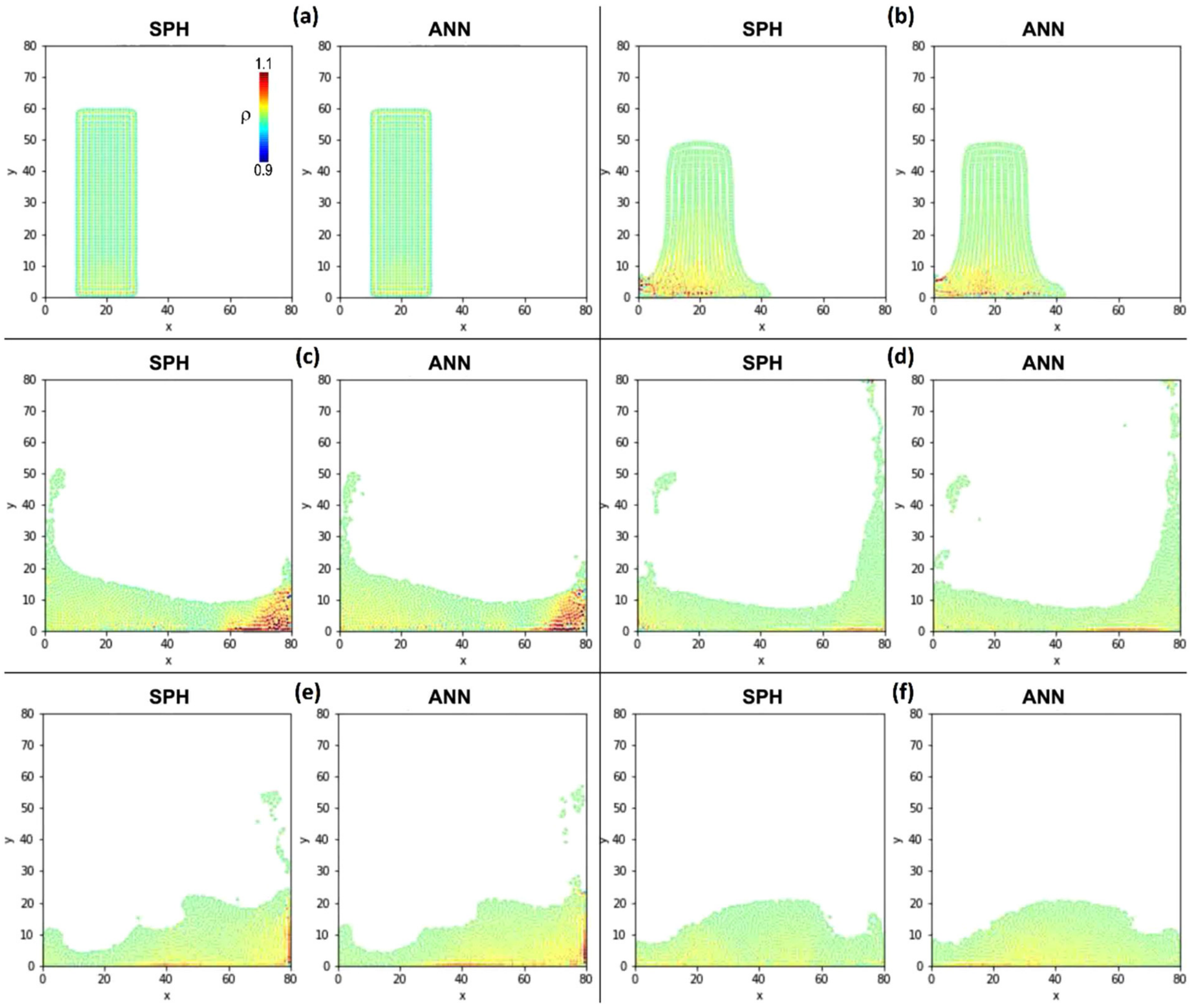


Fig. 5. Simulation of the dam break calculated with the SPH model and with the ANN model. Particles are coloured by density (see also Video 1). (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

