Computer Methods in Applied Mechanics and Engineering 433 (2025) 117536

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

A physical-information-flow-constrained temporal graph neural network-based simulator for granular materials

Shiwei Zhao *, Hao Chen, Jidong Zhao *

Department of Civil and Environmental Engineering, The Hong Kong University of Science and Technology, Hong Kong Special Administrative Region

ARTICLE INFO

Dataset link: https://www.sudosimlab.com/en/download/

Keywords:
Temporal graph neural network
Granular materials
Graph network simulator
Physical-information-flow-constrained

ABSTRACT

This paper introduces the Temporal Graph Neural Network-based Simulator (TGNNS), a novel physical-information-flow-constrained deep learning-based simulator for granular material modeling. The TGNNS leverages a series of frames, each representing material point positions, enabling particle dynamics to propagate through the sequence, resulting in a more physically grounded architecture for granular flow learning. The TGNNS has been thoroughly trained, validated, and tested using simulation data derived from a hierarchical multiscale modeling approach, DEMPM, which combines the Material Point Method (MPM) and the Discrete Element Method (DEM). Results demonstrate that the TGNNS performs robustly with previously unseen datasets of varying granular column sizes, even under manually incorporated barrier boundary conditions. Remarkably, the TGNNS operates at a speed 100 times faster than direct numerical simulation using the state-of-the-art GPU-based DEMPM. Employing a unique deep learning architecture that is constrained by the flow of physical information, the TGNNS offers a pioneering learning paradigm for multiscale emerging behaviors of granular materials and provides a potential solution to physics-based modeling in digital twins involving granular materials.

1. Introduction

Granular materials are ubiquitous in nature, industry, and engineering, and they have complex collective behaviors steaming from their discrete solid particles, such as soil, powders, and grains, that interact with each other through various contact forces. Understanding their behaviors is crucial for a wide range of applications in our daily lives, such as geotechnical engineering, pharmaceutical manufacturing, and food processing. For instance, with climate change, landslides have become more frequent and can trigger catastrophic natural hazards, posing a significant threat to both human lives and infrastructure [1].

Digital twins are emerging as facilitators of substantial and sustainable progress in various fields, including science, engineering, and medicine [2,3]. In the realm of disaster management such as natural hazards like landslides, for example, digital twins can be instrumental in enhancing preparedness and response strategies. By enabling real-time monitoring, preventive maintenance, resource allocation, and post-disaster recovery, digital twins can provide a comprehensive approach to mitigating risks and safeguarding communities.

It is crucial to model the dynamic behaviors of granular materials accurately and efficiently for virtual representation in digital twins. Direct numerical simulations (DNS) are commonly used for this purpose, employing two dominant numerical methods: continuum-based and micromechanics-based methods. The most prevalent continuum-based method is the finite element method

* Corresponding authors.
E-mail addresses: ceswzhao@ust.hk (S. Zhao), jzhao@ust.hk (J. Zhao).

https://doi.org/10.1016/j.cma.2024.117536
Received 2 August 2024; Received in revised form 4 November 2024; Accepted 5 November 2024
Available online 18 November 2024

(FEM), but it may encounter numerical issues, such as mesh distortion for large-deformation problems, due to its mesh-dependent nature. In contrast, particle-based or mesh-free methods, such as the material point method (MPM) [4] and smoothed particle hydrodynamics (SPH) [5], are better suited for large-deformation problems like landslides [6,7]. However, these continuum-based methods require phenomenological constitutive models to describe the stress–strain relation of the material. Unfortunately, these models struggle to capture the discrete nature of granular materials at the particulate scale, making them less accurate or robust in modeling the complex behaviors of granular flows.

The discrete element method (DEM) [8], a representative micromechanics-based method, has proven effective and robust in modeling the complex behaviors of granular materials by considering interactions between particles at the particulate scale. By incorporating more detailed particle characteristics, such as realistic particle shape [9,10], it is even possible to achieve hyper-fidelity simulations of granular materials. However, the issue of computational efficiency remains a significant challenge for DEM, hindering its application in engineering-scale simulations. This challenge can be largely mitigated by utilizing advanced parallel computing algorithms on modern graphics processing units (GPUs), e.g., advanced ray-tracing algorithms for contact detection [11,12]. In addition, machine learning can also be employed to accelerate DEM simulations, e.g., learning the contact interaction for non-spherical particles [13]. It is important to note that the increase in computational efficiency resulting from utilizing machine learning for contact interactions may not be significant if the machine learning model is overly complex and aimed at generalization purposes.

Hierarchical coupling of the continuum and discrete methods is a promising approach for effectively capturing the multiscale characteristics of granular materials. This approach leverages the efficiency of the continuum-based method and the accuracy of the micromechanics-based method. Two exemplified coupling schemes, FEM-DEM [14] and MPM-DEM [15], have been extensively explored and proven effective for modeling granular materials. While these hierarchical multiscale modeling approaches have significantly improved computational efficiency compared to pure-DEM simulations for engineering-scale problems, they still face challenges in providing prompt responses for digital twins.

To address the limitations of model-driven DNS, surrogate modeling is a crucial area for virtual representation in digital twins [16]. Surrogate modeling can be implemented at various levels to solve the dynamics of granular materials. Specifically, one can propose surrogate models to replace conventional constitutive models, enhancing their accuracy and robustness, and subsequently incorporate them into continuum-based methods for simulations (e.g., [17,18]). This type of surrogate modeling is often referred to data-driven constitutive modeling. For a comprehensive overview of this topic, interested readers are referred to a recent review [19]. While data-driven constitutive models are computationally more efficient than utilizing DEM-simulated responses in conjunction with continuum-based numerical methods, their computational efficiency may not be sufficient for prompt responses of granular dynamics in a digital twin. Therefore, it is necessary to emphasize the importance of fully surrogate modeling of granular materials.

Graph neural networks (GNNs) [20] have garnered increasing attention for their ability to handle complex structured data in machine learning. Researchers have introduced specific Graph Networks (GNs), such as Interaction Networks [21,22], to learn physical systems and enable fully surrogate modeling. GN-based surrogate modeling has proven effective in modeling the dynamics of granular materials [23,24], giving rise to the term Graph Network Simulator (GNS). The GNS learns granular dynamics by utilizing a conventional multi-layer GNN, which maps an input graph to an output graph with the same structure. However, the information flow between layers is abstracted as message passing, lacking interpretability. To address these limitations, we propose a novel neural network architecture called the Temporal Graph Neural Network-based Simulator (TGNNS) for learning granular dynamics. The proposed TGNNS offers enhanced interpretability by incorporating a physical information flow constraint directly into the architecture, as opposed to the conventional approach of applying constraints solely on the loss functions, e.g., physics-informed neural networks [25,26]. Note that the interpretability provided by the TGNNS is model based, and interested readers are referred to the literature [27] for a detailed discussion on the model-based interpretability.

The rest of the paper is organized as follows. Section 2 provides detail of the proposed temporal graph neural network, followed by the proposed TGNNS in Section 3. Section 4 introduces the details of data preparation, training, and inference for the TGNNS. A comprehensive evaluation of the proposed model is carried out in Section 5. Conclusions are made in Section 6.

2. Temporal graph neural network

2.1. Graph representation

In particle-based numerical methods such as material point method (MPM) [28], smoothed particle hydrodynamics (SPH) [29] and peridynamics (PD) [30], the partial differential equations governing particle dynamics are integrated over subdomains within finite influencing regions. As shown in Fig. 1(a), for example, to model particle dynamics at a given time instant, particle–node mapping in MPM is implemented among nodes and their neighboring particles in terms of a shape function (Fig. 1(b)), and particles in SPH interact directly through a kernel function defined on a subdomain that covers only limited neighboring particles (Fig. 1(c)). The holistic inter-particle interactions can be implicitly represented by a universal approximate function through a neural network no matter whether those particles interact with each other directly or indirectly.

The interactions among n material points in a simulation domain can be described by an n-node undirected graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}) \tag{1}$$

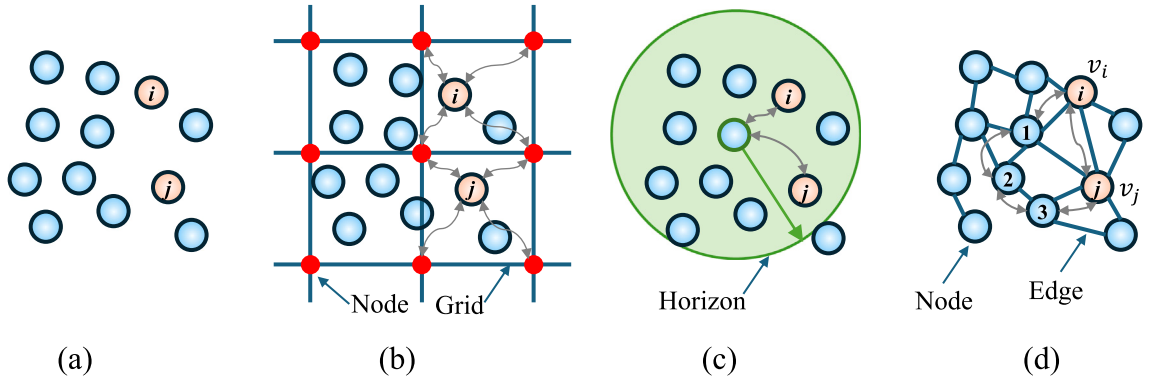

Fig. 1. (a) Lagrangian particles or material points, (b) particles interact via an Eulerian background grid, (c) particles interact via horizon-based integration in SPH or PD, and (d) particles interact via k-hop neighbors in a graph.

