Powder Technology 448 (2024) 120258

Contents lists available at ScienceDirect

Powder Technology

journal homepage: www.journals.elsevier.com/powder-technology

A discrete element solution method embedded within a Neural Network

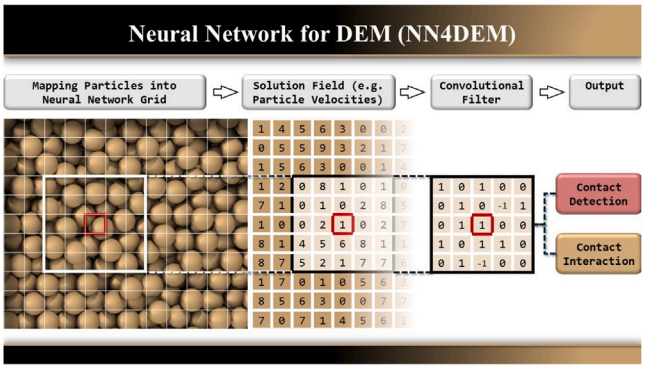
Sadjad Naderi [a], Boyang Chen [a,*], Tongan Yang [b], Jiansheng Xiang [a], Claire E. Heaney [a,c], John-Paul Latham [a], Yanghua Wang [b], Christopher C. Pain [a,c,d,*]

[a] Applied Modelling and Computation Group, Department of Earth Science and Engineering, Imperial College London, London, SW7 2AZ, UK
[b] Resource Geophysics Academy, Department of Earth Science and Engineering, Imperial College London, London, SW7 2AZ, UK
[c] Centre for AI-Physics Modelling, Imperial-X, White City Campus, Imperial College London, London, W12 7SL, UK
[d] Data Assimilation Laboratory, Data Science Institute, Imperial College London, London, SW7 2AZ, UK

HIGHLIGHTS

- A novel integration: neural networks with DEM principles.
- Physical laws dictate network weights.
- Neural Network solver benefits without training data.
- GPU parallelisation with PyTorch.
- Random packing demonstrates computational capacity.

GRAPHICAL ABSTRACT



ARTICLE INFO

Dataset link: https://github.com/bc1chen/NN4DEM

Keywords:
Discrete Element Method
Neural network
Partial differential equations
GPU-accelerated computing
Large computations
AI

ABSTRACT

This paper introduces a novel methodology, the Neural Network framework for the Discrete Element Method (NN4DEM), as part of a broader initiative to harness specialised AI hardware and software environments, marking a transition from traditional computational physics programming approaches. NN4DEM enables GPU-parallelised computations by mapping particle data (coordinates and velocities) onto uniform grids as solution fields and computing contact forces by applying mathematical operations that can be found in convolutional neural networks (CNN). Essentially, this framework transforms a DEM problem into a series of layered "images" composed of pixels, using stencil operations to compute the DEM physics, which is inherently local. The method revolves around custom kernels, with operations prescribed by the laws of physics for contact detection and interaction. Therefore, unlike conventional AI methods, it eliminates the need for training data to determine network weights. NN4DEM utilises libraries such as PyTorch for relatively easier programmability and platform interoperability. This paper presents the theoretical foundations, implementation and validation of NN4DEM through hopper test benchmarks. An analysis of the results from random packing cases highlights the ability of NN4DEM to scale to 3D models with millions of particles. The paper concludes with potential research directions, including further integration with other physics-based models and applications across various multidisciplinary fields.

* Corresponding authors.
E-mail addresses: boyang.chen16@imperial.ac.uk (B. Chen), c.pain@imperial.ac.uk (C.C. Pain).

https://doi.org/10.1016/j.powtec.2024.120258
Received 15 April 2024; Received in revised form 30 August 2024; Accepted 5 September 2024
Available online 17 September 2024

1. Introduction

The Discrete Element Method (DEM) serves as a powerful tool for analysing particle-based phenomena across a wide array of applications, encompassing granular flow [1,2], powder mechanics [3,4], rock mechanics [5], molecular dynamics [6] and astrophysical processes [7]. Despite its potential and longstanding development history, the widespread adoption of DEM simulations is hindered by computational limitations. As particle systems increase in scale and complexity, conventional DEM methodologies face difficulties in maintaining efficiency, often necessitating specialised hardware and long simulation times. This computational bottleneck profoundly inhibits the practical utility of DEM, constraining its capacity to tackle real-world problems with the requisite precision and complexity. These limitations manifest across various facets of DEM application.

Firstly, traditional DEM algorithms exhibit quadratic scaling with the number of particles, resulting in a rapid escalation of computational expense as the system size increases. For simulations comprising millions or even billions of particles, conventional methods become impractical, if not unfeasible, to execute using standard computing hardware [8]. Secondly, DEM simulations necessitate the storage and continual updating of information pertaining to each individual particle, including parameters such as position, velocity and contact forces with neighbouring particles. This requirement imposes a substantial memory overhead, which can swiftly exceed the capabilities of even high-performance workstations when dealing with large-scale systems [9]. Thirdly, the computational overhead associated with contact detection and force calculations during particle collisions adds to the overall simulation time. This issue is particularly pronounced in systems characterised by highly dynamic interactions and frequent collisions, exacerbating the computational burden [10].

The constraints outlined above limit the range of problems that can be effectively addressed by DEM, particularly when integrating it with other physics, such as Computational Fluid Dynamics (CFD), which introduces additional complexities and escalates further the computational demands [11–13]. The pursuit of efficient parallelisation techniques for DEM simulations, based on CPU and GPU-accelerated methods, has been a focal point for years [8,14–18]. However, overcoming this challenge remains difficult due to the complex nature of particle interactions and dependencies [1,8,19]. Additionally, while existing parallelisation techniques yield performance enhancements, their applicability to new scenarios may be limited, necessitating extensive calibration efforts. The development of frameworks that are both generalisable and easily validated across diverse applications poses a critical barrier. For multi-physics applications, DEM is often coupled with other computational methods such as Finite Element Analysis (FEA) [20,21] and the Lattice Boltzmann Method [22,23]. However, existing DEM implementations may not integrate well with these tools, impeding collaborative workflows for complex multi-physics simulations. This bottleneck not only obstructs performance optimisation and code implementation but also undermines scalability, thereby further complicating the computational landscape.

To address these challenges effectively, and mitigate concerns regarding cross-platform compatibility and the utilisation of state-of-the-art energy-efficient computing systems, we propose the adoption of a pioneering methodology: Neural Networks for DEM (NN4DEM). This innovative approach represents a paradigm shift in computational techniques by integrating artificial neural networks (ANN) with fundamental principles of physics, providing a modern solution to complex computational problems. NN4DEM capitalises on the many advantages offered by ANN-based solvers, including platform agnosticism, as the code can run on CPUs, GPUs and the next generation AI processors; increased computational capacity; enhanced programmability; straightforward coupling for modelling multi-physics systems, as neural networks representing different physical processes can be easily joined together; compatibility with surrogate models based on trained neural networks; and fully differentiable models, which enable optimisation processes such as data assimilation, uncertainty quantification and inversion. One example of integrating physics-based and AI models incorporates a discretised system within an inversion problem. As the physics-based model produced by NN4PDEs is fully differentiable, the inversion problem can be solved easily using the AI tools associated with backpropagation. However, if the discretised system were to be a Fortran/CUDA model, this model would not be differentiable and additional work would be required to perform the inversion. In all but the simplest data-driven applications, our (differentiable) model will have advantages over a traditional (non-differentiable) Fortran/CUDA approach.