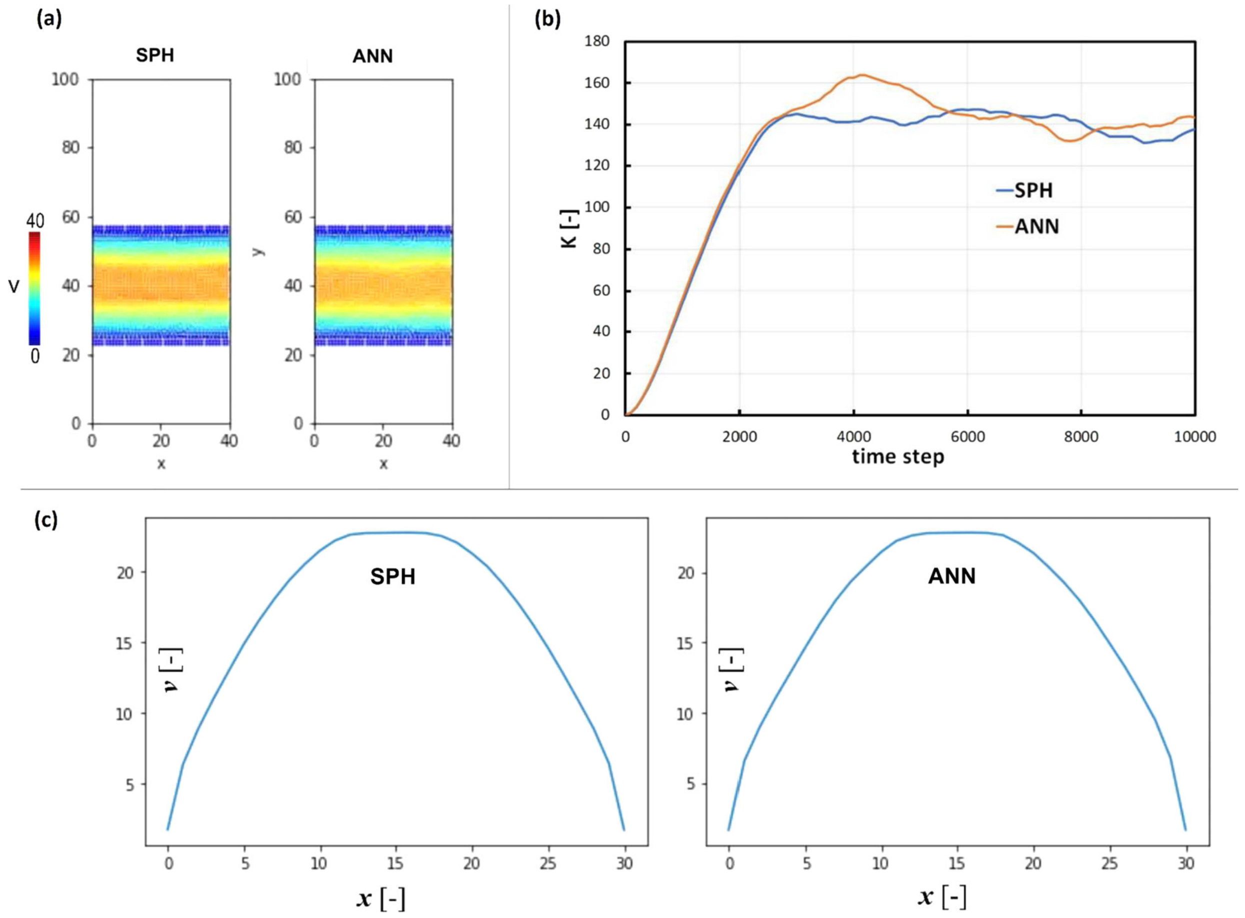


Fig. 6. (a) Comparison between ANN and SPH model in the case of channel flow (particles coloured by velocity, see also Video 2), (b) kinetic energy, (c) average velocity profile. The ANN was trained with the dam break dataset, but it replicates the SPH model also when applied to channel flow. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

model (see also Video 2). The system is completely different from the previous one in terms of geometry and boundary conditions, but the ANN trained with the dam break dataset replicates the SPH model also when applied to the channel flow. The flow is turbulent-like, with fluctuations that could depend on the fact that our 'vanilla' model is not very accurate for channel flow. However, the goal is to verify that the ANN model behaves like the SPH model: if the SPH model produces spurious fluctuations, also the ANN must produce spurious fluctuations. Due to turbulence, the flow is stochastic, but the statistics are similar for both the average kinetic energy (Fig. 6b) and the average velocity profile (Fig. 6c).

From the fluid dynamics point of view, free-surface flow and channel flow are two very different types of flows. If the proposed approach only worked as a sort of video generation algorithm, without learning the actual physics of the system, it would not correctly simulate channel flow after being trained exclusively with the dam break example (which is an example of free-surface flow). Untrained regions represent physical phenomena that do not occur in the training data. For instance, the untrained region in Fig. 4 represents a reduction in fluid pressure associated with a high increase in velocity. This phenomenon can occur, for instance, in Venturi tubes, but it does not occur in either of the examples considered. Therefore, the model trained with the free-surface flow can replicate channel flow, because the two flows share a common physics; but it cannot replicate a Venturi tube because the training data does not provide any learnable example of a phenomenon like the Venturi effect.

5.7. Noisy datasets

A systematic analysis on the effect of noise is beyond the scope of this paper. We only carry out a single comparison to make sure the approach is robust to a given level of noise. We add 10% white noise (i.e. the original signal was stochastically incremented of ±10% based on a flat distribution within this range) to the training dataset, train for 50,000 epochs and compared the final losses calculated with and without noise. Noise is only added to the training dataset not to the validation dataset. In this way, we train the ANN with the noisy data and validate it against pristine data. The final loss for the pristine dataset is $1.15\cdot10^{-4}$ for training and $2.55\cdot10^{-4}$ for validation; for the noisy dataset $9.98\cdot10^{-3}$ for training and $2.97\cdot10^{-4}$ for

Table 7
Dimensionless DEM parameters scaled with respect the particle mass m and radius R.

Dimensionless variable	Formula
Length r^*	$r^* = \frac{r}{R}$
Time t^*	$t^* = t\sqrt{\frac{g}{R}}$
Force F^*	$F^* = \frac{F}{mg}$
Elastic coefficient k^*	$k^* = \frac{kR^2}{mg}$
Damping coefficient g^*	$\gamma^* = \gamma\sqrt{\frac{R}{g}}$

validation. The training loss is higher because of the noise, but the validation loss is similar showing that the method can tolerate a relatively high level of noise.