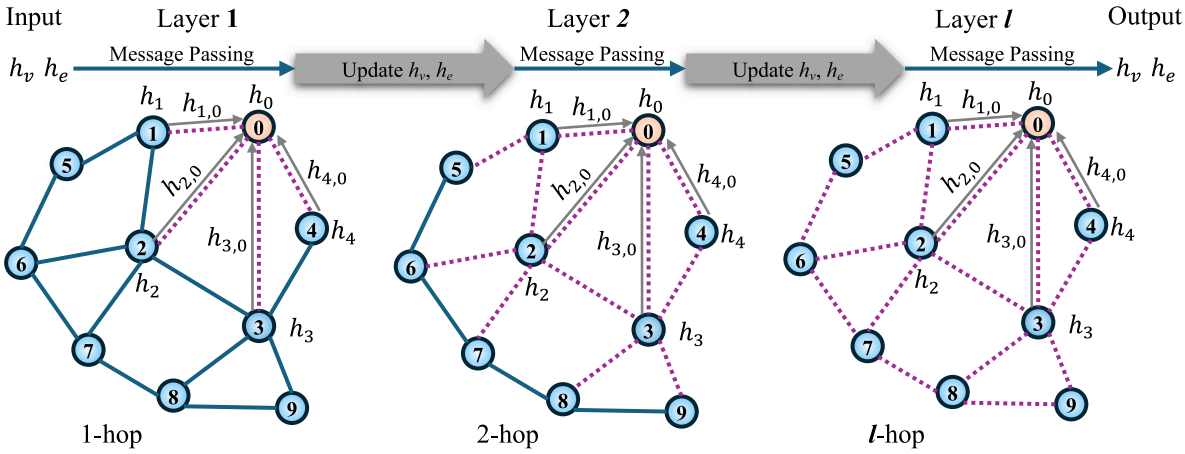

Fig. 2. Graph neural networks with a message passing scheme: information flows on a fixed graph. The dashed edges denote the information used between node 0 and its k-hop neighbors.

where $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ and $\mathcal{E} = \{e_{1,1}, e_{1,2}, \dots, e_{i,j}\}$ are all graph nodes and edges, respectively. Each graph node corresponds to a material point, and each bidirectional edge represents the interaction between two nodes, e.g., $e_{i,j} = (v_i, v_j) \in \mathcal{E}$ denoting an edge from node v_i to node v_j (see Fig. 1(d)). The neighbor set of node v_i is defined as

$$\mathcal{N}(v_i) = \{v_j | (v_i, v_j) \in \mathcal{E}\}. \tag{2}$$

All edges can be defined by an adjacency matrix $\mathcal{A} \in \{0, 1\}^{n \times n}$, and the sum of its all items is equal to the edge count. Note that the adjacency matrix $\mathcal{A}$ is strongly sparse with the assumption that interactions take place within a finite region (influencing range). It is worth pointing out that the inter-node interaction occurs not only between two neighboring nodes but also through intermediate nodes, e.g., nodes 1, 2 and 3 between v_i and v_j in Fig. 1(d). These immediate nodes denote as k-hop neighbors.

2.2. Graph neural networks

The prevailing graph neural networks (GNNs) follow a message passing scheme within a graph $\mathcal{G}$ itself such that the output graph $\mathcal{G}'$ remains the same structure. Both the input and output can have three different levels of features, namely, graph-level feature, node-level feature and edge-level feature. For simplicity, the graph feature is not considered in this work. Hence, given node feature vectors $\boldsymbol{x}_v$ for all $v \in \mathcal{V}$ and edge feature vectors $\boldsymbol{x}_e$ for all $e \in \mathcal{E}$, the graph $\mathcal{G}^{(l)}$ at the lth layer is updated by $\mathcal{GNN}$ through learned representations, i.e.,

$$\mathcal{G}'^{(l)} = \underset{\boldsymbol{h}_v, \boldsymbol{h}_e}{\mathcal{GNN}}(\mathcal{G}^{(l-1)}) \tag{3}$$

where $\boldsymbol{h}_v$ and $\boldsymbol{h}_e$ are the node and edge embeddings, respectively. Note that the superscript $*^{(l)}$ denotes the function or embedding at the lth layer of the GNN hereafter. In the message passing scheme, messages are first generated by a generation function with

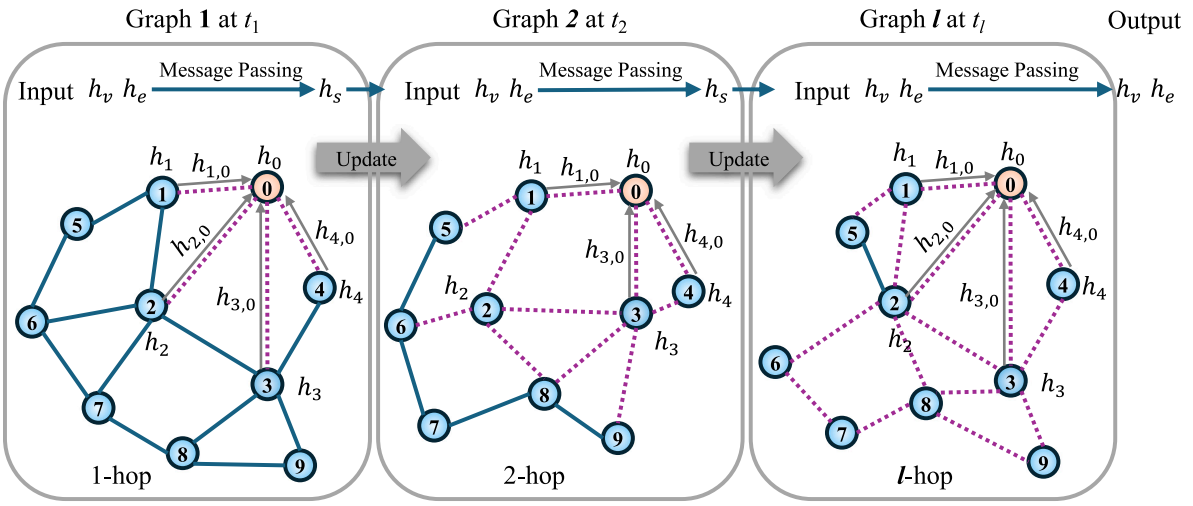

Fig. 3. Temporal graph neural networks: dynamic graphs at different layers. The dashed edges denote the information used between node 0 and its k-hop neighbors.

node and edge embeddings $(\boldsymbol{h}_w, \boldsymbol{h}_v, \boldsymbol{h}_e)$ as input, where $\boldsymbol{h}_w$ is the embedding of the neighbor of node v, then aggregated to update the node embedding. Mathematically, the node embedding $\boldsymbol{h}_v^{(l)}$ is given by

$$\boldsymbol{h}_v^{(l)} = \boldsymbol{f}_{\boldsymbol{\theta}}^{(l)}\Big(\boldsymbol{h}_v^{(l-1)}, \bigoplus_{w\in\mathcal{N}(v)} \boldsymbol{g}_{\boldsymbol{\theta}}^{(l)}\big(\boldsymbol{h}_w^{(l-1)}, \boldsymbol{h}_v^{(l-1)}, \boldsymbol{h}_e^{(l-1)}(w,v)\big)\Big) \tag{4}$$

where $\boldsymbol{f}_{\boldsymbol{\theta}}$, $\bigoplus$ and $\boldsymbol{g}_{\boldsymbol{\theta}}$ are the embedding update function, the aggregation function and the message generation function, respectively. All the three functions are differentiable to facilitate the auto differentiation. Specifically, $\bigoplus$ is permutation invariant such that it does not depend on the arbitrary ordering of aggregated items, and the candidate functions include summation, mean and maximum; $\boldsymbol{f}_{\boldsymbol{\theta}}$ and $\boldsymbol{g}_{\boldsymbol{\theta}}$ can be represented by neural networks such as multi layer perceptrons (MLPs). The edge embedding $\boldsymbol{h}_e^{(l)}$ is updated by the message generation function, i.e.,

$$\boldsymbol{h}_e^{(l)}(w,v) = \boldsymbol{g}_{\boldsymbol{\theta}}^{(l)}\big(\boldsymbol{h}_w^{(l-1)}, \boldsymbol{h}_v^{(l-1)}, \boldsymbol{h}_e^{(l-1)}(w,v)\big). \tag{5}$$

2.3. Temporal graph neural network

Temporal graph neural networks (TGNNs) have been attracting increasing attention in deep learning for dynamic graphs [31]. There are many variants of TGNNs that have been proposed for different applications (see a recent review [32]). However, there is no such a specific TGNN designed for learning particle-based simulation data, especially for granular materials. For a given simulation rollout, the configuration of material points varies over time such that the corresponding interactions among material points can be represented by a temporal graph,

$$\mathcal{G}^t = (\mathcal{V}, \mathcal{E}, \mathcal{V}^t, \mathcal{E}^t) \tag{6}$$

where $\mathcal{V}^t$ and $\mathcal{E}^t$ are time-dependent nodes and edges, respectively. For example, either node insertion events or node deletion events will be triggered when there are material points adding into or removing from the simulation domain. Most often, the amount of material points remains fixed for simplicity, while the configuration of material points may vary significantly during the course of a simulation. As a result, one has to consider the change of graphs (i.e., $\mathcal{E}^t$) due to the distinct configurations of material points between two time instants. Here we propose a novel temporal graph neural network (TGNN) for our deep learning-based simulator, coined as Temporal Graph Neural Network-based Simulator (TGNNS).

Based on an assumption that particle dynamics at time t can be determined by a limited history across a duration of t_h, it is possible to select a finite set of discrete time instants for capturing the history of particle dynamics during the continuous duration t_h. Therefore, a discrete set of time instants $\mathcal{T} = \{i\Delta t \parallel i = 0, 1, 2, \dots, n; n\Delta t \le t_h\}$ with a constant interval Δt can be one of the solutions for simplicity. Therefore, a neural network, i.e., our TGNN, can be used to learn the history-dependent particle dynamics based on the discrete history. Specifically, a graph $\mathcal{G}_t$ of configuration of material points at a time instant $t \in \mathcal{T}$ is employed to capture such particle dynamics. As shown in Fig. 3, we introduce a node historical state $\boldsymbol{h}_s^{(t)}$ to store the embedding information of particle dynamics at each time instant t,

$$\boldsymbol{h}_s^{(t)} = \boldsymbol{f}_{\boldsymbol{\alpha}}^{(t)}\Big(\boldsymbol{h}_v^{(t)}, \bigoplus_{w\in\mathcal{N}(s)} \boldsymbol{g}_{\boldsymbol{\theta}}^{(t)}\big(\boldsymbol{h}_w^{(t)}, \boldsymbol{h}_v^{(t)}, \boldsymbol{h}_{e0}^{(t)}(w,v)\big)\Big) \tag{7}$$