In NN4DEM, the weights of the convolutional layers within a Convolutional Neural Network (CNN) are determined using physics-based principles, rather than relying on training data to establish relationships and patterns. NN4DEM enables the construction of neural networks that embody fundamental physical laws (albeit through their discretisations), meaning that the weights of the neural network are shaped by centuries of scientific understanding. This contrasts with traditional use of NNs in existing work [24–28] to accelerate DEM simulations, where some parts of the simulations are approximated by a trained NN, resulting in a data-driven model. For example, Lu et al. [24] trained a CNN on DEM-generated images to derive position corrections for contact detection between particles and boundaries. The use of physics-based principles to determine the weights of the neural network rather than training, represents the fundamental difference between our proposed approach and other ANN-based methods. Although, as for traditional methods, NN4DEM can also be used to generate training datasets for applications that could benefit from data-driven models. The idea of representing a discretised system of equations as a neural network with pre-determined weights can be seen in recent literature. Zhao et al. [29] presented a method which expressed a discretised system of PDEs as a neural network whose weights (or filters) were specified by choice of finite difference scheme. Later, Wang et al. [30], who also applied this idea to finite difference schemes, demonstrated good strong and weak scaling on Tensor Processing Units for some benchmarks in fluid dynamics. Chen et al. [31] extended this idea to finite element discretisations and introduced a neural network with a U-Net architecture in order to represent a multigrid solver, therefore expressing and solving the discretised system entirely with neural networks. Referred to as NN4PDEs (Neural Networks for PDEs), the method in [31] has since been developed to model neutron diffusion [32], multiphase flow [33] and flooding [34], and has been extended to model higher-order finite-element discretisations [35] and to use unstructured meshes [36]. The method has also been integrated with trained neural networks for a geological inversion problem, where a trained generative neural network suggests prior conductivity fields and a forward model, written as a neural network, makes predictions for the potential field [37]. Using backpropagation functions from machine learning libraries, the mismatch between observations of potential and the prediction of the forward model is minimised with respect to the hidden variables of the network that generates the prior. This demonstrates seamless integration of trained and untrained neural networks.

This study proposes the adoption and adaption of the NN4PDEs approach, thus far applied to PDEs [31–36]. Inspired by the inherent analogy present in AI grid-based calculations, our aim is to simulate particle dynamics and interactions within a unified computational framework established on the PyTorch environment. This paper presents the implementation of NN4DEM and demonstrates its effectiveness through benchmark cases, including a particle packing problem and a hopper test. The structure of the paper is as follows: Firstly, the background is provided to elucidate the fundamental aspects of the physical phenomena and governing equations used in DEM, as well as particle packing simulations. Subsequently, the paper describes the implementation of NN4DEM, detailing the methodology and techniques employed. Following this, the paper presents and discusses the results obtained from the

benchmark case studies, highlighting the performance and capabilities of NN4DEM. Finally, concluding remarks are provided, summarising the key findings and implications of the study, along with suggestions for future research directions.

2. Background

2.1. Particle motion

The equations governing translational and rotational particle motion are as follows:

$$\dot{\vec{v}} = \frac{\vec{F}}{m} + \vec{g}, \tag{1}$$

$$\dot{\vec{\omega}} = \frac{\vec{T}}{I}. \tag{2}$$

Here, $\dot{\vec{v}}$ represents the particle velocity vector, m denotes the particle mass, $\vec{F}$ is the sum of forces acting on the particle, $\vec{g}$ represents the gravitational acceleration vector, $\vec{\omega}$ is the angular velocity vector, $\vec{T}$ denotes the net torque caused by the contact force, I represents the moment of inertia of the particle and the superscript dot denotes a time derivative. The new velocities and position after the time step Δt are calculated:

$$\vec{v} = \vec{v}_0 + \dot{\vec{v}}_0 \Delta t, \tag{3}$$

$$\vec{r} = \vec{r}_0 + \vec{v}_0 \Delta t, \tag{4}$$

$$\vec{\omega} = \vec{\omega}_0 + \dot{\vec{\omega}}_0 \Delta t, \tag{5}$$

where subscript 0 denotes the initial value.

2.2. Modelling of contact force

The concept of contact force, a pivotal aspect of DEM, has been extensively covered in existing literature [38]. However, for the sake of clarity and convenience, a concise overview will be provided. In this approach, contact is modelled using an elastic spring and viscous dissipation model. When two particles (i and j) come into contact with each other, the force exerted by particle j on particle i, $\vec{F}_{j\to i}$, can be decomposed into two primary components: a normal contact force $\vec{f}_{n,j\to i}$ and a tangential force $\vec{f}_{t,j\to i}$. The former is described as follows:

$$\vec{f}_{n,j\to i} = \begin{cases} -k\delta_{ij}\vec{n}_{ij} - \eta\vec{v}_{ij,n} & \text{if } \delta_{ij} < (r_i + r_j); \\ \mathbf{0} & \text{otherwise}, \end{cases} \tag{6}$$

where δ_{ij} denotes to the overlap between the particles ($r_i + r_j - |\boldsymbol{x}_i - \boldsymbol{x}_j|$), for two particles whose centres are located at $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ respectively; r_i and r_j refer to the radius of particles i and j; k is the stiffness of the spring; η is the coefficient of viscous dissipation; and $\vec{n}_{ij}$ represents the unit normal vector from particle i to particle j. Finally, the normal relative velocity between particle i and j is denoted by $\vec{v}_{ij,n}$ and is calculated as,

$$\vec{v}_{ij,n} = ((\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \vec{n}_{ij})\vec{n}_{ij}. \tag{7}$$

Taking account of both friction and rotation, the tangential force is given by,

$$\vec{f}_{t,j\to i} = -k\delta_{ij,t}\vec{t}_{ij} - \eta\vec{v}_{ij,t} \tag{8}$$

where the unit tangential vector $\vec{t}_{ij}$ is defined by $\vec{t}_{ij} = \vec{v}_{ij,t}/|\vec{v}_{ij,t}|$. The term $\delta_{ij,t}\vec{t}_{ij}$ represents the displacement in the tangential direction and is calculated using the displacement from the previous time step, denoted by $(\delta_{ij,t}\vec{t}_{ij})^*$, and the current velocity and time step as $(\delta_{ij,t}\vec{t}_{ij})^* + \vec{v}_{ij,t}\Delta t$. If, however, the following condition is met

$$|\vec{f}_{t,j\to i}| > \mu_f \left|\vec{f}_{n,j\to i}\right| \tag{9}$$

where μ_f refers to friction coefficient, then sliding occurs, and the tangential force is given by

$$\vec{f}_{t,j\to i} = -\mu_f \left|\vec{f}_{n,j\to i}\right| \vec{t}_{ij}. \tag{10}$$

Table 1 lists the two sets of particle properties used in this study.

Table 1
Mechanical and geometric properties.

Properties	Values	
	I	II
Sphere radius, R (m)	0.05	0.003
Density, ρ (kg m^{-3})	2700	1592
Stiffness, k (N m^{-1})	50 000	10 000
Coefficient of viscous dissipation, η	0.5	0.79
Friction coefficient, μ_f	0.5	0.4

3. Methodology: Implementing neural network for DEM

We use AI tools from the PyTorch library, as it offers several advantages over TensorFlow for the NN4DEM framework: (i) greater flexibility in supporting multiple GPUs, which can facilitate large-scale particle simulation; (ii) more feasibility in customising models due to dynamic computational graph capabilities, allowing for efficient management of computational resources during training (which might, in the future, be integrated with the current method that does not involve training); and (iii) competitive computational efficiency given equivalent model structures.

3.1. Grid system overview for neural network-based computations

The proposed computational methodology relies on a structured grid which facilitates the easy application of convolutional neural networks. The computations are organised in correlation with the geometry and motion of particles, establishing the following guidelines, elucidated with reference to Fig. 1a: (i) Each grid cell can accommodate only one particle at a time; (ii) Each particle is treated as a perfect sphere; (iii) The diagonal cell length is equal to the diameter of each particle (i.e., $2R = \sqrt{3}\Delta x = \sqrt{3}\Delta y = \sqrt{3}\Delta z$ for 3D, $2R = \sqrt{2}\Delta x = \sqrt{2}\Delta y$ for 2D); (iv) An explicit time-stepping scheme is implemented in both spatial and temporal dimensions; and (v) Each particle is permitted to move to a neighbouring cell per time step. Therefore, the maximum time step should be calculated considering the cell size/particle radius, physical properties and stability of the calculation. A widely accepted criterion [38] was implemented as described below:

$$\Delta t_{\max} = \min\left\{\frac{\pi}{5}\sqrt{\frac{m}{k}}, \frac{\Delta x}{v_{\max}}\right\} \tag{11}$$

where $v_{\max}$ denotes the maximum particle velocity at each time step. Adopting a smaller time step will lead to smaller particle movements in each step, thereby reducing the likelihood of unrealistic overlaps. However, this comes at the cost of increased computational time.