6. Case 3 granular mechanics: non-point-particles systems with unknown boundary conditions

The third case study refers to a granular mechanics problem simulated with DEM. In the previous two case studies, we assumed that the boundary conditions were known and there was no need to extract them from data. Case Study 3 shows how the methodology can be extended to include boundary conditions. The next section provides a brief overview of DEM; the reader can refer to Seville and Wu [27] for more details.

6.1. Discrete element method background

Differently from MD and SPH, DEM particles are not point-particles. They have a physical radius and interact only when their physical distance is lower than the sum of their radii. In our example, all particles have the same radius R. Collision occurs when their overlap

$$\delta_{ij} = 2R - \left\|\mathbf{r}_{ij}\right\| \tag{21}$$

is positive. This implies that the contact forces between two particles i and j can be expressed by

$$\mathbf{f}_{ij} = \begin{cases} \left(f_{ij}^{Hertz} + f_{ij}^{damp}\right)\dfrac{\mathbf{r}_{ij}}{\left\|\mathbf{r}_{ij}\right\|}, & \delta_{ij} > 0 \\ 0, & \delta_{ij} \le 0 \end{cases} \tag{22}$$

where f_{ij}^{Hertz} are conservative forces and f_{ij}^{damp} dissipative forces between two colliding particles. According to Hertz theory, conservatives (elastic) forces can be expressed by

$$f_{ij}^{Hertz} = k\sqrt{R}\delta_{ij}^{3/2}, \tag{23}$$

where k is the stiffness of the material. Dissipative (damping) forces can be expressed as

$$f_{ij}^{damp} = -\gamma m \delta_{ij}^{1/2} v_{ij}^{n}, \tag{24}$$

where γ is the viscoelastic constant.

For non-point particles, we should consider rotation. The main goal of this section is to explain how boundary conditions are accounted for in the inverse problem. Therefore, for the sake of simplicity, we use a simplified model where rotation is neglected. As before, we use dimensionless variables (Table 7).

Boundary conditions are expressed in a similar way to eq. (22). When the distance between a particle and the walls is less than R, the particle collides with the wall. In this case, the boundary b exchanges with particle i both conservatives f_{ib}^{Hertz} and dissipative f_{ib}^{damp} forces, whose equations are the same of eq. (23) and (24), but with different values of k and γ.

6.2. Simulation details

The simulation consists of 30 particles placed in the middle of the computational box with random velocity and falling due to gravitational acceleration g. Details of the simulation in reduced units are given in Table 8. The simulation runs for 6,000 timesteps and the results written to the output file every timestep. This time, all simulation output data are used for training; this is important for the training phase as explained later.

Table 8
DEM simulation parameters in dimensionless units.

Width simulation box L^*x	20
Height simulation box L^*y	20
Elastic constant k^*	1,000
Damping coefficient g^*	10
Wall elastic constant k^W	10,000
Wall damping coefficient g^W	1,000
Number of balls N	30
Timestep $Dt*$	0.001
Gravity g^*	10

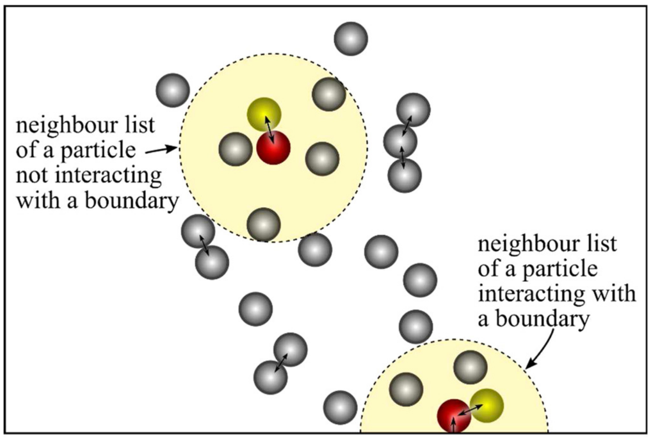

neighbour list of a particle not interacting with a boundary
neighbour list of a particle interacting with a boundary

Fig. 7. Particles that do not interact with the boundaries experience only particle-particle interactions. Particles that interact with the boundaries experience both particle-particle and particle-wall interactions.

6.3. *The minimalistic approach for solving the DEM inverse problem*

The ANN must learn the behaviour of both the particle-particle interactions and the particle-wall interactions, we call Φ^{ANN} the first and Φ^{WANN} the latter. As Fig. 7 shows, particles that do not interact with the boundaries experience only particle-particle interactions, while particles that interact with the boundaries experience both particle-particle and particle-wall interactions.

The solving strategy is to divide the training in two phases. During the first phase, we train the ANN only with data from 'bulk neighbour lists' that do not interact with the walls. During the second phase, we train another ANN only with data from 'boundary neighbour lists' that interact with the walls.

Phase 1 (*ANN model for the bulk*)

The Loss function is almost the same of eq. (20)

$$L = \sum_{t}^{N_t} \frac{\sum_i^{N-B} \left\| \mathbf{F}_{i,t}^{TARGET} - \sum_{j \neq i}^{N_{\ell,i,t}} \Phi_{ij,t}^{ANN} \frac{\mathbf{r}_{ij,t}}{r_{ij,t}} \right\|^2}{NN_t}, \tag{25}$$

where $\Phi_{ij}^{ANN}(r_{ij}, v_{ij}^n) = f_{ij}^{Hertz} + f_{ij}^{damp}$, and B is the number of particles that interacts with the boundaries. The only difference between eq. (20) and eq. (25) is that the calculation now is carried out only for the $N - B$ bulk neighbour lists.