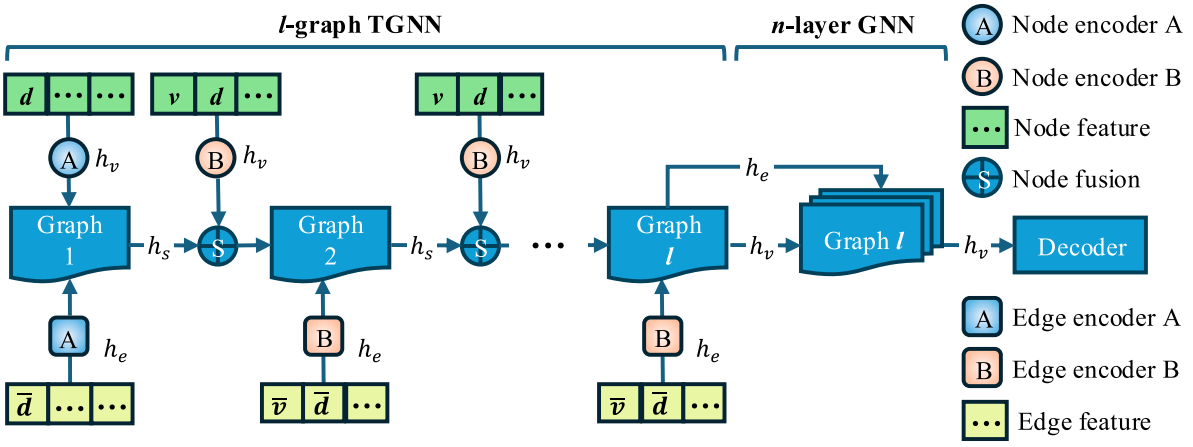

Fig. 4. A systematic workflow for predicting particle dynamics in the TGNNS framework.

with

$$\boldsymbol{h}_v^{(t)} = \mathcal{S}_{\boldsymbol{\gamma}}^{(t)}(\boldsymbol{h}_s^{(t-\Delta t)}, \boldsymbol{h}_{v0}^{(t)}) \tag{8}$$

where $\boldsymbol{h}_{v0}^{(t)}$ and $\boldsymbol{h}_{e0}^{(t)}$ are the input embedding for node and edge features at t, respectively; $\boldsymbol{f}_{\boldsymbol{\alpha}}^{(t)}$, $\boldsymbol{g}_{\boldsymbol{\theta}}^{(t)}$ and $\mathcal{S}_{\boldsymbol{\gamma}}^{(t)}$ are multi-layer perceptrons (MLPs) with learnable parameters $\boldsymbol{\alpha}$, $\boldsymbol{\theta}$ and $\boldsymbol{\gamma}$, respectively; and $\boldsymbol{h}_v^{(t)}$ and $\boldsymbol{h}_w^{(t)}$ are the node embedding after considering the historical state and input embedding (w denotes the neighbor of node v). Note that no historical state $\boldsymbol{h}_s$ is applied to the first graph, i.e.,

$$\boldsymbol{h}_v^{(t1)} = \boldsymbol{h}_{v0}^{(t1)}. \tag{9}$$

Compared with the conventional GNN, only the node embedding at the current graph or layer passes into the next one in the TGNN. While the updated edge embedding is dropped out, new node and edge features are fed into the next graph or layer. By sequentially inputting historical node and edge features, the information flow is more interpretable through the entire network.

3. Temporal graph network simulator

3.1. TGNNS framework

As shown in Figs. 2 and 3, both GNN and TGNN require node and edge features as input embedding, and the output is also in the embedding space. Therefore, both encoder and decoder are necessary to encode the feature and decode the embedding, respectively. Indeed, researchers have proposed an encoder–processor–decoder framework for learning particle-based simulation data [23]. They coined it as graph network simulator (GNS), where the processor was built on the conventional graph neural networks.

Fig. 4 shows the workflow of our proposed TGNNS framework. The framework has two major components: one is an l-layer TGNN, and the other is a conventional n-layer GNN. All node and edge features are input into the TGNN layer by layer, while the GNN is just for further message passing of the last graph $\boldsymbol{l}$ of the TGNN if required. Specifically, in the TGNN, we have four different encoders: node encoder A, node encoder B, edge encoder A and edge encoder B. Basically, we can have a single encoder for node/edge features, while the encoders for the first layer are isolated from others by considering that the particle historical dynamics is manually cutoff at the first layer (i.e., no historical information for the first layer). Moreover, for the layer with node historical state $\boldsymbol{h}_s$, a neural network (node fusion) is employed to fuse the new input node embedding $\boldsymbol{h}_v$ and the node historical state $\boldsymbol{h}_s$ (see Eq. (8)). Note that the output of TGNN can be directly as input of the decoder in absence of the n-layer GNN.

Remarks: Compared with the GNS framework [23,24], we have delineated several potential advantages of our TGNNS framework as outlined below:

- *Utilizing Dynamic Graphs*: The utilization of dynamic graphs within the TGNN in lieu of static graphs in traditional GNN aligns more coherently with the inherently dynamic nature of granular flow snapshots.
- *Inter-Graph Message Passing*: Within the context of inter-graph message passing, we deliberately exclude edge information prior to its traversal to the subsequent graph. This deliberate operation ensures that inter-graph information flows exclusively through nodes, mimicking the information propagation characteristic of Lagrangian particle methods like MPM. While we refrain from explicitly labeling the information carried on nodes as particle dynamics due to its latent nature, we can affirm that this latent information is intricately linked to particle dynamics, thus enhancing the physical interpretability of information flow.

- *Addressing Neighbor Expansion Challenges*: A persistent challenge in the GNN is neighbor expansion, significantly impeding their application on large-scale graphs despite existing mitigation techniques that often involve accuracy trade-offs [33]. This challenge equally affects the GNS. Conversely, our novel architecture presents a potential alternative for modeling large-scale simulations of granular materials. Firstly, by discarding inter-graph edge information, nodes in the subsequent graph exhibit reduced reliance on neighbors from the preceding graph. Secondly, the introduction of new data at each subsequent graph aids in mitigating the adverse effects stemming from information loss, particularly neighbor information, in the previous graph. These pivotal features not only distinguish our architecture but also offer compatibility with complementary techniques such as graphSAGE [33], GAS [34] and ClusterGNN [35], thereby facilitating large-scale simulations.

Regarding the model interpretability, our proposed TGNNS diverges from conventional physics-informed methods like Physics-Informed Neural Networks (PINN) [25] by embracing a novel approach—a physical-information-flow constraint. This innovation centers on regulating the information flow, particularly the dynamics of particles in latent space, through inter-graph message passing. Notably, we deliberately exclude edge information before its traversal to subsequent graphs, ensuring that inter-graph information exclusively traverses through nodes. This strategic choice mirrors the information propagation patterns observed in Lagrangian particle techniques such as MPM. While we abstain from explicitly labeling the node-carried information as particle dynamics due to its latent nature, we affirm its intimate connection to particle dynamics, thereby augmenting the interpretability of information flow [27].

3.2. Graph construction

For each snapshot in a simulation (referred to as a frame), a graph is constructed using the positions of particles. To account for the decrease in interaction between particles with increasing distance, an influencing radius, denoted by r, is used to limit the number of 1-hop neighbors and create sparse neighbor sets, as shown in Eq. (2). This approach is similar to the use of an influencing horizon in SPH and a cutoff range in molecular dynamics. In MPM, the domain of the shape function can be employed to define the influencing radius. Typically, the shape function in MPM does not extend beyond four cells, thus a value of twice the cell size is used for the influencing radius, i.e., $r = 2l$, where l is the cell size in MPM.

3.3. Node features and encoders

Potential node features for material points or particles can encompass any associated properties, however, redundancy within these features may impede the training process. This is due to the fact that it can make convergence more challenging and even result in degradation of model performance. For example, the incorporation of particle positions into node features has been proven to have negative effects on model performance [36]. To mitigate this issue, previous literature [23,24] has opted to exclude positions from node features, while including particle velocity $\boldsymbol{v}_p$ and distance $\boldsymbol{d}_p$ from four/six (2D/3D) boundaries. Each boundary can be viewed as a particle, allowing for the influencing range to remain applicable to particle–boundary interactions. Unlike graph construction, we do not eliminate particle–boundary pairs that exceed a certain distance. Instead, we assign a placeholder value of zero to this feature for simplicity, as reported in [23]. Notably, the boundary can be treated as a unique particle, and it is feasible to introduce another feature, such as particle type, to distinguish the boundary from other normal particles. This approach has been successfully implemented in the GNS [37].

With the proposed node features, we can define the node embedding $\boldsymbol{h}_v^{(t)}$ as follows:

$$\boldsymbol{h}_v^{(t=0)} = \mathcal{F}_{n0}(\boldsymbol{v}_p, \boldsymbol{d}_p) \tag{10}$$

$$\boldsymbol{h}_v^{(t\neq 0)} = \mathcal{F}_{n1}(\boldsymbol{v}_p, \boldsymbol{d}_p) \tag{11}$$

where $\mathcal{F}_{n0}$ and $\mathcal{F}_{n1}$ are the functions for the encoders at the first graph and subsequent graphs, respectively, represented as MLPs.

3.4. Edge features and encoders

The end nodes connected to the edges will inherit the information of their edge features, which are formulated in Eqs. (4) and (7). The candidate edge features are such potential properties that can influence inter-particle interactions. Specifically, we adopt the relative velocity $\bar{\boldsymbol{v}}_p$ and displacement $\bar{\boldsymbol{d}}_p$ as edge features. The displacement feature comprises the unit direction and normalized norm, where the norm is normalized by the influencing radius. Consequently, we define the edge embedding $\boldsymbol{h}_e^{(t)}$ as follows:

$$\boldsymbol{h}_e^{(t=0)} = \mathcal{F}_{e0}(\bar{\boldsymbol{v}}_p, \bar{\boldsymbol{d}}_p) \tag{12}$$

$$\boldsymbol{h}_e^{(t\neq 0)} = \mathcal{F}_{e1}(\bar{\boldsymbol{v}}_p, \bar{\boldsymbol{d}}_p) \tag{13}$$

where $\mathcal{F}_{e0}$ and $\mathcal{F}_{e1}$ are the encoder functions for the first and subsequent graphs, respectively, represented as MLPs.

Table 1
MLPs in the TGNNS.

MLP	Num.	Layer size/number			
		Input	Hidden layer size & #	Output	NormLayer
$\mathcal{F}_{n0}$	1	$2n_d$	128 & n_{hl}	128	128
$\mathcal{F}_{n1}$	1	$3n_d$	128 & n_{hl}	128	128
$\mathcal{F}_{e0}$	1	$n_d + 1$	128 & n_{hl}	128	128
$\mathcal{F}_{e1}$	1	$2n_d + 1$	128 & n_{hl}	128	128
$\mathcal{S}_r$	1	128*2	128 & n_{hl}	128	128
$\boldsymbol{g_\theta}$	$n_{graph}^{TGN} + n_{msg}^{GNN}$	128*3	128 & n_{hl}	128	128
$\boldsymbol{f_\alpha}$	$n_{graph}^{TGN} + n_{msg}^{GNN}$	128*2	128 & n_{hl}	128	128
$\mathcal{F}_a$	1	128	128 & n_{hl}	n_d	–

Note: n_d - problem dimension (2D/3D); n_{hl} - number of hidden layers; n_{graph}^{TGN} - number of graphs; n_{msg}^{GNN} - message passing steps in the GNN.