As illustrated in Fig. 1b, particles are initially mapped onto the grid structure. The discretised solution fields, which include particle coordinates and velocity components in different Cartesian directions, are then generated within the same grid structure. Each cell in the grid contains relevant information pertaining to the particle located within it. The core of our implementation strategy revolves around the grid system and leveraging the convolutional operations used within a CNN, tailored for tasks such as contact detection and interaction. Discrete convolutions of a CNN are realised by applying kernels/filters (that contain what are commonly referred to as the weights of the neural network) to images (in many machine learning applications) or solution fields (in our case). The weights are typically determined by training the network with datasets. However, instead of training for the weights, we designed a custom kernel based on particle kinematics and dynamics in order to represent physics with the neural network. This filter is a 5 × 5 in 2D (5 × 5 × 5 in 3D) tensor that traverses (or convolves) across the grid-based input data. At each position of the filter (illustrated by the cell (i, j) highlighted in red in the figure), tensor operations are conducted between the corresponding cell values of the grid data currently aligned with the filter and the values of the

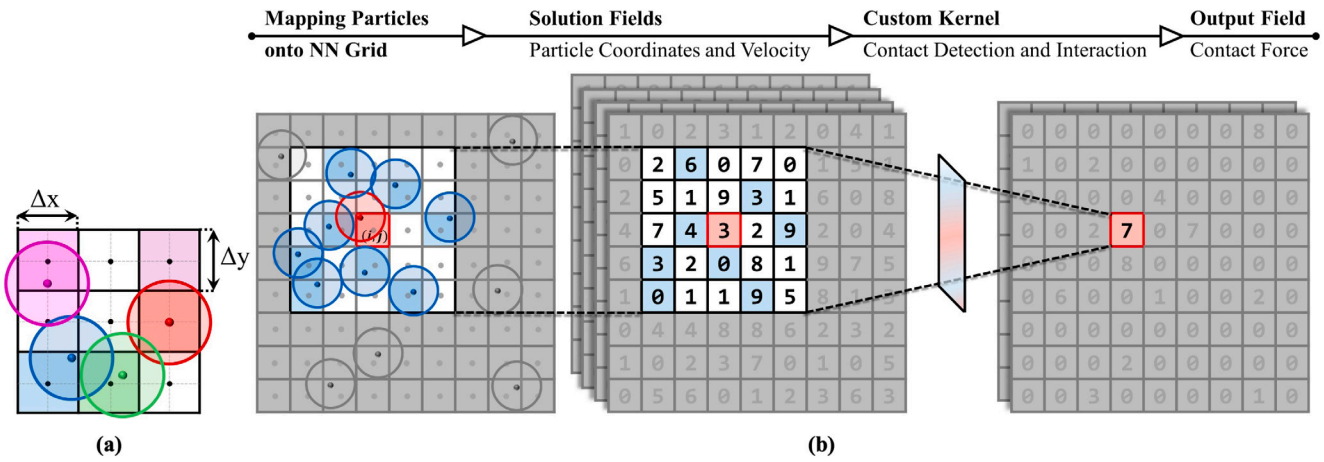


Fig. 1. Illustration of grid system guidelines for NN-based computations using 2D examples. (a) Alignment with particle geometry and motion. (b) Overview of domain discretisation, particle mapping onto the grid and a 5×5 convolutional filter operation applied to the solution fields for contact detection and interaction calculation. The operation outputs the contact force for particles at cell (i, j) (if any). The cell values are arbitrary and used for illustration purposes, not representing actual particle physics.

neighbouring cells. This process results in the kernel outputting the resultant contact force for the particle at cell (i, j).

In essence, this method offers a means by which to replace the computationally expensive loops often encountered in DEM algorithms, executing operations simultaneously across the entire grid and thereby facilitating fast computations. More details on the operations of the kernel and the NN4DEM structure will be presented in the next section.

3.2. Custom kernel for contact detection and interaction

Fig. 2 presents an overview of the CNN architecture used for contact detection and force calculations in the 2D case. The architecture comprises three primary components: a multi-layer input, a physics-based kernel and a convolutional layer. The multi-layer input includes four datasets for particle coordinates and velocity (`x_grid`, `y_grid`, `vx_grid`, and `vy_grid`), all sharing a common grid structure. The input tensor has dimensions $(n_{height} \times n_{width}) \times (n_{channel} = 4)$, where n denotes the number of cells in each dimension and channels. A new kernel, based on the kinematics and dynamics of particles, is used for operations related to contact detection and contact force computations, as shown in the figure. The kernel, with a size of 5, stride of 1 and padding of 2, is applied to the input layers. It returns the contact force interaction between the particle located in the central cell (if any) and the particles in the neighbouring cells (if any). The convolutional layer outputs `fx_grid` and `fy_grid` with the same dimensions as the input layers.

3.2.1. Contact detection algorithm

As depicted in Fig. 2, two key functions are employed to detect particle contacts: `slicing` (Function 1) and `PairwiseDistance` (Function 2). The function `slicing` is used to extract $5 \times 5 \times 5$ sliding local blocks (i.e., sub-grids) from the original input grid as outlined in Algorithm 1. The process effectively segments the input grid into manageable sub-grids, which enables localised analysis of particle interactions.

`PairwiseDistance` is a built-in PyTorch function that computes the pairwise distance between input vectors or between columns of input matrices [39]. Here, this function calculates the distance (`Dist`) between the particles in the central cell of the sub-grid and those in the neighbouring cells using the sub-grids of `x` and `y`. If `Dist` is smaller than the particle diameter $2R$, the particles are considered to be in contact. The condition can be verified using `torch.lt(Dist,2R)`. Accordingly, a `mask` is introduced with `torch.where(torch.lt(Dist,2R), input, other)` function to represent the sub-grid space, containing cell values of 0 and 1, where a cell value of 0 signifies the absence of a particle. The `mask` is later integrated with the contact force computation process.

Algorithm 1: Function for extracting sub-grids

```
import torch.nn.functional as F

def torch_slicing(self, grid, filter_size):
    # Pad the input grid with zeros
    padded_grid = F.pad(grid, (2, 2, 2, 2, 2, 2))
    # Use unfold to extract sliding local blocks from a batched input tensor
    subgrids = padded_grid.unfold(2, filter_size, 1).unfold(3, filter_size, 1).unfold(4, filter_size, 1)
    return subgrids
```

To simulate particle interactions with boundaries, the relative distance of particles from the centre of each cell's side is calculated, representing the free surface or boundary. For ease of implementation and compatibility with the contact detection algorithm, we introduce sets of dummy cells and particles as halo layers around the model by extending the grid. Fig. 3 schematically illustrates the construction of these halo layers at the boundary. Essentially, a dummy particle is added, mirroring the original particle with respect to the boundary.

The current contact detection method is designed for monodisperse particles using a uniform grid and is now described. First, determine the smallest particle's mass and radius, as well as the largest stiffness, to calculate the adaptive time step (Eq. (11)). Then, set up a uniform grid with cell sizes smaller than the diameter of the smallest particle. For particles with a radius equal to the minimum radius ($R = R_{min}$), each particle can occupy one cell as before. For particles with a radius greater than the minimum radius ($R > R_{min}$), they can occupy multiple cells, with each cell centre inside the particle, but each cell can still be occupied by only one particle. Finally, determine the filter size based on the maximum and minimum particle radii using the formula: $filtersize = 5 + \text{floor}(R_{max}/R_{min})$. For a polydisperse system, modifications would be required to adapt this approach. Building upon traditional DEM techniques, additional modifications can be considered for more complex particle systems. In cases where particles have a high aspect ratio, a multiscale approach can be beneficial by employing a hierarchical grid structure. This method divides the simulation space into nested grids with varying resolutions. Finer grid systems handle