Phase 2 (*WANN: ANN model for the Wall*)

After Φ^{ANN} is known, a second ANN $\Phi_{ib}^{WANN}(r_{ib}, v_{ib}^n) = f_{ib}^{Hertz} + f_{ib}^{damp}$ is trained to approximate the interaction between particle i and boundary b. In our simulation, all four walls behave in the same way, but if there were different types of boundaries, we would need a different WANN for each type of boundary. The WANN is fed only with boundary neighbour lists. The Loss function becomes

$$L = \sum_{t}^{N_t} \frac{\sum_i^{B} \left\| \mathbf{F}_{i,t}^{TARGET} - \sum_{j \neq i}^{N_{\ell,i,t}} \Phi_{ij,t}^{ANN} \frac{\mathbf{r}_{ij,t}}{r_{ij,t}} - \sum_{b}^{N_{b,i,t}} \Phi_{ib,t}^{WANN} \mathbf{n}_{ib,t} \right\|^2}{NN_t}, \tag{26}$$

where $N_{b,i,t}$ is the number of boundaries touched by i, and $\mathbf{n}_{ib,t}$ the vector normal to that boundary.

Table 9
Network Architecture and Training hyperparameters for the DEM problem.

ANN architecture		
	Neurons	Activation function
Input layer	2 representing r_{ij}, v_{ij}^{n}	–
Hidden Layer 1	8	Elu
Hidden Layer 2	16	Elu
Hidden Layer 3	8	Elu
Output layer	1 representing $f_{ij}^{Hertz}+f_{ij}^{damp}$	none
Loss function	Eq. (25)	
WANN architecture		
	Neurons	Activation function
Input layer	2 representing r_{ib}, v_{ib}^{n}	–
Hidden Layer 1	8	Elu
Hidden Layer 2	16	Elu
Hidden Layer 3	8	Elu
Output layer	1 representing $f_{ib}^{Hertz}+f_{ib}^{damp}$	None
Loss function	Eq. (26)	
Training hyperparameters		
Optimizer	Adaptive Moment Estimation (Adam)	
Initial Learning rate	10^{-4}	

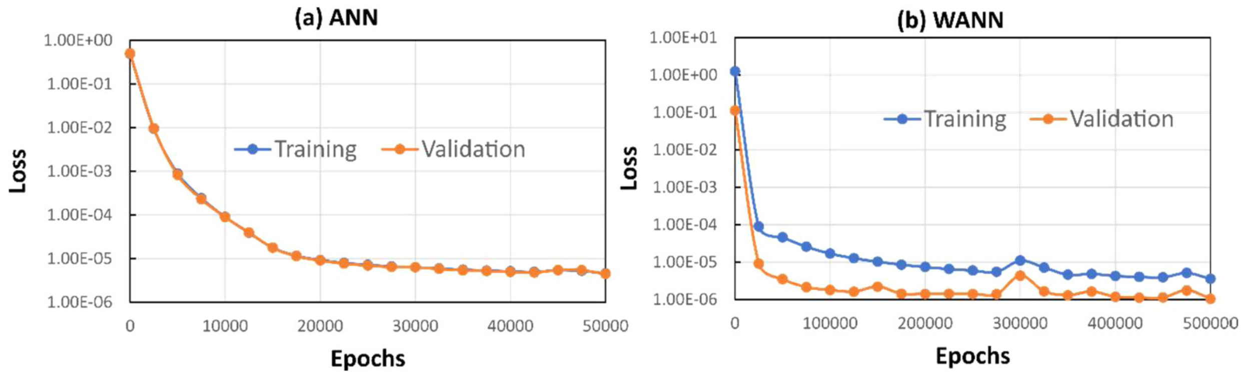


Fig. 8. (a) Loss function of the ANN (bulk), and (b) the WANN (wall).

6.4. Training the ANN/WANN

Network architectures and Training hyperparameters are shown in Table 9.

The Loss function during training is shown in Fig. 8. Because this time we use all timesteps, the dataset covers abundantly the whole accessible part of the state-space. The training and validation data are statistically equivalent and the training and validation loss in Fig. 8a almost overlap. The same does not occur at the boundaries (Fig. 8b), for reasons explained in the next section.

6.5. Results, validation, and discussion

Fig. 9 compares the physics-based simulation with (i) the results obtained using Φ^{ANN} and the real boundary conditions, and (ii) the result obtained using both Φ^{ANN} and Φ^{WANN} (see also Video 3). Results are almost identical up to the very end of the simulation. Particle systems tend to be chaotic and, therefore, small differences will, sooner or later, separate the trajectories of two systems. Therefore, the statistical behaviour is more important that the actual trajectory. Fig. 9e, for instance, shows the average kinetic energy is very close for all three systems.

To further assess the results, we can compare $\Phi_{ij}^{ANN}(r_{ij}, v_{ij}^{n})$ with the real $f_{ij}^{Hertz}+f_{ij}^{damp}$ (Fig. 10a) and $\Phi_{ij}^{WANN}(r_{ij}, v_{ij}^{n})$ with the real $f_{ib}^{Hertz}+f_{ib}^{damp}$ (Fig. 10b).