3.5. Message passing

The conventional message passing scheme in GNNs involves passing node and edge embeddings within a fixed structured graph, which we refer to as intra-graph message passing. As depicted in Fig. 4, this occurs at each graph that may have multiple layers. In the proposed TGNN, there is also the added complexity of inter-graph message passing, which involves passing messages between varying graphs over time. Specifically, for each layer of the TGNN, the message passes n_{msg}^{TGN} steps with shared parameters in the node embedding update function $\boldsymbol{f}_\theta$ and the message generation function $\boldsymbol{g}_\theta$, while these functions do not share parameters across the GNN layers with n_{msg}^{GNN} steps. To handle the inter-graph message passing, the output node embedding from the previous graph $\boldsymbol{h}_s$ is fused with the new node embedding $\boldsymbol{h}_v$ of the current graph using a function $\mathcal{S}_\gamma$ prior to being passed into the current graph. It is worth noting that in contrast to intra-graph message passing, the inter-graph message passing does not include the output edge embedding of the previous graph. This implementation is based on several considerations: (1) the output node embedding of the previous graph is expected to represent the dynamic information carried by each material point, similar to the approach used in Lagrangian particle methods; (2) passing the edge embedding from the previous graph to the current one would require more implementation due to edge changes; and (3) passing the edge embedding would not significantly improve model performance compared to node embedding passing, as shown in Section 5.2 later.

3.6. Decoder

The node decoder is responsible for decoding the particle dynamics $\boldsymbol{h}'_v$ learned from the TGNNS network. Specifically, the decoder $\mathcal{F}_a$ is designed to output the particle acceleration $\boldsymbol{a}_p^t$ as follows:

$$\boldsymbol{a}_p^t = \mathcal{F}_a(\boldsymbol{h}'_v) \tag{14}$$

Previous studies [23] indicate that decoding particle acceleration yields superior model performance compared to decoding particle position. One possible reason is that we can hard-code the integration scheme into the neural network, making it more physics-informed. With the particle acceleration $\boldsymbol{a}_p^t$ as input, the Euler integration scheme can be used to predict the particle velocity $\boldsymbol{v}_p^{t+\Delta t}$ and position $\boldsymbol{r}_p^{t+\Delta t}$:

$$\boldsymbol{v}_p^{t+\Delta t} = \boldsymbol{v}_p^t + \boldsymbol{a}_p^t \Delta t \tag{15}$$

$$\boldsymbol{r}_p^{t+\Delta t} = \boldsymbol{r}_p^t + \boldsymbol{v}_p^{t+\Delta t} \Delta t \tag{16}$$

Here, Δt represents the time interval between two frames. In the TGNNS, an MLP is utilized to represent the decoder $\mathcal{F}_a$.

3.7. Implementation

The prevailing machine learning library PyTorch [38] and its graph neural network library PyG [39] are adopted to implement the TGNNS. For enhanced training, we leverage the Automatic Mixed Precision package for automatic mixed precision training and the Distributed Data Parallel package for multi-GPU parallel training. All MLPs used in the TGNNS are listed in Table 1. The default setting for the latent dimension is 128. We utilize the LeakyReLU activation function with a negative slope of 0.1. Moreover, we apply layer normalization [40] after the output layer, as needed, to normalize the output embedding.

4. Data, training and inference

In this section, our proposed TGNNS will be trained to learn simulation data from hierarchical multiscale modeling of column collapse tests of granular materials.

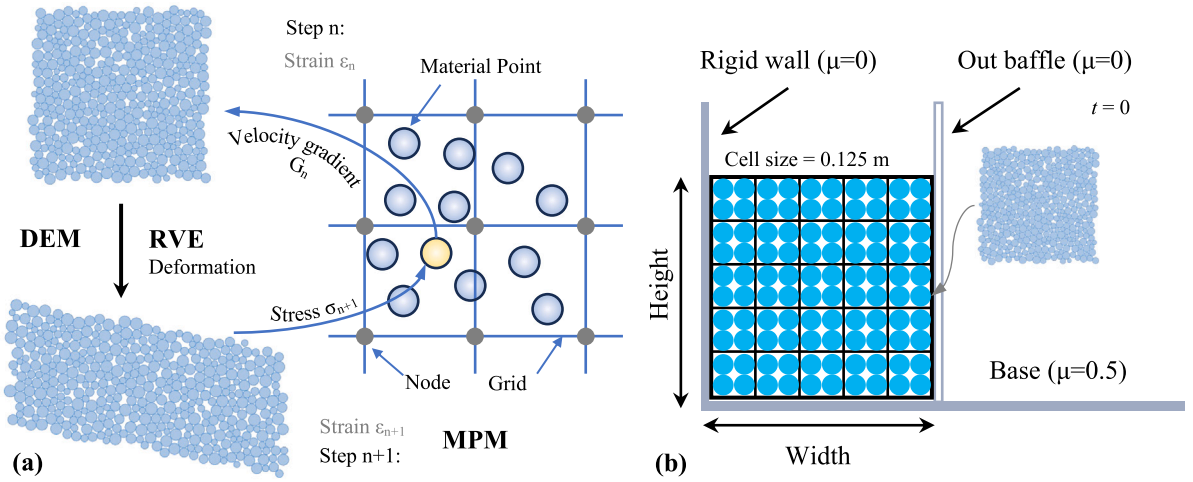

Fig. 5. Illustrations of: (a) hierarchical multiscale DEMPM; (b) column collapse tests.

4.1. Hierarchical multiscale modeling of column collapse

Hierarchical multiscale modeling approaches couple continuum-based methods, such as the finite element method (FEM) and the material point method (MPM), with micromechanics-based methods, such as the discrete element method (DEM). Two examples of such approaches are FEM-DEM [14] and MPM-DEM [15]. In this work, we choose the hierarchical MPM-DEM approach to generate datasets due to its superior capability in modeling large-deformation problems, such as column collapse tests. In this approach, as illustrated in Fig. 5(a), each material point is associated with an exclusive DEM simulated representative volume element (RVE). The mechanical response of each RVE provides a stress–strain relation to MPM, bypassing the need for conventional continuum-based constitutive models.

To simulate the column collapse tests, we use the efficient tool DEMPM, which leverages thread-block-wise parallelism of RVEs on GPUs [41]. DEMPM has been validated against experimental data reported in the literature [42]. As shown in Fig. 5(b), a granular column is first generated in MPM with a cell size of 0.125 m and four material points per cell. Each material point is attached a DEM RVE assembly composed of 400 particles with size uniformly ranging from 2.5 mm to 5 mm. The linear spring contact model is adopted in conjunction with the Coulomb sliding friction model in DEM. Specifically, the normal and tangential contact stiffnesses are set to 1×10^5 N/m, and interparticle coefficient of friction is set to 0.5. Prior to triggering collapse by removing the boundary constraints, e.g., the right out baffle in the figure, the granular column will be consolidated into an equilibrium state under gravity.

4.2. Datasets

4.2.1. Simulation

A series of simulations for column collapse tests is conducted with different initial configurations. Subtly different from a standard column collapse test as illustrated in Fig. 5(b), all granular columns are set up with the same size but different initial positions and velocities to obtain more trajectories of granular flows. As shown in Fig. 6(a), a granular column has an initial stress due to its self-weight under gravity, and all granular columns have the same vertical stress (stress YY) as illustrated in the inset of Fig. 6(a). The simulation of column collapse starts by removing the constraints confining the granular column. We run 25 simulations for the training datasets and 5 simulations for the validation datasets. Each simulation has 2304 material points and 921 600 DEM particles. The detailed initial positions and velocities for those simulations are plotted in Fig. 6(b), where each elliptic shape corresponds to a simulation (black edge for the training, and red edge for the validation). The center location of an elliptic shape denotes the initial position, i.e., the location of the bottom-left corner of a granular column. The axis length of an elliptic shape is proportional to the corresponding velocity component (v_x or v_y), and the filling color is for the magnitude of velocity. Note that the initial velocity is applied to all material points of the granular column. Table 2 summarizes the simulation configurations for TGNNS datasets.

4.2.2. Input sequence of particle positions

In the TGNNS, the only required simulation data is particle positions at each frame, while the other kinematics data including particle velocities and accelerations can be estimated based on the sequence of particle positions. Specifically, given particle positions $\boldsymbol{r}_p^t$ at time t, $\boldsymbol{r}_p^{t-\Delta t}$ at time $t-\Delta t$ and $\boldsymbol{r}_p^{t-2\Delta t}$ at time $t-2\Delta t$, the forward Euler difference gives particle velocity $\boldsymbol{v}_p^t$ and acceleration $\boldsymbol{a}_p^t$ as follows:

$$\boldsymbol{v}_p^t = \frac{\boldsymbol{r}_p^t - \boldsymbol{r}_p^{t-\Delta t}}{\Delta t} \tag{17}$$

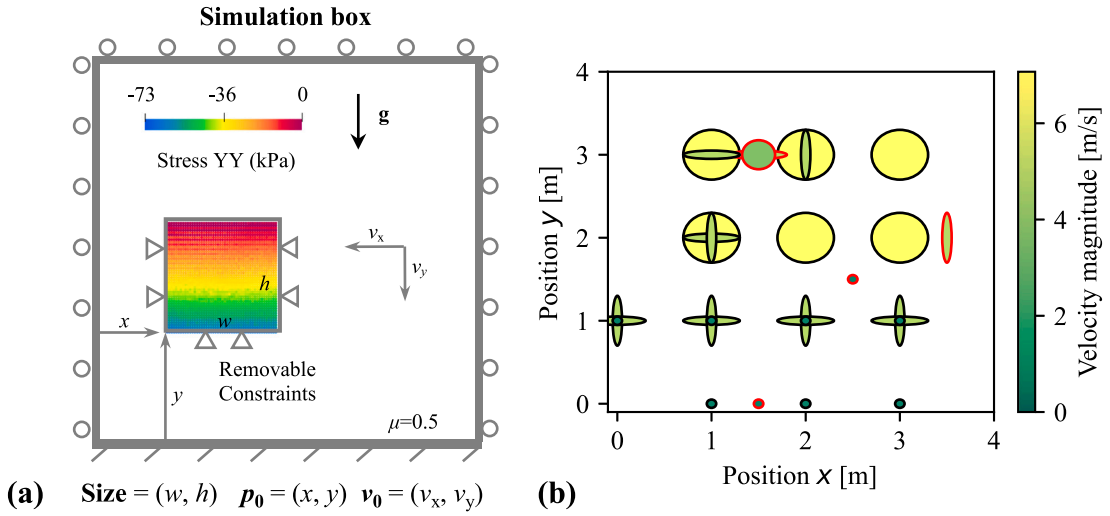

Fig. 6. (a) Initial configuration of a granular column with specific size, position and velocity within a 10 m-by-10 m simulation box; (b) initial positions and velocities of 3 m-by-3 m granular columns for training (black edge) and validation (red edge) datasets. Note: the x-axis (or y-axis) length of an elliptical scatter is proportional to v_x (or v_y), and the filling color is for the magnitude of velocity. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Simulation configurations for training and validation datasets.