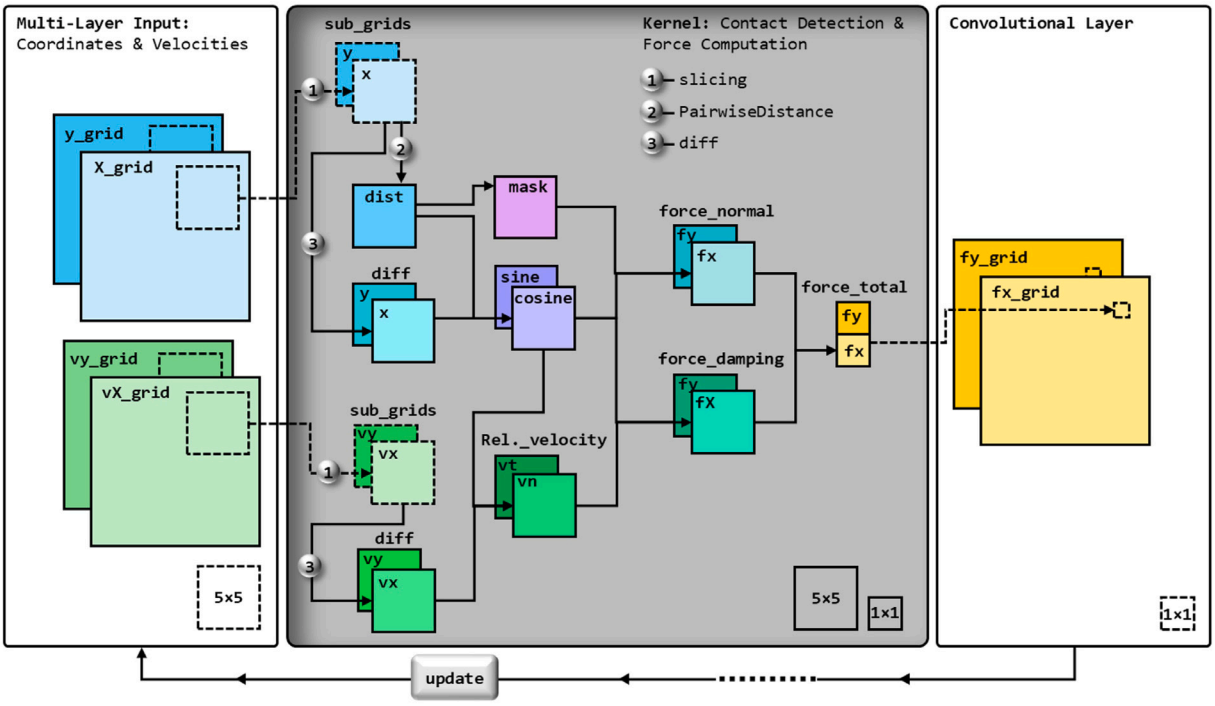


Fig. 2. Overview of the CNN architecture and the tensor operations designed for the physics-based kernel.

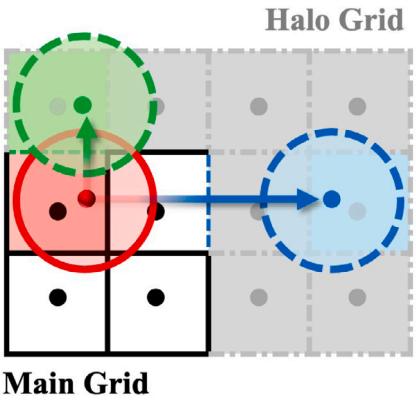


Fig. 3. Schematic representation of the extended 2D halo grid depicted in grey and dotted lines surrounding the main grid, used for boundary condition simulation within the DEM framework. Dummy particles (highlighted in blue and green) are introduced within the halo grid to mirror the positions of original particles with respect to two boundaries.

smaller particles, preventing their overshadowing by larger particles in the same grid cell, which can be an issue with uniform grids. The rules and constraints for NN4DEM will vary depending on the grid coupling techniques employed. For irregularly shaped particles, techniques such as clumped spheres can be implemented [40,41]. The approach involves creating a rigid assembly of smaller spheres that approximate the shape of the irregular particle, with each sphere fixed relative to the others. Within the clump, each sphere functions as an individual contact point, ensuring that interactions adhere to the rigid body constraints of the assembly.

3.2.2. Contact interaction algorithm

To compute the contact forces, `diff` (Function 3) is introduced as another key function for calculating differential values between the central cell and its neighbouring cells, as detailed in Algorithm 2. The function operates by generating a dummy 5×5 matrix, where the value of the central cell is replicated across all cells in the matrix. Then, the function calculates the difference between the original input matrix and this dummy matrix.

Algorithm 2: Calculation of normal contact force excluding damping term

```
def diff(input_matrix):
    # Assume input_matrix is of shape [n, 5, 5], where n is the number of sub-grids
    n, h, w = input_matrix.shape

    # Initialise a dummy matrix where the central value is copied to all cells
    central_value = input_matrix[:, 2, 2].unsqueeze(-1).unsqueeze(-1)
    dummy_matrix = central_value.expand(n, h, w)

    # Calculate the difference between the input matrix and the dummy matrix
    diff_matrix = input_matrix - dummy_matrix

    return diff_matrix
```

This function is applied to the sub-grids of `x` and `y` coordinates to compute the sine and cosine values, representing the normal direction between two particles. Moreover, the `diff` function is applied to the velocity components `vx` and `vy` to obtain the relative velocities (`vn` and `vt`). Using these relative velocities, along with the previously computed `mask`, the force components are calculated in the `x` and `y` directions for both the normal and damping forces (`force_normal` and `force_damping`). The total contact force is then computed as the sum of these force components in the `x` and `y` directions, and this resultant force is outputted by the kernel. To illustrate this concept,

consider the following code snippet for calculating the spring term in the normal contact force:

Algorithm 3: Calculation of normal contact force excluding damping term

```
    # "epsilon" guarantees a non-zero value in the denominator.
    fx_normal = torch.where(torch.lt(dist,2*R),
                k*(dist-2*R)*diffx/torch.maximum(epsilon, dist), zeros)
    fx_total  = fx_normal.view(batch_size, -1).sum(dim=1)
```

The same implementation approach will be applied to handle rotational motion. Following the computation of contact interactions, the translational and rotational velocities, as well as the coordinates and orientations, require updating. The particle's relocation to neighbouring cells involves updating the cell data to reflect the ID (identification) numbers of the particles present, if any. To calculate the tangential force as described in Eqs. (8) to (10), it is necessary to maintain a history of tangential displacement. This can be achieved by storing and recalling data for each particle across time steps. However, in this study, the primary focus is on evaluating the computational performance of the NN4DEM framework, prioritising sufficient physics over detailed exploration. Therefore, we have simplified the calculation by basing the tangential force solely on the current state (Eq. (10)). While this may lead to some inaccuracy in the physics, it allows for a more straightforward assessment of the computational efficiency and scalability of the method.

In a CNN, operations are performed at two levels: (i) convolutional filters are applied across the entire input simultaneously, and (ii) within each filter, some operations can be vectorised/parallelised (e.g., `diff`), while others, such as force calculation processes, are naturally sequential. Vectorised operations at the filter level can be problematic as increasing the batch size of NNs demands more memory e.g., the batch size can increase from (1, 100, 100, 100) to (125, 100, 100, 100). Therefore, to improve memory efficiency, kernel operations can be performed sequentially for the central cell and each neighbouring cell within each iteration. This approach, which mirrors the training process in traditional CNNs, involves iterating through each neighbouring cell, calculating the distance, detecting contact, computing the interaction forces and adding these forces to the existing total force. For example, in Algorithm 4, the process allows the kernel to focus on one neighbouring cell at a time, calculating the differential and distance values, and updating the total force accordingly. However, it should be noted that this method comes at the cost of reduced computational speed. Each iteration updates the total contact force, ensuring memory efficiency by not storing large intermediate results.

Algorithm 4: Calculation of normal contact force excluding damping term

```
    def compute_forces(x_sub_grid, y_sub_grid, vx_sub_grid, vy_sub_grid, R):
        n, h, w = x_grid.shape
        force_x = torch.zeros_like(x_grid)
        force_y = torch.zeros_like(y_grid)

        for i in range(5):
            for j in range(5):
                if i == 2 and j == 2:
                    continue  # Skip the central cell
                diff_x = x_grid[:, 2, 2] - x_grid[:, i, j]
                diff_y = y_grid[:, 2, 2] - y_grid[:, i, j]
                dist = torch.sqrt(diff_x**2 + diff_y**2)
                contact_mask = (dist < 2 * R).float()

                # Compute forces based on differences and contact mask
                fx[:, 2, 2] += contact_mask * (diff_x / dist)
                fy[:, 2, 2] += contact_mask * (diff_y / dist)

        return fx, fy
```

3.3. Single particle validation

To assess the accuracy and reliability of the developed DEM code, we use two benchmark tests: dropping a single particle under gravity both with and without damping, and the sliding test. These tests serve as standard benchmarks to verify the code's ability to simulate particle dynamics, interactions and frictional behaviour through domain discretisation and neural network computations. The fidelity of the results is assessed by comparing them with those obtained from analytical methods. The physical properties of the particle used in the simulations are those listed under Set I in Table 1. The values of k and μ_f for particle–particle interactions are also used for particle-surface interactions. In both cases the time step is set to $0.001\,\mathrm{s}$. For the free fall test, a single particle is released from an initial height of $0.95\,\mathrm{m}$ above a flat rigid surface, subjected to the gravitational acceleration of $9.8\,\mathrm{m/s^2}$. In the sliding test, the particle begins moving with an initial velocity of $\sqrt{2}\,\mathrm{m/s}$ in the xy plane. In the free fall test without damping, the particle returns to its original height. For the damped case, Fig. 4a compares simulation results with outcomes from the analytical method, demonstrating alignment in particle's height history data. In the sliding test, the model ceases movement after traversing a distance of $0.20\,\mathrm{m}$, consistent with the analytical method (Fig. 4b).