For simplicity, results are shown with respect of the overlap δ_{ij} rather than r_{ij}. The ANN does not know the physical radius of the particles and autonomously understands that there are no forces for values of r_{ij} corresponding to $\delta_{ij} < 0$. We

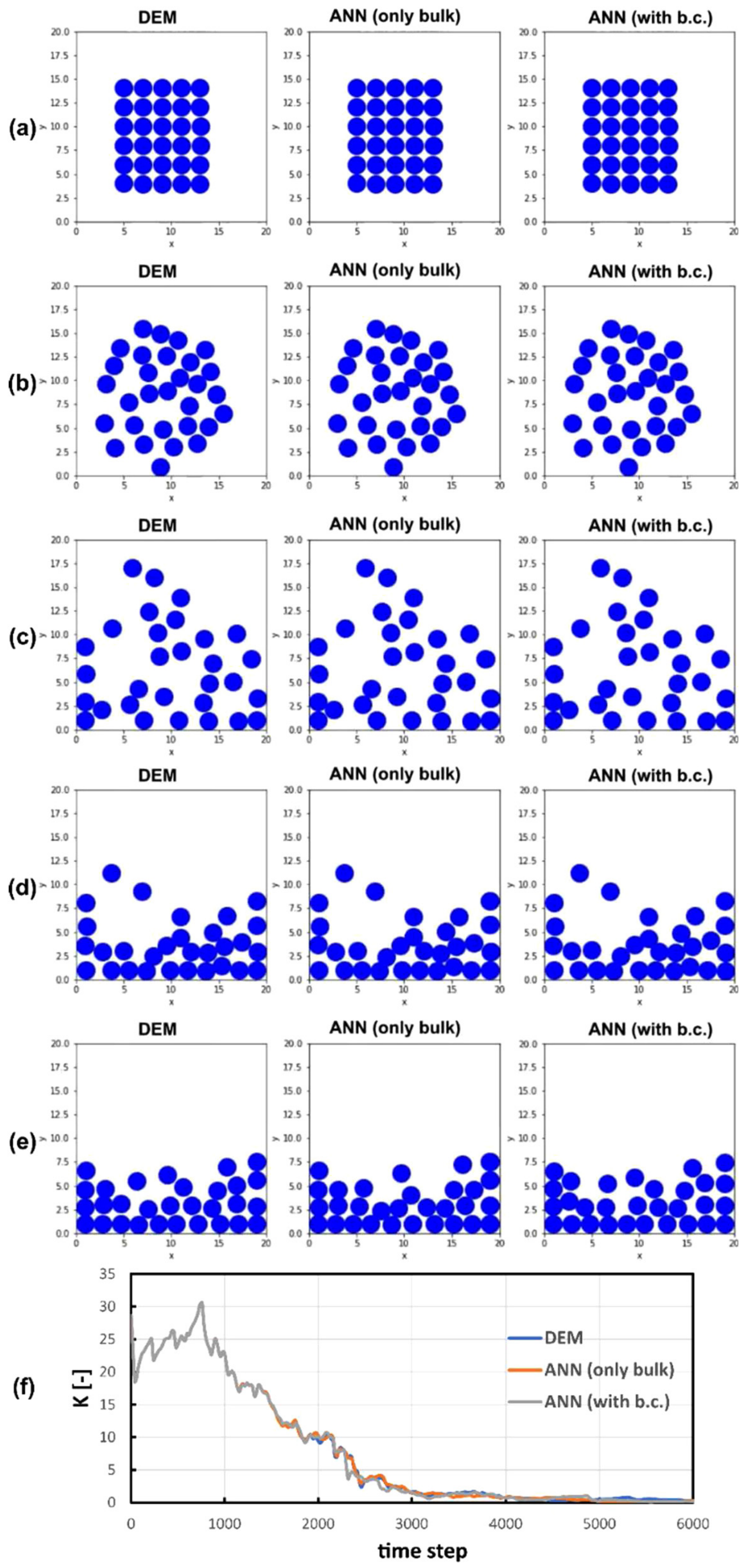


Fig. 9. (a-e) Granular simulation calculated with the DEM model, the Φ^{ANN} model and the Φ^{ANN} + Φ^{ANN} models (see also Video 3); (f) average Kinetic energy of the system. Initial velocities are randomly generated but the same for all cases.