Granular column simulation setup	Train	Validation				
		Case 1	Case 2	Case 3	Case 4	Case 5
Position (x, y) [m]	$x, y \in (0, 1, 2, 3)$	(1.5, 0)	(2.5, 1.5)	(3.5, 2)	(1.5, 3)	(1.5, 3)
Velocity (v_x, v_y) [m/s]	$v_x, v_y \in (0, 5)$	(0, 0)	(0, 0)	(0, 5)	(2.5, 2.5)	(5, 0)
Size (w, h) [m]	(3, 3)			(3, 3)		
Time steps	30 000			30 000		
Rollout #	25			5		

Note that the detailed combination of position and velocity for both the training dataset can be found in Fig. 6(b).

$$\boldsymbol{a}_p^t = \frac{\boldsymbol{v}_p^t - \boldsymbol{v}_p^{t-\Delta t}}{\Delta t} = \frac{\boldsymbol{r}_p^t - 2\boldsymbol{r}_p^{t-\Delta t} + \boldsymbol{r}_p^{t-2\Delta t}}{\Delta t^2} \tag{18}$$

where Δt is the interval time between two frames. Data augmentation is also adopted by introduced noises into the input sequence of particle positions during training.

4.2.3. Normalization

The normalization is an important pre-processing for preparing training, validation and testing data in machine learning. One outstanding benefit is that we can make the ranges of different features (such as velocity and position) into a similar range (e.g., [−1, 1]), facilitating the matrix operation (that makes sure the result not dominated by one of those features). We standardize each dimension of particle velocity v_i' and acceleration a_i' using the z-score normalization method, i.e.,

$$v_i' = \frac{v_i - \bar{v}}{\sigma_v} \tag{19}$$

$$a_i' = \frac{a_i - \bar{a}}{\sigma_a} \tag{20}$$

where $\bar{v}$ and $\bar{a}$ represent the mean velocity and mean acceleration, respectively, and σ_v and σ_a represent their corresponding standard deviations. We rescale the size of the simulation box to a dimensionless 1×1. It is worth noting that this scaling has no impact on the input features of TGNNS, thanks to the standardization process employed.

4.2.4. Multi-resolution sampling

The recovery of historical particle trajectory relies heavily on the cohesion between two neighboring sampled frames. If there is a large history gap between the two frames, a machine learning model may fail to learn the underlying physics behind the particle trajectory due to information loss. In the case of physics-based simulation data, we can sample different datasets with varying time step intervals, as illustrated in Fig. 7. Initially, the simulation data is characterized by sequential numbering corresponding to the time steps (e.g., ranging from 0 to 29,999 for a simulation spanning 30,000 time steps as depicted in Fig. 7a). Subsequently, a

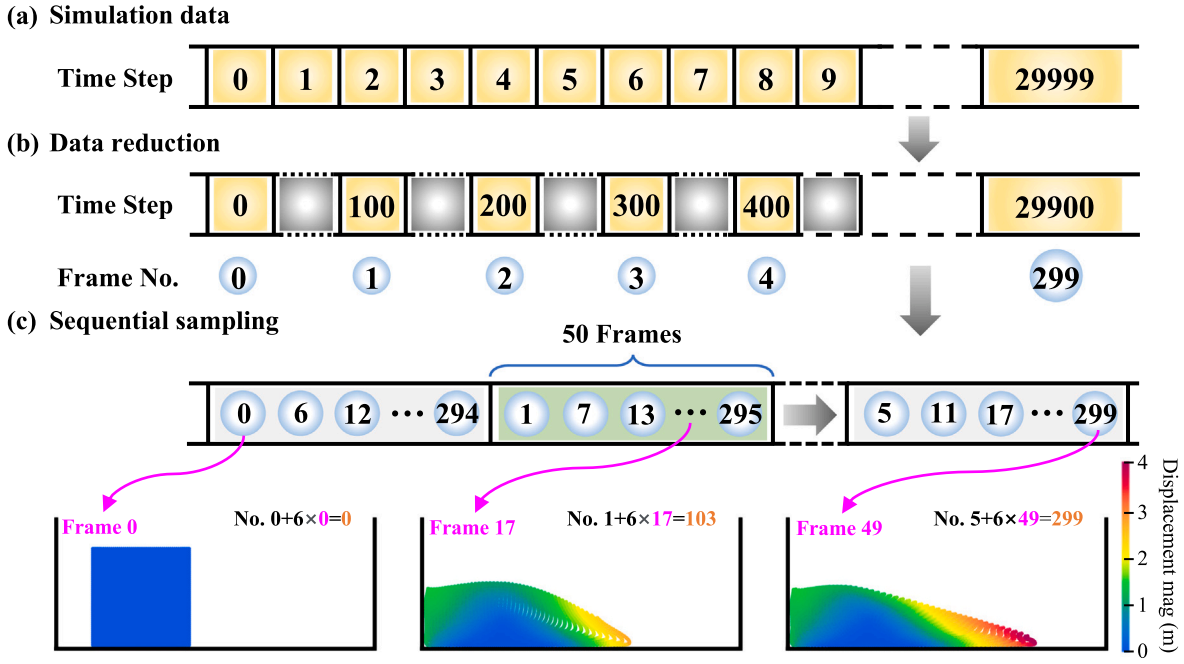

Fig. 7. Illustration of particle position input sequences with varying sampling intervals.

Table 3
Datasets with different resolutions.

	Simulation	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5
Sample step[a]	–	100	200	300	600	1000
Frame #	30 000	300	150	100	50	30

[a] The term *sample step* refers to the frequency at which frames are added to a dataset, with each frame being sampled at regular intervals (time steps).

trajectory is constructed by sampling the simulation data at regular time step intervals (e.g., sampling every 100 time steps as shown in Fig. 7b). This trajectory can then be segmented into a series of sub-trajectories, each comprising a reduced number of frames. For instance, a 300-frame trajectory can be partitioned into 6 sub-trajectories, each consisting of 50 frames, with frames within each sub-trajectory sampled at intervals of 6 frames (refer to Fig. 7c). As shown in the figure, the particle configuration at each frame can vary significantly, resulting in diverse graphs. To explore the performance of TGNNS in coarse sampling datasets, five datasets with different sampling resolutions are prepared as summarized in Table 3.

4.3. Training and inference

For each dataset in Table 3, a model will be trained for a given resolution. Dataset 1 will be used to train the reference model with the highest resolution, which will subsequently be used for hyperparameter tuning. The mini-batch gradient descent method will be applied to train the model with a default batch size of 2. For most of the training, two GPUs are used for parallel training, and the effective batch size will be 4. The parameter updating will be done using the Adam optimizer [43] without weight decay. The learning rate α_i will be set to exponentially decay with training step i as $(10^{-4} - 10^{-6}) \times 0.1^{\frac{in_d}{5\times10^6}} + 10^{-6}$, where n is the number of GPUs. The training process aims to minimize the deviation in particle acceleration between the output $\boldsymbol{a}_i$ and target $\boldsymbol{a}_i^*$. The deviation is quantified as a loss function using relative mean squared error (reMSE) in Eq. (21):

$$loss = \frac{\sum(\boldsymbol{a}_i - \boldsymbol{a}_i^*)^2}{\sum(\boldsymbol{a}_i^*)^2} \tag{21}$$

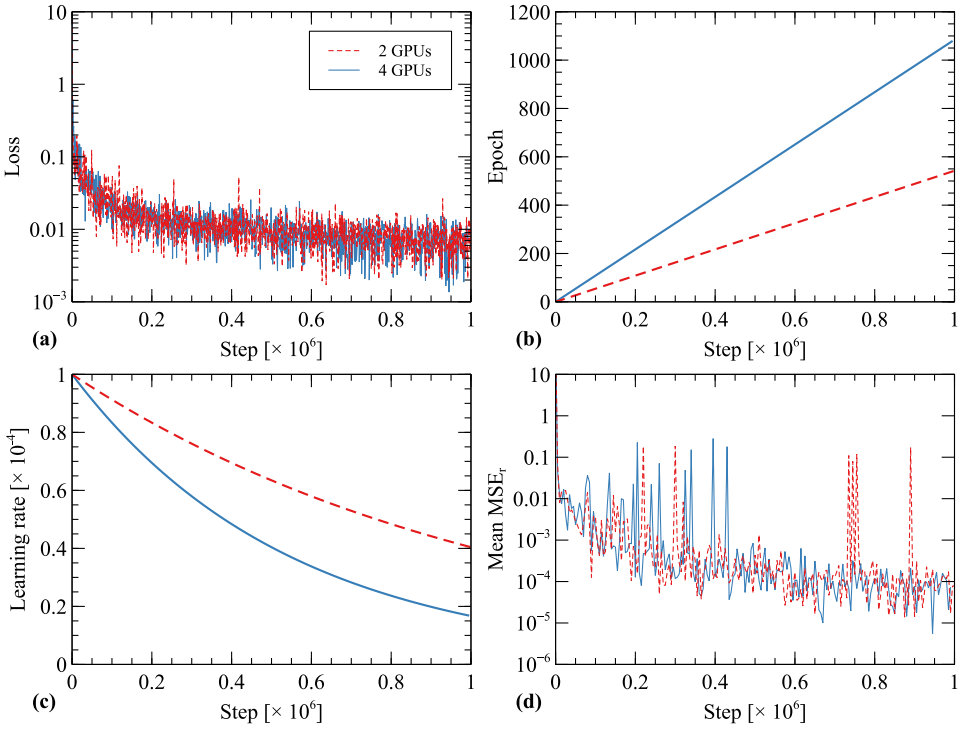

Fig. 8. Variation of metrics during training and validation.

For validation and testing, the inference error is quantified using the mean squared error (MSE) of particle position:

$$MSE_r = \frac{1}{N}\sum_{i\in N}(\boldsymbol{r}_i - \boldsymbol{r}_i^*)^2 \tag{22}$$

where $\boldsymbol{r}_i$ and $\boldsymbol{r}_i^*$ are the inferred and ground truth particle positions (from the DEMPM simulations) in a frame, respectively. The mean over all frames in a trajectory is used to evaluate the model performance during validation instead of using the error of acceleration in Eq. (21).