4. Test cases

4.1. Random loose packing and scaling analysis

Assessing the performance of NN4DEM in handling particle packing is essential for understanding its computational capabilities and scalability. This evaluation allows us to determine its efficiency and accuracy in simulating large-scale systems. Random loose packing scenarios, especially those involving spheres, are prevalent across various engineering and scientific fields, including geotechnical engineering and the pharmaceutical industries [42–46]. The significance of these scenarios lies in their interesting geometric properties, technological relevance and potential utility as simplified models. We initially investigate two special cases with differing particle numbers to evaluate the computational efficiency and capacity of the method based on the kernel with the sequential operations. These scenarios are run on two different GPU types, representing two distinct computing resources: the Apple MacBook Pro M1 with an 8-core GPU, representing a typical consumer laptop, and an NVIDIA A100 Tensor Core GPU with 6912 CUDA cores, representing high-performance computing for large-scale parallel computation. It should be mentioned that PyTorch can potentially leverage both Tensor Cores and CUDA Cores on an NVIDIA A100 GPU, but it does not directly control which core type gets used for specific operations. It focuses on providing a high-level interface for working with tensors and computations. It relies on the underlying CUDA libraries to manage hardware resources such as cores. The CUDA libraries schedule operations on the available cores (Tensor or CUDA)

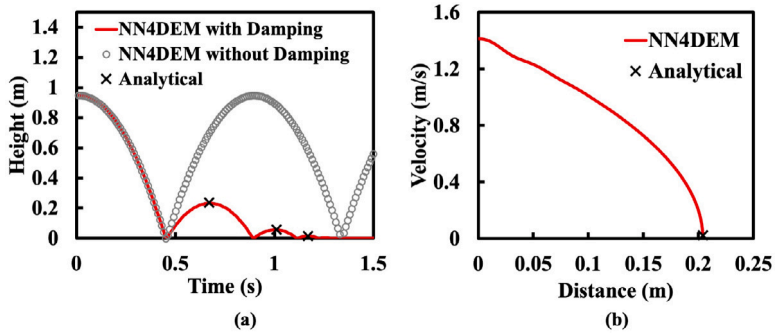


Fig. 4. Comparison of simulation results with analytical predictions for (a) free fall test and (b) sliding test.

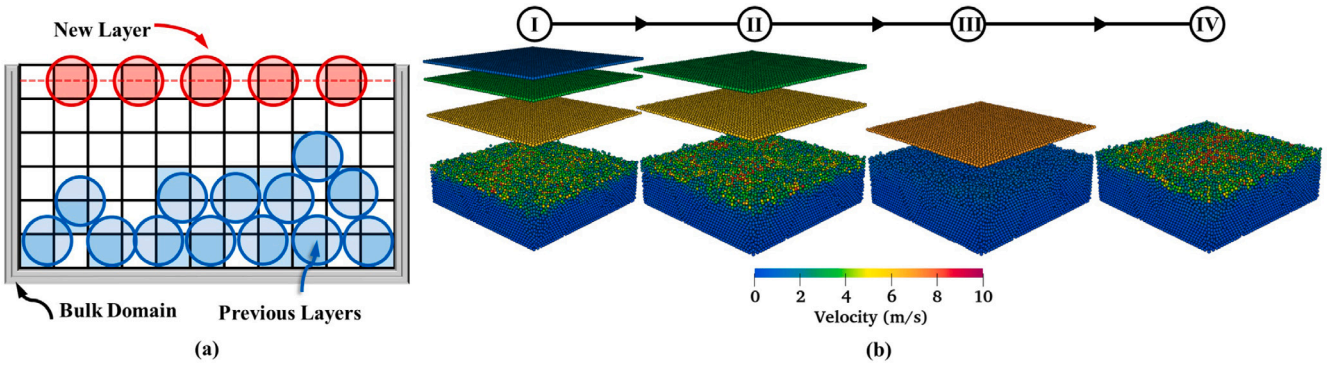


Fig. 5. Illustration of the poured packing process in cuboid containers: (a) schematic example in 2D and (b) 3D DEM results at three sequential steps.

based on factors including the type of operation, data precision and overall workload efficiency [47,48].

In the poured packing scenario, we conduct two simulations in cuboid containers (bulk domains) of dimensions 5 m×5 m×5 m and 80 m × 80 m × 5 m, accommodating 72 029 particles (Case I) and 32 million particles (Case II), respectively. The particle properties for both cases are configured as Set I in Table 1 for both cases. To mitigate the risk of excessive overlap, which could lead to unrealistic solid packing within a stationary bed, we applied a criterion for the stiffness of the spring (k) as follows:

$$k \gg H_b \frac{m_p g}{d_p^2} \tag{12}$$

where H_b denotes the height of the bed, and m_p and d_p refer to the mass and the diameter of particles respectively.

The simulations begin with empty containers and particles are introduced progressively from the top in a layer-by-layer fashion under the influence of gravitational force. In each new layer, particles are arranged in a single row within the grid (i.e., a single sheet), with one particle occupying the centre of every two adjacent cells. To introduce randomness into the packing configuration, the initial vertical velocities (parallel to gravity) are taken from a uniform distribution over the range $[-0.5, 0.5]$ m/s and horizontal velocities are taken from a uniform distribution over the range of $[-0.1, 0.1]$ m/s. The particles are filled in that fashion until the targeted number of particles is reached. Fig. 5 provides a schematic illustration of this process. In this simulation: (i) particle growth is not considered, (ii) rolling motion is included in the dynamics, (iii) friction is neglected in Case II to examine the maximum number of particles computationally feasible, (iv) physical properties remain consistent with the validation cases and (v) the time step is set to 0.001 s for both cases. It is important to note that it is well known that a more rapid feed rate for each particle sheet will contribute to less energy dissipation and sub-optimal particle positions resulting in less dense packings. The speed in which a packing simulation will be completed also depends on the damping due to the restitution coefficient and to a lesser extent on the friction. However, the objective here is not to devise a method for generating the densest packs possible for a given bulk shape and particle size, but rather to create packs that are sufficiently dense and randomly (dis)ordered to serve as representative samples of granular compacts for computational simulations.

Fig. 6 illustrates examples of particle packing, showcasing average packing densities of 30.01% and 52.36%, respectively, based on three realisations for each case — packing density is defined as the ratio of the total volume of particles to the volume of the bulk domain. The simulation runtimes for Cases I and II are 0.05 s per time step and 15 s per time step, respectively. The simulation parameters are summarised in Table 2. A scaling analysis was conducted to evaluate the relationship between the number of particles and simulation speed, while also enabling a comparison with a traditional GPU-parallelised code [49]. The problem domain was a square patch with dimensions $L \times L \times 1$ (in metres) and the particle properties were set according to Table 1 under Set I. To accommodate an increasing number of particles, L ranged from 0.2 m to 2.7 m. Both vectorised and sequential kernels were tested on the A100 40 GB chip. Mixed precision arithmetic is implemented, utilising both single (float32) and half precision (float16), using the `torch.autocast` function. The results were compared with those from simulations using the Chrono::Granular code on an NVIDIA Tesla V100 with 32 GB of memory, with both simulations running for 50,000 steps. As shown in Fig. 7, the codes exhibited linear scaling with problem size up to approximately 40 million particles, where the device memory was exhausted using NN4DEM::Sequential. The memory limit for NN4DEM::Vectorised was about 12 million particles, but its speed was 4.1 times faster than NN4DEM::Sequential and 4.9 times faster than Chrono::Granular. Chrono::Granular can handle up to 700 million particles, which is significantly higher than the current version of NN4DEM, highlighting the potential for further memory efficiency optimisation in NN4DEM. Although, for a fair evaluation of computational efficiency, the following points should be considered: (i) The methodology presented in this paper prioritises demonstrating principles without adding