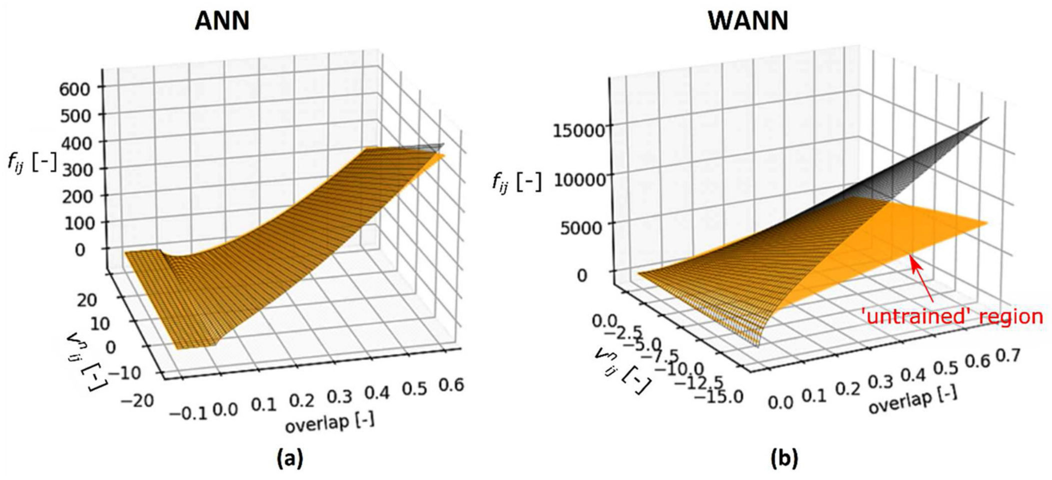


Fig. 10. Comparison between real forces and their ANN (a) and WANN (b) approximants.

have a good sampling of the bulk data, and the only untrained region occurs at high overlaps and high absolute values of the velocity (Fig. 10a). Because we have considerably less wall-particle interactions than particle-particle interactions, the WANN state-space is not sampled as thoroughly (Fig. 10b). The untrained region corresponds to scenarios (high negative velocities and high overlaps) that never occur during the simulation and have little physical relevance. In fact, high overlaps occur when the particle begins to bounce back after hitting the wall. But, when this occurs, the velocity cannot be high because the particle has been already decelerated by the wall.

7. Conclusions

This study proposes a 'minimalist' hybrid physics-ML approach that combines particle mechanics with Neural Networks. It is called 'minimalist' since, by using data structures such as neighbour lists, it minimizes the ANN both in terms of complexity and number of input variables. This brings two advantages: (i) the ANN is more easily trainable than in other methods, and (ii) the model is highly generalizable and can predict systems with geometries and boundary conditions very different from those of the training set.

A question worth asking is why this approach works. Why the minimalistic method achieves, with simple feedforward networks, what other methods achieve with ad hoc, and increasingly complex network architectures?

The answer lies in the sequence of steps adopted by the minimalistic method to incorporate the physics into the ANN. For the sake of clarity, these steps are summarised below.

The starting point is a particle system $\mathbf{x}(t)$ that evolves by means of a physical model φ

$$\mathbf{x}(t) \xrightarrow{\varphi} \mathbf{x}(t+dt)\,, \tag{27}$$

which is unknown and must be derived from data. The naïve approach would be to model φ directly. If we use an ANN (Fig. 11a), the input size of the network should match N, the degrees of freedom of the system (which can easily be of the order of 10^6 or higher).

We can improve the naïve approach by considering that all particle systems (from molecules to galaxies) must obey an equation of motion. The difference between one system and another lies in the forces $-\nabla V$ and $\mathcal{Q}_i$ exchanged by the particles. Therefore, if we account for the equation of motion, instead of approximating φ, we can learn from data only $-\nabla V$ and $\mathcal{Q}_i$: all physical principles underlying the equation of motion will be automatically satisfied.

Moreover, if we subdivide the particles in neighbour lists, in each neighbour list, the physics must be the same. Therefore, the network does not need to model the whole system, but only the neighbour list (the 'minimal ANN' in Fig. 11b). Consequently, the size of the input layer goes from N ($\sim 10^6$) to N_ℓ (number particles in a neighbour list, usually $\sim$20–30).

Finally, if the forces acting on each particle can be formulated as pairwise interactions, each pairwise force f_{ij} must follow the same model. Therefore, the input size of the minimal ANN can be further reduced from N_ℓ to just 2 (r_{ij} and v_{ij}^n). However, at this point there is a mismatch between the minimal ANN and the level of training (Fig. 11c). The ANN models the pairwise forces f_{ij}, but the training data only provide F_i. Resolving this problem requires a tweak to the loss function as shown in eq. (8).

The minimalistic method is here applied to inverse problems because they provide a direct way to validate the results, but the same ideas can be extended to other problems. For instance, it can be used to design corrective terms that improve the accuracy of low-resolution simulations speeding up calculations (in this study, the focus is on solving the inverse problem not on speed). Alternatively, the inverse problem could be applied to experimental data. For instance, several experimental techniques in fluid dynamics such as Particle Tracking Velocimetry (PTV), Particle Image Velocimetry (PIV) or