5. Model evaluation

5.1. Baseline evaluation

We initially train a reference model, which will serve as a baseline for subsequent comparisons in the ablation study. The baseline model incorporates the default settings mentioned in Section 3.7, and it adopts a neural network architecture with the following specifications: (i) it consists of six input frames (referred to as a 6-layer TGN), (ii) it has two hidden layers in each MLP (see Table 1), and (iii) it includes two additional inter-graph message passing layers after the final layer of the TGNN (refer to Table 4 for a detailed summary).

Fig. 8 depicts the typical variations observed in various metrics during the training process, including loss, epoch, and learning rate, as well as the mean MSE_r for validation. Notably, the training loss exhibits a significant drop at the outset, prior to reaching 0.2×10^6 steps. Interestingly, doubling the number of GPUs to increase the effective batch size does not significantly contribute to the convergence of the loss and mean MSE_r. However, training with a larger effective batch size (by doubling the number of GPUs) can mitigate the fluctuation observed in the mean MSE_r during validation, although it may also lead to slight overfitting of the model. This claim is supported by the rollout MSE_r values for the five validation cases, depicted in Fig. 9.

Fig. 9 also illustrates the increasing trend of the rollout MSE_r with frame step, which occurs due to error accumulation. Although the MSE_r can be as small as around 10^{-10} for a single-step inference (as observed at the beginning of a rollout), the accumulated error can reach several orders of magnitude for the final configuration of a collapsed granular column. Previous studies [23,24] have also reported similar observations. Moreover, it is important to note that the dynamics of a granular column significantly impact the single-step MSE. This explains the presence of a plateau in the MSE values before reaching the final state.

To more intuitively demonstrate the cumulative error in MSE_r, snapshots of the collapsing column in Case 1 were captured at various frame steps, as depicted in Fig. 10. At Frame 50, the mean MSE_r increases significantly to approximately 10^{-6} from the initial state (Frame 0), as shown in Fig. 9. Moreover, the majority of material points exhibit an MSE_r of less than 10^{-5}, indicating

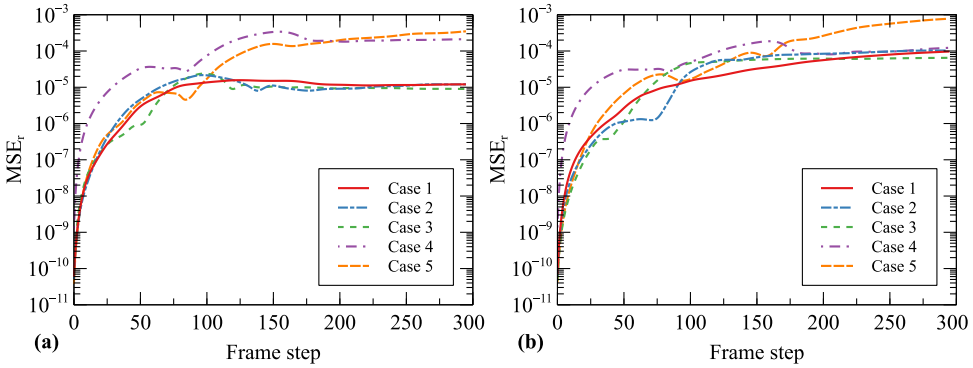

Fig. 9. Rollout MSE$_r$ for validation at step 1M: (a) 2 GPUs and (b) 4 GPUs.

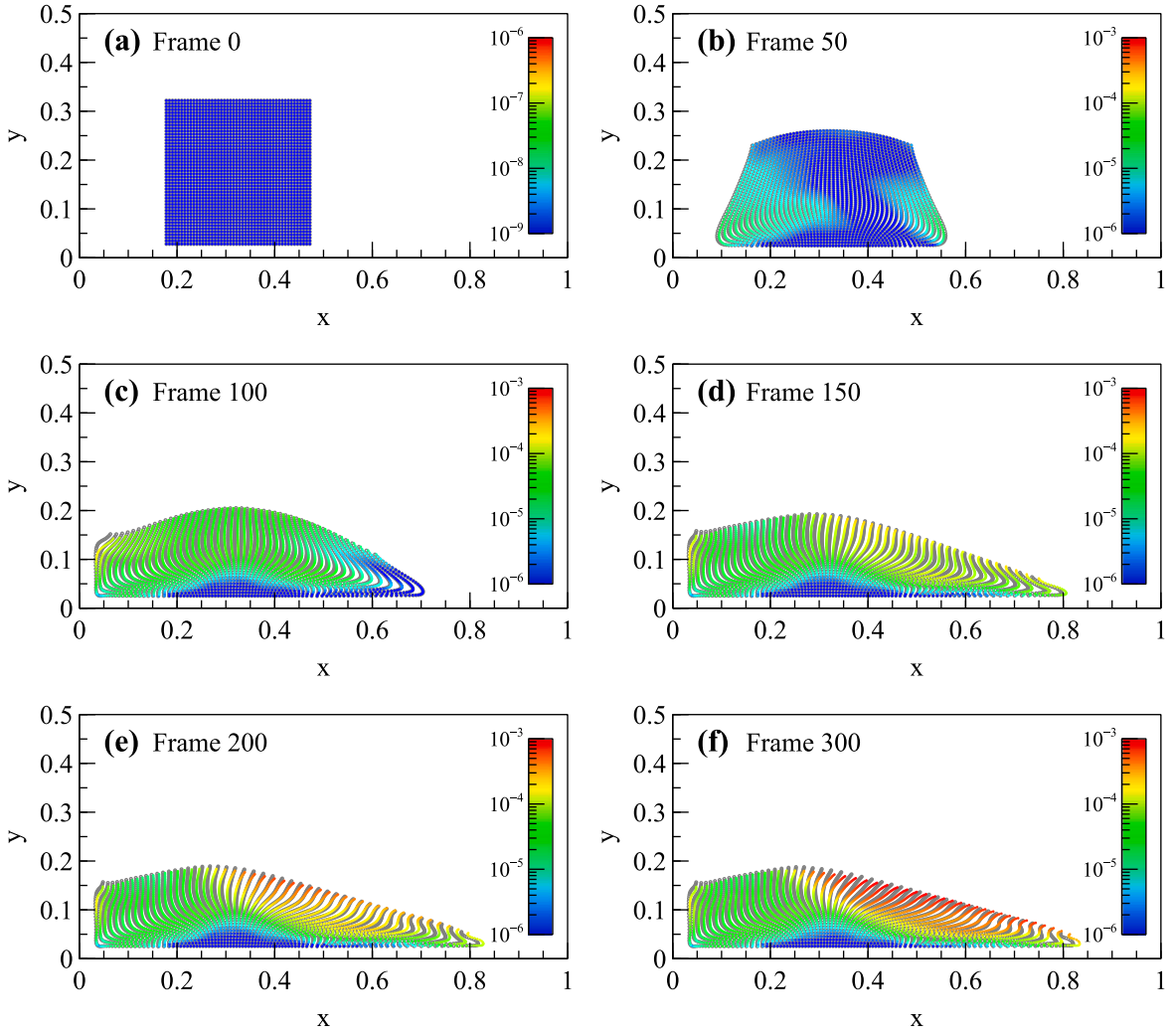

Fig. 10. Comparison of ground truth (gray) and learned (colorful) particle positions for model validation in Case 1, trained with a million steps on two GPUs. The color map represents squared displacement errors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

a good agreement with the ground truth (represented by the gray points in the background). Moving to Frame 100, the mean MSE$_r$ reaches a plateau, and the predicted column profile matches well with the ground truth, except for a moderate deviation at

Table 4
Networks used in the ablation study.

Network	Hidden layer #	Message passing[a] layer #	Input frame #	Inter-graph edge[b] message passing
Baseline	2	2	6	–
Hidden1	1	2	6	–
Msg1	2	1	6	–
Msg0	2	0	6	–
Msg0-hidden1	1	0	6	–
Frame4	2	2	4	–
Frame4-msg0-hidden1	1	0	4	–
Frame4-msg1	2	1	4	–
Frame4-hidden1	1	2	4	–
Frame4-msg1-hidden1-edgemsg[b]	2	2	4	√

[a] The message passing layer refers to the intra-graph message passing layer in the GNN following the TGNN.
[b] Similarly to the node state, inter-graph edge message passing is introduced for the network marked by '√'.

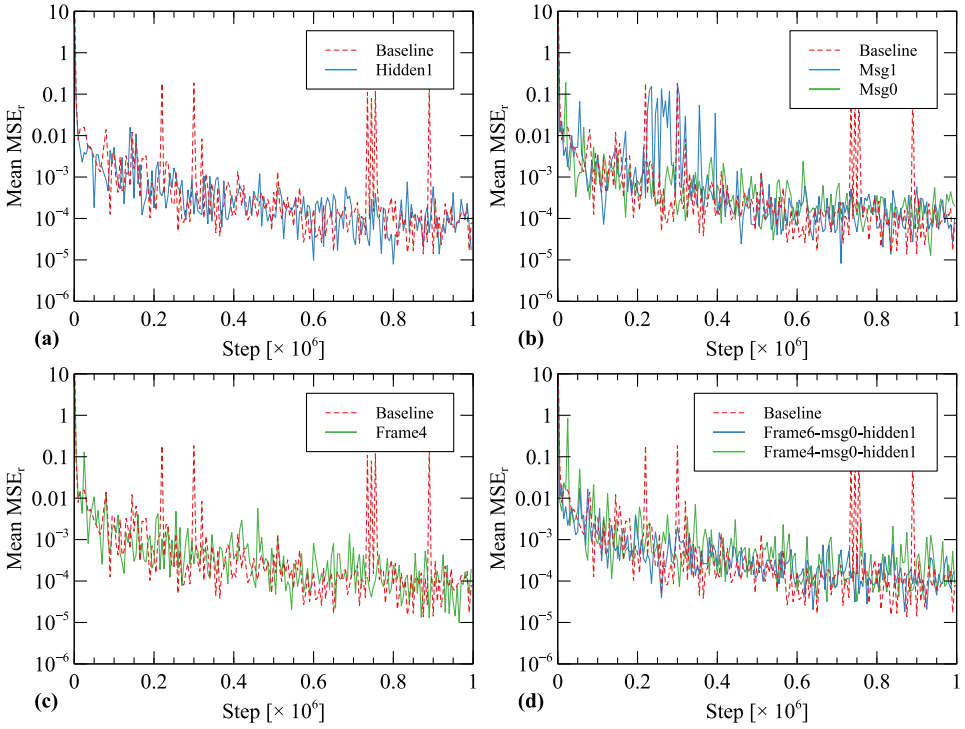

Fig. 11. Mean MSE$_r$ for validation during training with different network architectures by individually reducing (a) the number of hidden layers, (b) the number of message passing layers (msg layer) in the GNN, (c) the input frame length, and (d) reducing these components together.

the left boundary. Subsequently, from Frame 100 onward, the granular column undergoes slight deformation to approach its final equilibrium state. Starting from Frame 150, the MSE$_r$ tends to saturate, with a slight increase observed at the right surface. Finally, at the ultimate state (Frame 300), although most material points do not precisely align with the ground truth, both the overall profile and the run-out head are accurately predicted. This demonstrates the validity of the proposed TGNNS for simulating granular flows.