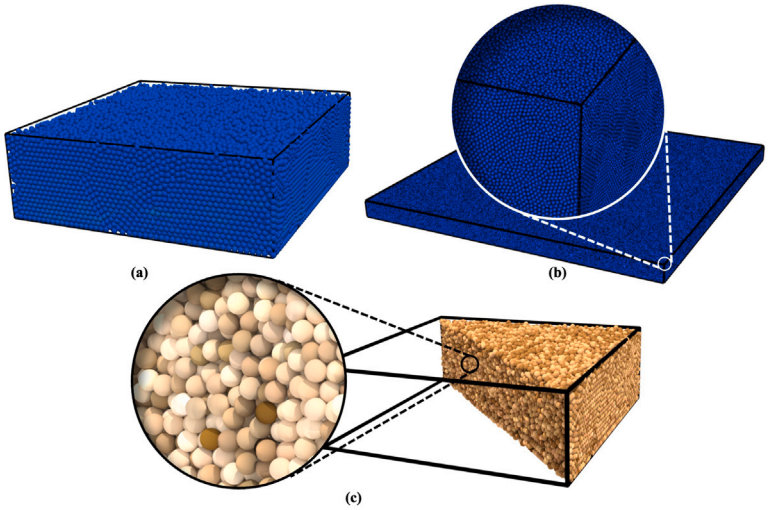


Fig. 6. Examples of particle packing for (a) Case I with 100 000 particles, (b) Case II with 32 million particles and (c) cross-sectional view of the pack in Case I showcasing particle arrangements in the middle of the bulk domain. Different colour tones represent individual particles for enhanced visualisation.

Table 2
Simulation parameters for the random packing test.

Properties	Values		Unit
	Case 1	Case 2	
Bulk domain dimensions	$5 \times 5 \times 5$	$80 \times 80 \times 5$	m
Cells number in x, y and z directions	100, 100, 100	1600, 1600, 100	–
Cell size	0.05	0.05	m
Time step	0.001	0.001	s
Number of particles	72 029	32 000 000	–
Initial velocity in x and y directions	$[-0.1, 0.1]$	$[-0.1, 0.1]$	m/s
Initial velocity in z direction	$[-0.5, 0.5]$	$[-0.5, 0.5]$	m/s
Feed rate	4	8	layer/s

complexity that may arise from further code optimisation, which could potentially improve computational efficiency. (ii) The computational capacity of the method is mainly determined by the number of grid cells, unlike traditional DEM, where the number of particles is the primary factor. Thus, comparing with conventional methods involving similar physics and employing other parallelisation techniques, there may be negligible differences in terms of runtime and maximum particle count.

Insights from our previous work in fluid dynamics [31] and neutron diffusion [32], which employs similar concepts albeit with more complex physics, suggests the potential for a substantial increase in particle count. These studies involve solving PDEs on structured grids, where variables are stored within the grid cells/nodes and have been able to run on multiple GPUs. This makes the proposed NN4DEM approach amenable to the parallelisation methods used in such structured grid models. The methods distribute and assign computational tasks to subdomains (i.e., GPUs), with communication between these subdomains occurring through halo cells/nodes. This message-passing approach is well-suited for model parallelisation and inter-GPU communication in PyTorch, using packages such as NCCL. With further code improvements, particularly focusing on memory efficiency (e.g., two-grid systems and compression approaches), NN4DEM could be extended to accommodate about *10 billion particles*. For example, simulating 200 million particles per GPU by employing techniques to store more particle data per cell would require approximately either 50 A100 40 GB or 25 A100 80 GB GPUs — at the time of writing, the necessary computational resources for both configurations are available.

It should be noted that the dependency of computational cost on the number of grid cells may present limitations in certain applications.

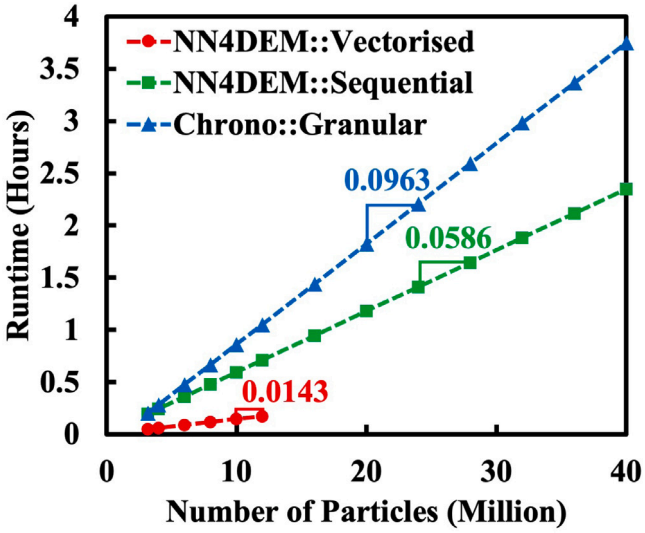


Fig. 7. Scaling results of the packing simulation, comparing NN4DEM methods with Chrono::Granular, a traditional GPU-parallelised code. The Chrono::Granular data refers to results from [49] using a V100 —SI32GB GPU.

However, this dependency can be particularly advantageous in scenarios such as DEM-Fluid modelling, where the entire domain can typically be discretised, and an Eulerian framework is used for fluids.

4.2. Hopper test

The hopper test evaluates NN4DEM on the dynamics exhibited by a flow of particles. A flat-bottomed hopper with a single-component system consisting of plastic spheres is set up according to the experiment in Jian and Gao [50], as shown in Fig. 8a. The hopper has a height (H_h) of 0.40 m, a width (W_h) of 0.20 m and a depth (D_h) of 0.04 m. An orifice (W_o) of 0.04 m is symmetrically positioned on the lower surface. The simulation consists of two parts: first, packing the particles into the hopper, and second, discharging them. The initial height of the packed particles (H_p) is 0.36 m. The properties of the particles are listed under Set II in Table 1. The discharging simulation runs for a physical time of 7.5 s, with approximate runtimes of 0.29 h and 1.18 h on the HPC system with the A100 chip, utilising the kernel with the vectorised and sequential operations, respectively. For comparison, a

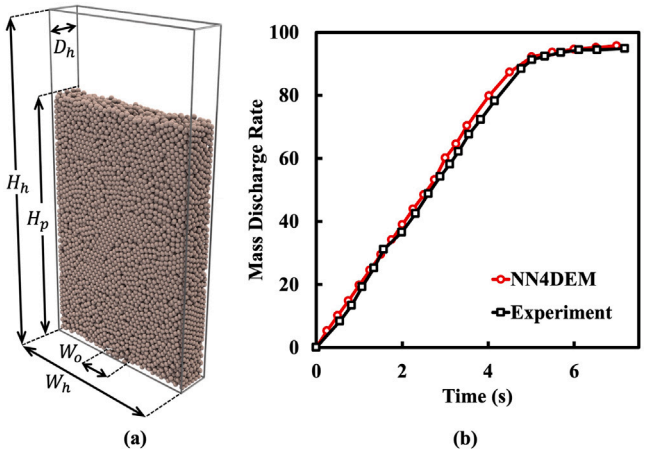


Fig. 8. (a) Schematic visualisation of the flat-bottomed hopper. (b) The mass discharge ratio for single-component hoppers with plastic spheres, as obtained from NN4DEM simulations and compared with experimental results [50].

similar case performed by the Chrono-DEM engine on a single NVIDIA A5000 GPU [51] has a runtime of 0.35 h. The runtime is ~17% slower than NN4DEM with the vectorised kernel, where both simulations used the same time step of 5×10^{-6} s. Some caution should be applied to this comparison, as there are differences in computational resources and the contact detection process used by Chrono-DEM and NN4DEM. Chrono-DEM uses an interesting method called asynchronous threads management algorithm, which runs contact detection and contact calculation in parallel to speed up the model. However, this approach involves a trade-off: contact detection is not performed at each time step. Instead, it artificially enlarges contact geometries in the DEM system, which can lead to false positives in contact detection. In contrast, NN4DEM performs contact checks at every time step. Therefore, NN4DEM's performance is competitive with other GPU-parallelised codes and could be significantly faster if such a threads management method is implemented.