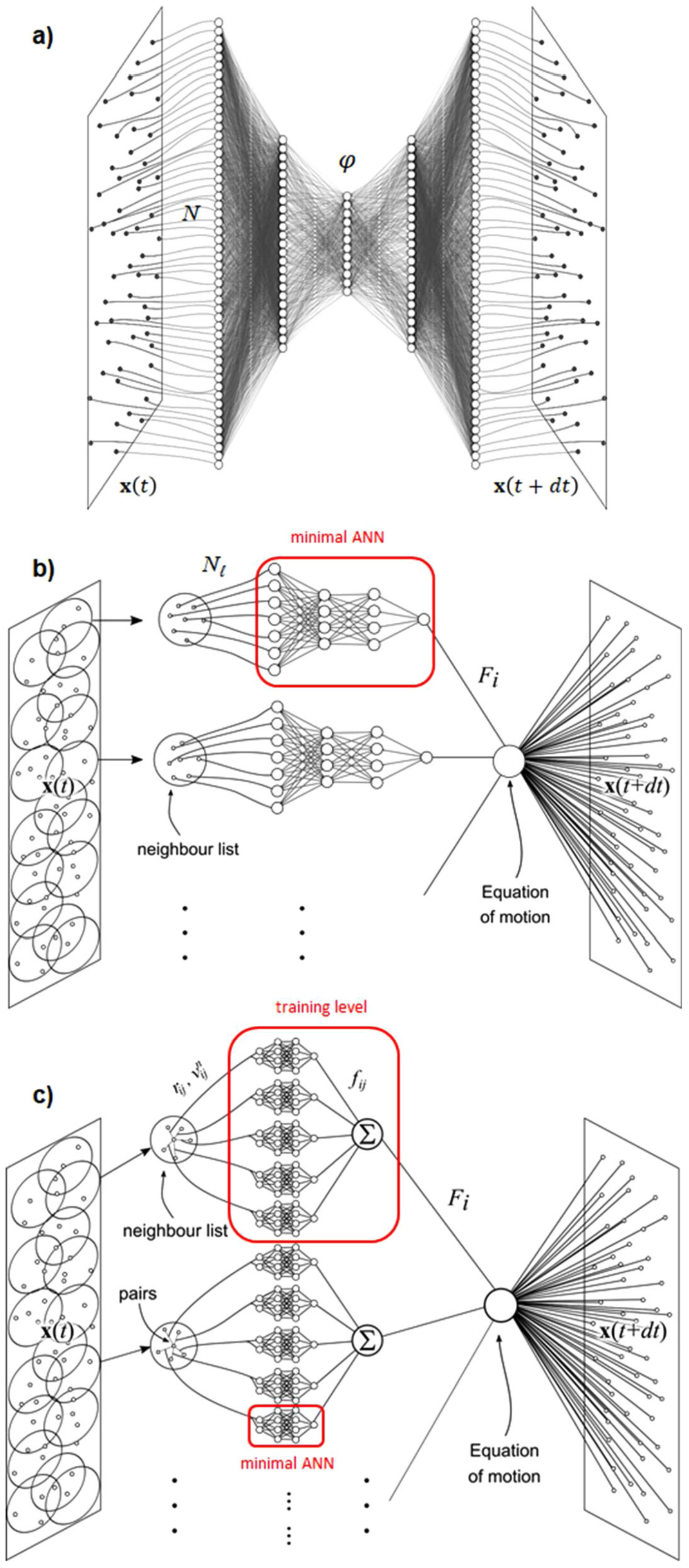


Fig. 11. Structure graphs showing (a) the naïve approach, (b) the minimal ANN based on neighbour lists, and (c) the minimal ANN based on neighbour list and pairwise interactions.

Ghost Particle Velocimetry (GPV) use tracer particles to seed the fluid, whose trajectory is used to calculate the fluid velocity [30–33]. Accelerations (which are numerical equivalent to forces if we use reduced units) can be calculated from the measured instantaneous velocities (e.g. [34], [35]). The dataset will forcibly include some level of noise due to experimental errors, but, in principle, these real particles can be handled as the computational particles of our case studies.

Finally, the minimalistic approach consolidates the 'special relationship' that particle methods and ANN seem to enjoy. In fact, these two algorithms work particularly well together to the point that a true particle-neuron duality has been suggested [36], [37].

CRediT authorship contribution statement

The author confirms sole responsibility for the following: study conception and design, methodology, software, validation, data curation, analysis and interpretation of results, and manuscript preparation.

Declaration of competing interest

The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data accessibility

The code used for this study is freely available under the GNU General Public License v3 and can be downloaded from the University of Birmingham repository edata.bham.ac.uk/744/. All code was written in Python using Numpy, PyTorch and Numba libraries. The code for the MD simulation was adapted from Rapaport and Rapaport [28]; the code for SPH from Liu and Liu [25]; the code for DEM from Pöschel and Schwager [29].

Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) grant no. EP/S019227/1.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jcp.2022.111750.