5.2. Ablation study

Based on the reference model (Baseline) introduced in Section 5.1, an ablation study is required to further evaluate the proposed TGNNS. This study involves reducing partial modules of the Baseline to create various network architectures, as outlined in Table 4.

Fig. 11 illustrates the variations of mean MSE$_r$ for validation across different reduced network architectures. It is evident that employing reduced network architectures with fewer learnable parameters leads to a reduction in fluctuations of MSE$_r$. This suggests that a decrease in learnable parameters within the TGNNS can enhance the model impartiality. In other words, an excess of learnable parameters can potentially result in slight overfitting on specific training data. For instance, in stochastic gradient training, overfitting may occur on input frames with lower mean particle velocities, which dominate the training data for a single trajectory

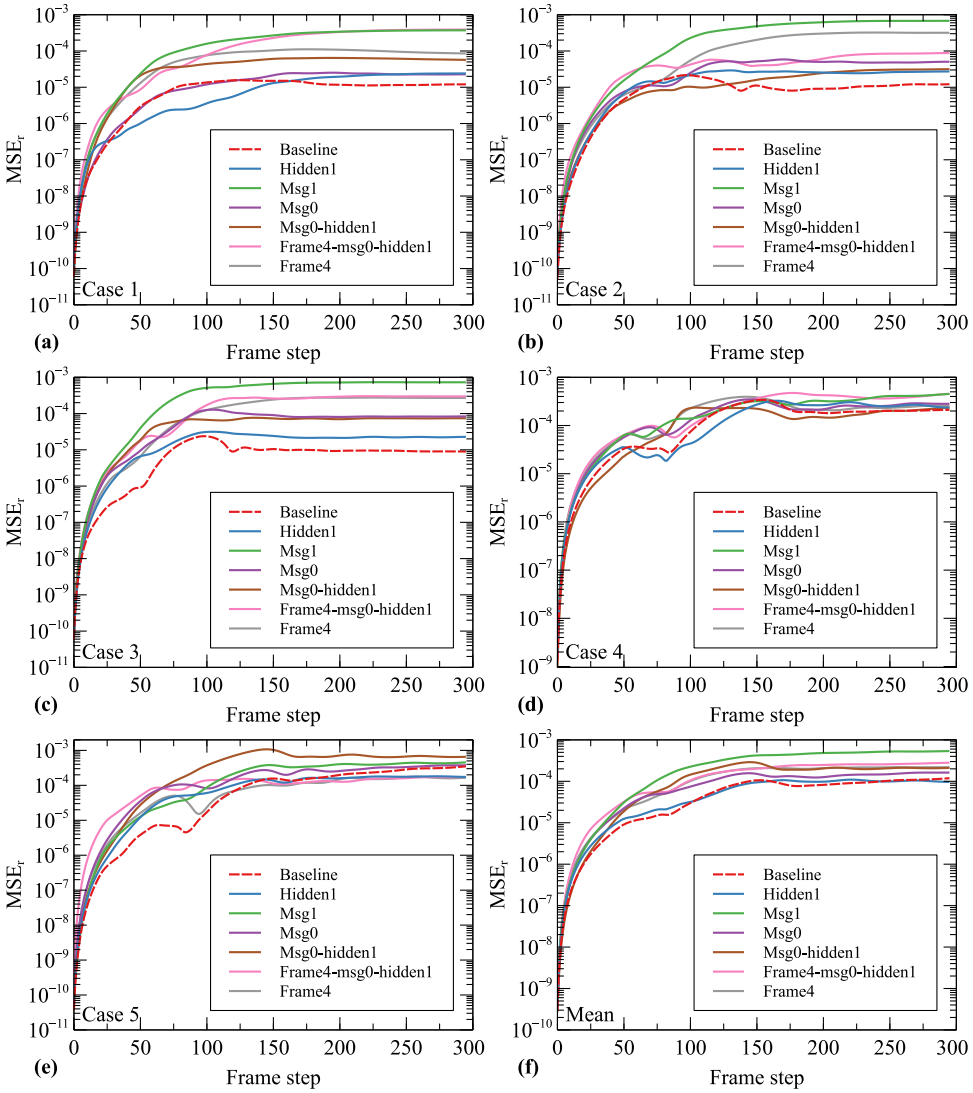

Fig. 12. Rollout MSE$_r$ for validation at step 1M: (a–e) for Case 1 to 5, and (f) for the mean over all cases.

of a granular column collapse. To address the issue of imbalanced training data, one possible approach is to explore weighted sampling techniques [44], which will be investigated in our future research. Regarding the convergence of MSE$_r$, the choice of network architecture does have some impact, although not significantly.

To assess the performance of the model with different network architectures after training for 1 million steps, we present the variation of MSE$_r$ during rollout prediction for each validation case in Fig. 12(a–e). It is evident that no single network architecture consistently exhibits the best performance across all five cases. However, the Baseline model consistently outperforms the others in most cases. Nevertheless, the network architecture (Hidden1) with one hidden layer for each MLP achieves nearly the same average performance as the Baseline model (Fig. 12(f)).

For the network architectures with reduced message passing layers, the architecture without any further message passing layers (Msg0) exhibits slightly degraded performance compared to the Baseline, which has two message passing layers. Surprisingly, the architecture with a single message passing layer (Msg1) performs the worst across most validation cases. This suggests that using more message passing layers does not always result in better model performance.

Indeed, increasing the number of message passing layers in the GNN part, without sharing parameters, leads to an increase in the number of learnable parameters. This increase in parameters can enhance the capacity of the network architecture to learn more complex patterns by propagating interactions from k-hop neighbors. However, it also inevitably hampers training convergence. Consequently, the improvement in capability observed in the network architecture Msg1 is not sufficient to balance the loss of convergence caused by the additional message passing layer.

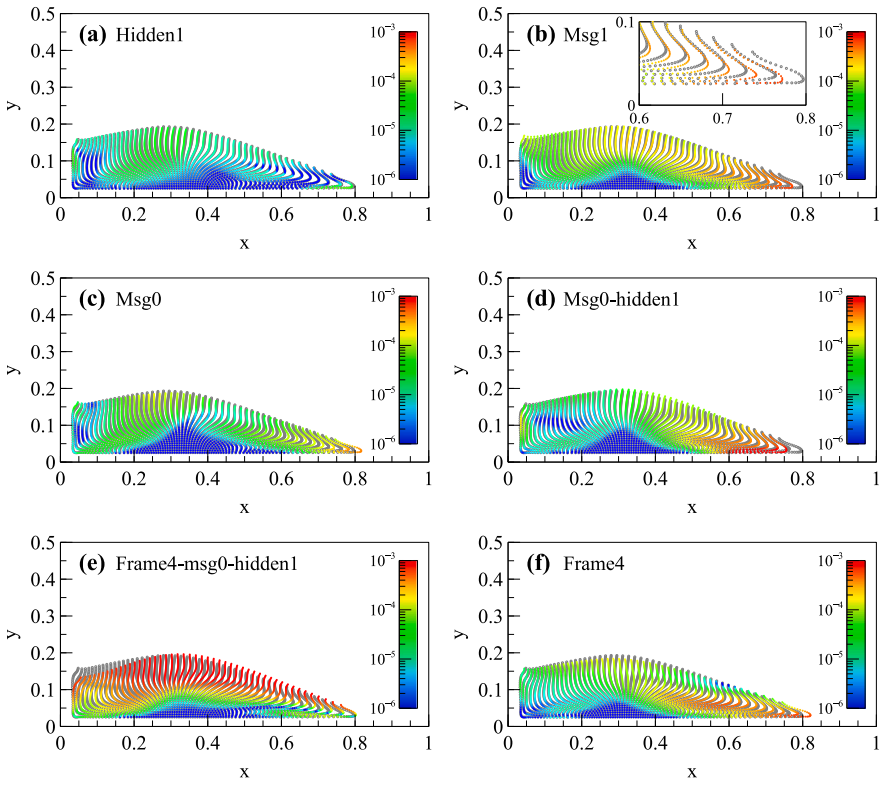

Fig. 13. Comparison of ground truth (gray) and learned (colorful) particle positions at Frame 150 of Case 1 using different ablative networks for model validation, trained with a million steps. The color map represents squared displacement errors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

When further reducing the network architecture, specifically referring to Msg0-hidden1 and Frame4-msg0-hidden1, the overall performance will not be significantly affected. In the case of the network architecture Frame4, the inter-graph message passing layers are reduced since the input frame number is reduced to 4, resulting in a decrease in overall performance. However, it is worth noting that Frame4 can exhibit superior performance compared to the Baseline model in certain scenarios, such as Case 5.

Fig. 13 shows the snapshots of the predicted rollout of Case 1 at Frame 150, using different reduced network architectures. It is evident that the predicted positions of material points align closely with the ground truth for the network architecture Hidden1. For the network architectures with reduced message passing layers (Msg0, Msg1, and Msg0-hidden1), the prediction of the overall profile matches well with the ground truth for all three network architectures, and Msg0 can predict the runout head best. Notably, there exists a significant mismatch between the predicted positions of material points from Msg1 and the ground truth, as indicated in the inset of Fig. 13(b). Consequently, this results in a substantial MSE_r in Fig. 12. When considering the network architecture Msg0-hidden1, further reducing the input frame number to 4 (Frame4-msg0-hidden1) yields the poorest performance. However, by restoring the message passing layers and hidden layers (Frame4), a more reliable prediction of the overall profile of the collapsed granular column is achieved.

We further investigate the performance of ablative network architectures in relation to Frame4. Fig. 14 illustrates the fluctuation of rollout MSE_r averaged across all validation cases, as well as the snapshots of Case 1 at the final state. It appears that the overall performance is slightly diminished when reducing the number of hidden layers or message passing layers. Moreover, it can be beneficial to incorporate edge message passing between graphs in the TGNN, but the improvement is not substantial.

To summarize, ablative network architectures, built upon our default Baseline, will experience performance degradation to some extent in most validation cases. This implies that ablative networks would benefit from additional training steps to potentially enhance model performance. While these observations may differ with varying training data, they provide a fundamental comprehension of the proposed TGNNS.