Fig. 8b presents the relative mass discharge, comparing the results obtained from NN4DEM with experimental data. The comparison demonstrates an excellent match. Also, Fig. 9 includes snapshots with lateral views of the hopper at one-second intervals. In the final frame at 6 s, the residual inclination angle is 18°, which aligns with experimental observations. This angle indicates the presence of dead zones within flat-bottomed hoppers, where particles are unable to fully discharge.

5. Conclusion

This paper introduces NN4DEM, a pioneering computational approach to DEM simulations employing a neural network framework. The neural network in NN4DEM derives its weights from the discretised governing equations of a spring-dashpot model, enabling contact detection and interaction between particles and boundaries. NN4DEM's performance was validated through standard benchmark cases such as free fall and sliding tests, where its results were compared against analytical methods. Moreover, NN4DEM exhibits computational efficiency and scalability, as demonstrated by its performance in random packing simulations using two different GPU types, capable of accommodating up to 10 billion particles.

The method presented here outlines the implementation principles of neural networks for this application. In future work, the method will be optimised in three ways: (i) improving versatility for broader applications, including polydisperse systems with irregularly shaped particles and multiscale multi-physics problems; (ii) advancing programming techniques through the following approaches: (a) implementing memory-efficient methods, such as a two-grid system and compression techniques, and addressing memory concerns by incorporating a comprehensive history-based friction model; (b) applying traditional optimisation techniques, such as selective updates, data management, error tolerance and adaptive methods; (c) employing multiple GPU parallelisation techniques; and (iii) integrating this method with surrogate data-driven modelling frameworks.

CRediT authorship contribution statement

Sadjad Naderi: Writing – original draft, Visualization, Supervision, Software, Methodology, Conceptualization. **Boyang Chen:** Writing – review & editing, Visualization, Software, Methodology, Conceptualization. **Tongan Yang:** Visualization, Software. **Jiansheng Xiang:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Claire E. Heaney:** Writing – review & editing, Conceptualization. **John-Paul Latham:** Writing – review & editing, Funding acquisition. **Yanghua Wang:** Funding acquisition. **Christopher C. Pain:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

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

Data availability

The code used to generate these results can be found at the following github repository: https://github.com/bc1chen/NN4DEM.

Acknowledgements

The authors would like to acknowledge the following EPSRC grants: the PREMIERE programme grant, "AI to enhance manufacturing, en-

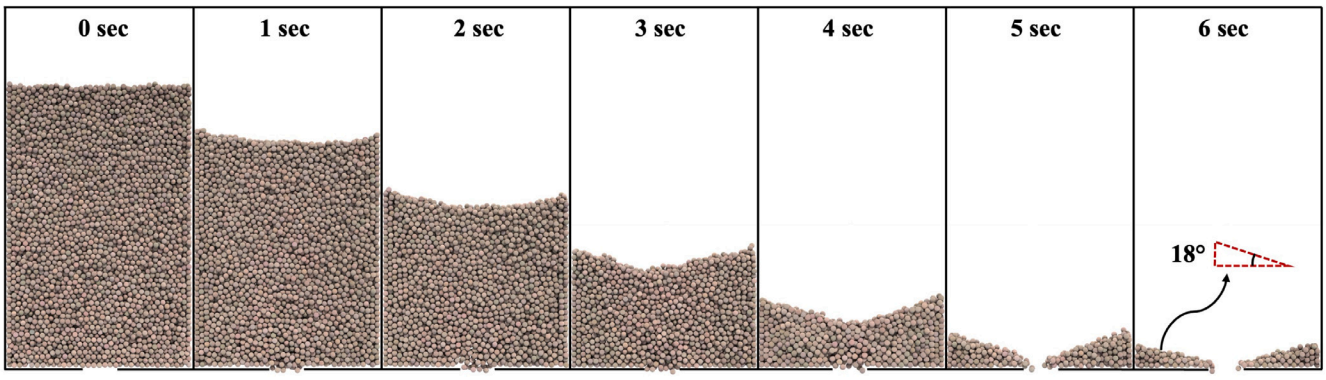


Fig. 9. Snapshots of the discharging behaviour of the plastic spherical particles.

ergy, and healthcare" (EP/T000414/1); ECO-AI, "Enabling CO_2 capture and storage using AI" (EP/Y005732/1); WavE-Suite, "New Generation Modelling Suite for the Survivability of Wave Energy Convertors in Marine Environments" (EP/V040235/1); INHALE, "Health assessment across biological length scales" (EP/T003189/1); AI-Respire, "AI for personalised respiratory health and pollution" (EP/Y018680/1); RELIANT, "Risk EvaLuatIon fAst iNtelligent Tool for COVID19" (EP/V036777/1); MUFFINS, "MUltiphase Flow-induced Fluid-flexible structure InteractioN in Subsea" (EP/P033180/1); and CO-TRACE, "COvid-19 Transmission Risk Assessment Case Studies — education Establishments" (EP/W001411/1). Support from Imperial-X's Eric and Wendy Schmidt Centre for AI in Science (a Schmidt Sciences program) is gratefully acknowledged.

References

[1] Y. Dong, D. Yan, L. Cui, An efficient parallel framework for the discrete element method using GPU, Appl. Sci. 12 (6) (2022) 3107.

[2] R. Li, G. Duan, M. Sakai, DEM simulations in nuclear engineering: a review of recent progress, J. Nucl. Sci. Technol. 61 (3) (2024) 285–306, http://dx.doi.org/10.1080/00223131.2023.2231969.

[3] P. Bhalode, M. Ierapetritou, Discrete element modeling for continuous powder feeding operation: Calibration and system analysis, Int. J. Pharm. 585 (2020) 119427.

[4] Z. Wu, Y. Wu, A. Zakhvatayeva, X. Wang, Z. Liu, M. Yang, Q. Zheng, C.-Y. Wu, Influence of moisture content on die filling of pharmaceutical powders, J. Drug Deliv. Sci. Technol. 78 (2022) 103985, http://dx.doi.org/10.1016/j.jddst.2022.103985.

[5] G.-Y. Liu, W.-J. Xu, N. Govender, D.N. Wilke, Simulation of rock fracture process based on GPU-accelerated discrete element method, Powder Technol. 377 (2021) 640–656.

[6] M. Spellings, R.L. Marson, J.A. Anderson, S.C. Glotzer, GPU accelerated discrete element method (DEM) molecular dynamics for conservative, faceted particle simulations, J. Comput. Phys. 334 (2017) 460–467.

[7] P. Sánchez, D.J. Scheeres, Simulating asteroid rubble piles with a self-gravitating soft-sphere distinct element method model, Astrophys. J. 727 (2) (2011) 120.

[8] Y.K. Gujjala, H.-M. Kim, D.-W. Ryu, GPGPU-based parallel computation using discrete elements in geotechnics: A state-of-art review, Arch. Comput. Methods Eng. 30 (3) (2023) 1601–1622.

[9] B. Yan, R.A. Regueiro, Superlinear speedup phenomenon in parallel 3D Discrete Element Method (DEM) simulations of complex-shaped particles, Parallel Comput. 75 (2018) 61–87.

[10] S. Zhao, J. Zhao, Revolutionizing granular matter simulations by high-performance ray tracing discrete element method for arbitrarily-shaped particles, Comput. Methods Appl. Mech. Engrg. 416 (2023) 116370.

[11] H. Norouzi, R. Zarghami, N. Mostoufi, New hybrid CPU-GPU solver for CFD-DEM simulation of fluidized beds, Powder Technol. 316 (2017) 233–244.

[12] M. Sakai, How should the discrete element method be applied in industrial systems?: A review, KONA Powder Part. J. 33 (2016) 169–178, http://dx.doi.org/10.14356/kona.2016023.

[13] J.-H. He, M.-G. Li, J.-J. Chen, A novel unresolved/semi-resolved CFD-DEM coupling method with dynamic unstructured mesh, Int. J. Numer. Anal. Methods Geomech. (2024).

[14] Y. Tian, S. Zhang, P. Lin, Q. Yang, G. Yang, L. Yang, Implementing discrete element method for large-scale simulation of particles on multiple GPUs, Comput. Chem. Eng. 104 (2017) 231–240.