References

[1] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, et al., Physics-informed machine learning, Nat. Rev. Phys. 3 (2021) 422–440.
[2] J. Willard, et al., Integrating physics-based modeling with machine learning: a survey, arXiv:2003.04919, 2020.
[3] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[4] D. Zhang, L. Guo, G.E. Karniadakis, Learning in modal space: solving time-dependent stochastic PDEs using physics-informed neural networks, SIAM J. Sci. Comput. 42 (2020) A639–A665.
[5] Sanchez-Gonzalez, et al., Learning to simulate complex physics with graph networks, arXiv:2002.09405, 2020.
[6] A. Karpatne, W. Watkins, J. Read, V. Kumar, Physics-guided neural networks (PGNN): an application in lake temperature modeling, arXiv:1710.11431, 2017.
[7] M.S. Shadloo, G. Oger, D.L. Le Touze, Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: motivations, current state, and challenges, Comput. Fluids 136 (2016) 11–34.
[8] A. de Vaucorbeil, V.P. Nguyen, S. Sinaie, J.Y. Wu, Material point method after 25 years: theory, implementation, and applications, Adv. Appl. Mech. 53 (2020) 185–398.
[9] H. Goldstein, C. Poole, J. Safko, Classical Mechanics, 3rd edition, Addison Wesley, Boston, 2002.
[10] S. Greydanus, M. Dzamba, J. Yosinski, Hamiltonian neural networks, in: Proceedings of the 33rd International Conference on Neural Information Processing Systems, vol. 1378, 2019, pp. 15379–15389.
[11] P. Jin, Z. Zhang, A. Zhu, Y. Tang, G.E. Karniadakis, SympNets: intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems, Neural Netw. 132 (2020) 166–179.
[12] M. Lutter, C. Ritter, J. Peters, Deep Lagrangian networks: using physics as model prior for deep learning, preprint, arXiv:1907.04490, 2019.
[13] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, S. Ho, Lagrangian neural networks, preprint, arXiv:2003.04630, 2020.
[14] L. Zhang, J. Han, H. Wang, R. Car, W. E, Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics, Phys. Rev. Lett. 120 (2018) 143001.
[15] J. Behler, M. Parrinello, Generalized neural-network representation of high-dimensional potential-energy surfaces, Phys. Rev. Lett. 98 (2007) 146401.
[16] S. Chmiela, A. Tkatchenko, H.E. Sauceda, I. Poltavsky, K.T. Schütt, K.R. Müller, Machine learning of accurate energy-conserving molecular force fields, Sci. Adv. 3 (2017) e1603015.
[17] M. Fey, J.E. Lenssen, F. Weichert, H. Müller, SplineCNN: fast geometric deep learning with continuous B-spline kernels, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 869–877.
[18] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, preprint, arXiv:1609.02907 [abs], 2016.
[19] N. Trask, R.G. Patel, B.J. Gross, P.J. Atzberger, GMLS-nets: a framework for learning from unstructured data, preprint, arXiv:1909.05371, 2019.
[20] Y. Xu, T. Fan, M. Xu, L. Zeng, Y. Qiao, SpiderCNN: deep learning on point sets with parameterized convolutional filters, in: Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, Yair Weiss (Eds.), Computer Vision – ECCV 2018, Springer, Cham, 2018.
[21] L. Verlet, Computer 'experiments' on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, Phys. Rev. 159 (1967) 98–103.

[22] A. Alexiadis, Deep multiphysics: coupling discrete multiphysics with machine learning to attain self-learning in-silico models replicating human physiology, Artif. Intell. Med. 98 (2019) 27–34.

[23] A. Alexiadis, M.J.H. Simmons, K. Stamatopoulos, H.K. Batchelor, I. Moulitsas, The virtual physiological human gets nerves! How to account for the action of the nervous system in multiphysics simulations of human organs, J. R. Soc. Interface 18 (2021) 20201024.

[24] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids, Oxford University Press, 2017.

[25] G.R. Liu, M.B. Liu, Smoothed Particle Hydrodynamics: A Meshfree Particle Method, World Scientific, 2003.

[26] K.N. Ching, L.K. Tan, W.Y. Tey, Meshless fluid structural interaction (FSI) simulation of deformation of flexible structure due to water dam break, J. Adv. Res. Fluid Mech. Therm. Sci. 71 (2020) 21–27.

[27] J. Seville, C.Y. Wu, Particle Technology and Engineering: An Engineer's Guide to Particles and Powders: Fundamentals and Computational Approaches, Butterworth-Heinemann, 2016.

[28] D.C. Rapaport, R.D.C. Rapaport, The Art of Molecular Dynamics Simulation, Cambridge University Press, Italy, 2004.

[29] T. Pöschel, T. Schwager, Computational Granular Dynamics: Models and Algorithms, Springer, Germany, 2005.

[30] T. Pirbodaghi, D. Vigolo, S. Akbari, A. DeMello, Investigating the fluid dynamics of rapid processes within microfluidic devices using bright-field microscopy, Lab Chip 15 (2015) 2140–2144.

[31] M. Riccomi, F. Alberini, E. Brunazzi, D. Vigolo, Ghost particle velocimetry as an alternative to μPIV for micro/milli-fluidic devices, Chem. Eng. Res. Des. 133 (2018) 183–194.

[32] Z. Schofield, H.A. Baksamawi, J. Campos, et al., The role of valve stiffness in the insurgence of deep vein thrombosis, Commun. Mater. 1 (2020) 65.

[33] M.G. Romano, F. Alberini, L. Liu, et al., Development and application of 3D-PTV measurements to lab-scale stirred vessel flows, Chem. Eng. Res. Des. 172 (2021) 71–83.

[34] A. Jensen, G.K. Pedersen, Optimization of acceleration measurements using PIV, Meas. Sci. Technol. 15 (11) (2004) 2275.

[35] N. Machicoane, P.D. Huck, A. Clark, A. Aliseda, R. Volk, M. Bourgoin, Recent developments in particle tracking diagnostics for turbulence research, in: F. Toschi, M. Sega (Eds.), Flowing Matter, in: Soft and Biological Matter, Springer, Cham, 2019.

[36] A. Alexiadis, Deep multiphysics and particle–neuron duality: a computational framework coupling (discrete) multiphysics and deep learning, J. Appl. Sci. 9 (2019) 5369.

[37] A. Alexiadis, M.J.H. Simmons, K. Stamatopoulos, et al., The duality between particle methods and artificial neural networks, Sci. Rep. 10 (2020) 16247.