[15] N. Govender, D.N. Wilke, C.Y. Wu, R. Rajamani, J. Khinast, B.J. Glasser, Large-scale GPU based DEM modeling of mixing using irregularly shaped particles, Adv. Powder Technol. 29 (10) (2018) 2476–2490.

[16] Z. Zhao, L. Zhou, L. Bai, M.A. El-Emam, R. Agarwal, Modeling and validation of coarse-grained computational fluid dynamics–discrete element method for dense gas—solid flow simulation in a bubbling fluidized bed, Phys. Fluids 35 (4) (2023).

[17] L. Fang, R. Zhang, C. Vanden Heuvel, R. Serban, D. Negrut, Chrono:: GPU: An open-source simulation package for granular dynamics using the discrete element method, Processes 9 (10) (2021) 1813.

[18] N. Govender, D.N. Wilke, S. Kok, Blaze-DEMGPU: Modular high performance DEM framework for the GPU architecture, SoftwareX 5 (2016) 62–66, http://dx.doi.org/10.1016/j.softx.2016.04.004.

[19] R. Zhang, C. Vanden Heuvel, A. Schepelmann, A. Rogg, D. Apostolopoulos, S. Chandler, R. Serban, D. Negrut, A GPU-accelerated simulator for the DEM analysis of granular systems composed of clump-shaped elements, Eng. Comput. (2024) 1–21.

[20] Q. Zhou, W.-J. Xu, R. Lubbe, Multi-scale mechanics of sand based on FEM-DEM coupling method, Powder Technol. 380 (2021) 394–407.

[21] Z. Zheng, M. Zang, S. Chen, H. Zeng, A GPU-based DEM-FEM computational framework for tire-sand interaction simulations, Comput. Struct. 209 (2018) 74–92.

[22] W. Liu, C.-Y. Wu, Modelling complex particle–fluid flow with a discrete element method coupled with Lattice Boltzmann methods (DEM-LBM), ChemEngineering 4 (4) (2020) http://dx.doi.org/10.3390/chemengineering4040055.

[23] N. Younes, A. Wautier, R. Wan, O. Millet, F. Nicot, R. Bouchard, DEM-LBM coupling for partially saturated granular assemblies, Comput. Geotech. 162 (2023) 105677, http://dx.doi.org/10.1016/j.compgeo.2023.105677.

[24] L. Lu, X. Gao, J.-F. Dietiker, M. Shahnam, W.A. Rogers, Machine learning accelerated discrete element modeling of granular flows, Chem. Eng. Sci. 245 (2021) 116832.

[25] Z. Lai, Q. Chen, L. Huang, Machine-learning-enabled discrete element method: Contact detection and resolution of irregular-shaped particles, Int. J. Numer. Anal. Methods Geomech. 46 (1) (2022) 113–140.

[26] M. Wu, J. Wang, Estimating contact force chains using artificial neural network, Appl. Sci. 11 (14) (2021) 6278.

[27] M. Wu, J. Wang, Prediction of 3D contact force chains using artificial neural networks, Eng. Geol. 296 (2022) 106444.

[28] T. Zhang, S. Li, H. Yang, F. Zhang, Prediction of constrained modulus for granular soil using 3D discrete element method and convolutional neural networks, J. Rock Mech. Geotech. Eng. (2024).

[29] X.-Z. Zhao, T.-Y. Xu, Z.-T. Ye, W.-J. Liu, A TensorFlow-based new high-performance computational framework for CFD, J. Hydrodyn. 32 (4) (2020) 735–746, http://dx.doi.org/10.1007/s42241-020-0050-0.

[30] Q. Wang, M. Ihme, Y.-F. Chen, J. Anderson, A TensorFlow simulation framework for scientific computing of fluid flows on tensor processing units, Comput. Phys. Comm. 274 (2022) 108292, http://dx.doi.org/10.1016/j.cpc.2022.108292.

[31] B. Chen, C.E. Heaney, C.C. Pain, Using AI libraries for incompressible computational fluid dynamics, arXiv (2024) http://dx.doi.org/10.48550/arXiv.2402.17913, 2402.17913.

[32] T.R.F. Phillips, C.E. Heaney, B. Chen, A.G. Buchan, C.C. Pain, Solving the discretised neutron diffusion equations using neural networks, Internat. J. Numer. Methods Engrg. 124 (21) (2023) 4659–4686, http://dx.doi.org/10.1002/nme.7321.

[33] B. Chen, C.E. Heaney, J.L. Gomes, O.K. Matar, C.C. Pain, Solving the discretised multiphase flow equations with interface capturing on structured grids using machine learning libraries, Comput. Methods Appl. Mech. Engrg. 426 (2024) 116974.

[34] B. Chen, et al., Solving the discretised shallow water equations using neural networks, 2024, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4956116.

[35] T.R.F. Phillips, C.E. Heaney, B. Chen, A.G. Buchan, C.C. Pain, Solving the discretised Boltzmann transport equations using neural networks: Applications in neutron transport, 2023, http://dx.doi.org/10.48550/arXiv.2301.09991, arXiv preprint, 2301.09991.

[36] L. Li, J. Xiang, B. Chen, C.E. Heaney, S. Dargaville, C.C. Pain, Implementing the discontinuous-Galerkin finite element method using graph neural networks, 2024, http://dx.doi.org/10.2139/ssrn.4698813, submitted for publication.

[37] Y. Li, et al., An AI-based integrated framework for anisotropic electrical resistivity imaging, 2024, in preparation.

[38] Y. Tsuji, T. Kawaguchi, T. Tanaka, Discrete particle simulation of two-dimensional fluidized bed, Powder Technol. 77 (1) (1993) 79–87.

[39] PyTorch, Pairwisedistance, 2024, URL https://pytorch.org/docs/stable/generated/torch.nn.PairwiseDistance.html. (Accessed: 04 July 2024).

[40] M.H. Ahmadian, W. Zheng, Simulating the fluid–solid interaction of irregularly shaped particles using the LBM-DEM coupling method, Comput. Geotech. 171 (2024) 106395.

[41] M. Alizadeh, A. Hassanpour, M. Pasha, M. Ghadiri, A. Bayly, The effect of particle shape on predicted segregation in binary powder mixtures, Powder Technol. 319 (2017) 313–322.

[42] C. Song, P. Wang, H.A. Makse, A phase diagram for jammed matter, Nature 453 (7195) (2008) 629–632.

[43] Y. Cheng, S. Guo, H. Lai, Dynamic simulation of random packing of spherical particles, Powder Technol. 107 (1–2) (2000) 123–130.

[44] S.R. Jaggannagari, R.K. Desu, J. Reimann, Y. Gan, M. Moscardini, R.K. Annabattula, DEM simulations of vibrated sphere packings in slender prismatic containers, Powder Technol. 393 (2021) 31–59.

[45] Q. Qian, L. Wang, X. An, Y. Wu, J. Wang, H. Zhao, X. Yang, DEM simulation on the vibrated packing densification of mono-sized equilateral cylindrical particles, Powder Technol. 325 (2018) 151–160.

[46] G.D. Scott, Packing of spheres: packing of equal spheres, Nature 188 (4754) (1960) 908–909.

[47] NVIDIA Corporation, NVIDIA Tensor Cores, Technical Report, NVIDIA, 2020, https://www.nvidia.com/en-us/data-center/tensor-cores/. (Accessed 30 May 2024).

[48] NVIDIA Corporation, CUDA Programming Guide, Technical Report, NVIDIA, 2021, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. (Accessed: 30 May 2024).

[49] C. Kelly, N. Olsen, D. Negrut, Billion degree of freedom granular dynamics simulation on commodity hardware via heterogeneous data-type representation, Multibody Syst. Dyn. 50 (2020) 355–379.

[50] B. Jian, X. Gao, Investigation of spherical and non-spherical binary particles flow characteristics in a discharge hopper, Adv. Powder Technol. 34 (5) (2023) 104011.

[51] R. Zhang, B. Tagliafierro, C.V. Heuvel, S. Sabarwal, L. Bakke, Y. Yue, X. Wei, R. Serban, D. Negruţ, Chrono DEM-Engine: A discrete element method dual-GPU simulator with customizable contact forces and element shape, Comput. Phys. Comm. 300 (2024) 109196.