5.3. Multi-resolution models

The TGNNS is further trained with multi-resolution datasets as listed in Table 3. Fig. 15(a) and (b) show the epoch during training and the mean rollout MSE_r in validation, respectively, where Res300 denotes a resolution of 300 frames in a single trajectory, i.e., the baseline Dataset 1. With the proposed multi-resolution sampling technique, all the five datasets have almost the same data size, as

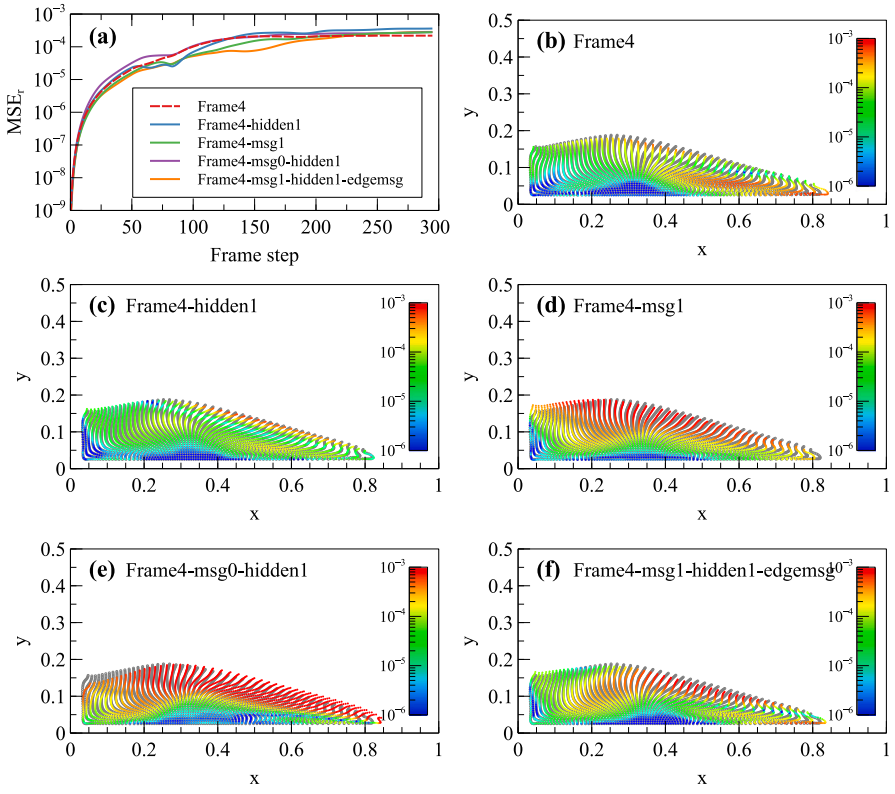

Fig. 14. Performance of ablative networks with respect to Frame4 after training one million steps: (a) Rollout MSE$_r$ averaged over all validation cases, and (b–f) comparison of ground truth (gray) and learned (colorful) particle positions of Case 1 at the final state. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

shown in Fig. 15(a), even though the sample resolution is remarkably different. The models Res300, Res150, and Res100 share similar convergence in mean MSE$_r$ for validation, while the other two models Res50 and Res30 have almost one order of magnitude larger mean MSE$_r$. The rollout MSE$_r$ averaged over all validation cases is plotted in Fig. 15(c). It is evident that the validation performance degrades as the dataset resolution decreases. Notably, the lowest-resolution dataset Res30 has a saturated MSE$_r$ 10 times greater than the baseline one (Res300), while Res300 has 10 times more frames than Res30. Those observations highlight the importance of the dataset resolution in the TGNNS.

In the numerical method MPM, a background grid is utilized to solve the partial differential equation, specifically the linear momentum equation in this study, as illustrated in Fig. 1(b). In order to achieve higher accuracy, it is necessary for the cells of the background grid to be sufficiently small. The time step is typically constrained by the classical CFL condition for explicit time integration [45]. Therefore, the displacement of material points between successive timesteps is relatively small compared to the cell size. In the TGNNS, the baseline dataset, referred to as Dataset1 (Res300), samples frames of a trajectory every 100 timesteps, resulting in a theoretical timestep scale-up of 100. Considering the satisfactory performance of the model with Res300, as discussed previously, the TGNNS can be viewed as an implicit solver with an effectively magnified timestep. However, it should be noted that while the implicit MPM remains unconditionally stable regardless of the timestep chosen, the convergence of the implicit solver will be compromised when employing a significantly larger timestep. Therefore, from a numerical solution standpoint, the magnitude of inter-timestep motion for material points cannot be arbitrarily large, even when employing an implicit solver.

Furthermore, we present histograms in Fig. 15(d) to illustrate the number of crossing cells between two frames across different resolution datasets. It is evident that the majority of material points do not cross two cells. Interestingly, for datasets Res100, Res150, and Res300, nearly no material points cross 4 cells, which corresponds to the domain size of the shape function in our MPM. In contrast, for the lowest-resolution dataset, Res30, material points can cross up to 9 cells between two frames, indicating a substantial displacement that is not acceptable for a single timestep, even when using an implicit MPM solver. This suggests a significant loss of physical information due to coarse sampling. While a deep learning-based simulator may exhibit good performance on such low-resolution datasets, it compromises the reasonability of the underlying physics. Importantly, our TGNNS is designed to enhance physical interpretability by transferring material point-carried information graph by graph. Such model-based interpretability increases descriptive accuracy through an easier-to-understand model, sometimes resulting in lower predictive accuracy [27]. Therefore, it is not surprising to observe a performance degradation on low-resolution datasets due to the loss of physical information between frames.

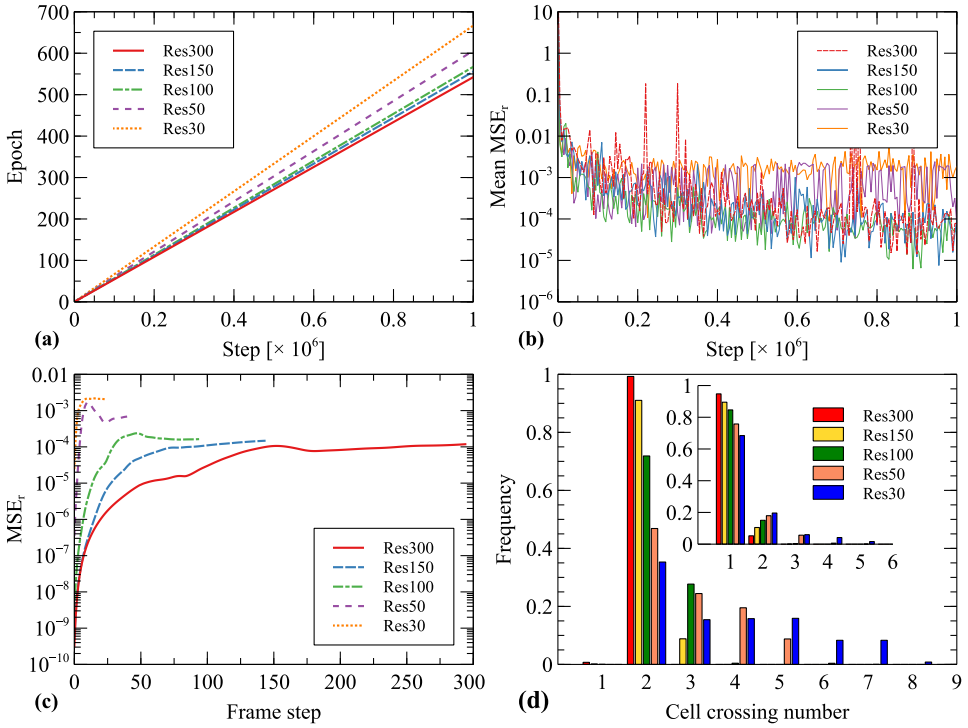

Fig. 15. Variation of (a) training epoch, (b) validation mean rollout MSE_r, (c) rollout MSE_r averaged over all validation cases, and (d) histograms of cell crossing number between two neighboring frames for datasets with different resolutions. Note: The frame number varies for different resolution datasets in (c). The maximum displacement per frame (the inset is for all material points) is used for the histograms in (d).

We consider validation Case 1 as an illustrative example to demonstrate the model performance with different dataset resolutions. Specifically, we compare snapshots of a collapsed granular column near the initial and final states, as depicted in Fig. 16. Taking the subfigures (a-1) and (a-2) as an example, we have Res300-frame50 and Res300-final, representing the 50th frame and the last frame of the trajectory with a resolution of 300 frames, respectively. The figure clearly indicates that only the lowest-resolution dataset (Res30) exhibits significant prediction errors in terms of material point positions, whereas the datasets with higher resolutions perform well.

5.4. Inductive performance

The inductive performance of the trained TGNNS model is assessed using an additional test dataset, which is outlined in Table 5. The positions of material points are normalized based on the size (10 m) of the training simulation box, unless otherwise specified. In the test dataset, we consider various factors, including column size, column number, and the presence of additional barriers, to evaluate the inductive performance of the TGNNS. These test trajectories can be divided into two groups based on the column height: those with a height less than 3 m and those with a height greater than or equal to 3 m. It should be noted that the training data only contains square columns with a size of 3 m. The final case presented in Table 5 involves a complex penetration scenario characterized by dynamic interactions between rigid boundaries and material points. This case is specifically designed to further evaluate the inductive performance of the trained model.

Fig. 17 shows the rollout MSE_r for each test trajectory. For the tests with a column height of 2 m, as shown in Fig. 17(a), the squared column Col2x2 has the smallest MSE_r, and Test Col2x2twins has a similar evolution of MSE_r as Test Col2x2. Increasing the column width, the MSE_r increases but not monotonically. For example, the peak MSE_r of Test Col5x2 reaches 10^{-4}, around four times larger than that of Test Col2x2, while Test Col10x2 has a peak MSE_r of around 10^{-4}. Regarding the MSE_r at the final states, the deviation between these tests is not significant, suggesting a good performance of the TGNNS on predicting granular column collapse regardless of column width. Nevertheless, the rollout MSE_r increases remarkably as the column height increases, as shown in Fig. 17(b). The squared column Col3x3barriers has the smallest MSE_r of 5×10^{-5}, even though two additional barriers are installed. It is worth noting that Col3x3barriers and Col5x3 have smaller MSE than Col2x2 even though they are taller than Col2x2. One major reason is that the column height of 3 m has already been seen by the TGNNS during training, and it will cause extra errors when extrapolating from the original height (3 m) to the unseen height (2 m).

Fig. 18 displays the snapshots of the granular columns Col2x2 and Col2x2twins during the collapsing process. Prior to the contact between the column twins, as depicted in Fig. 18(e), each twin exhibits a profile that closely resembles that of the single column

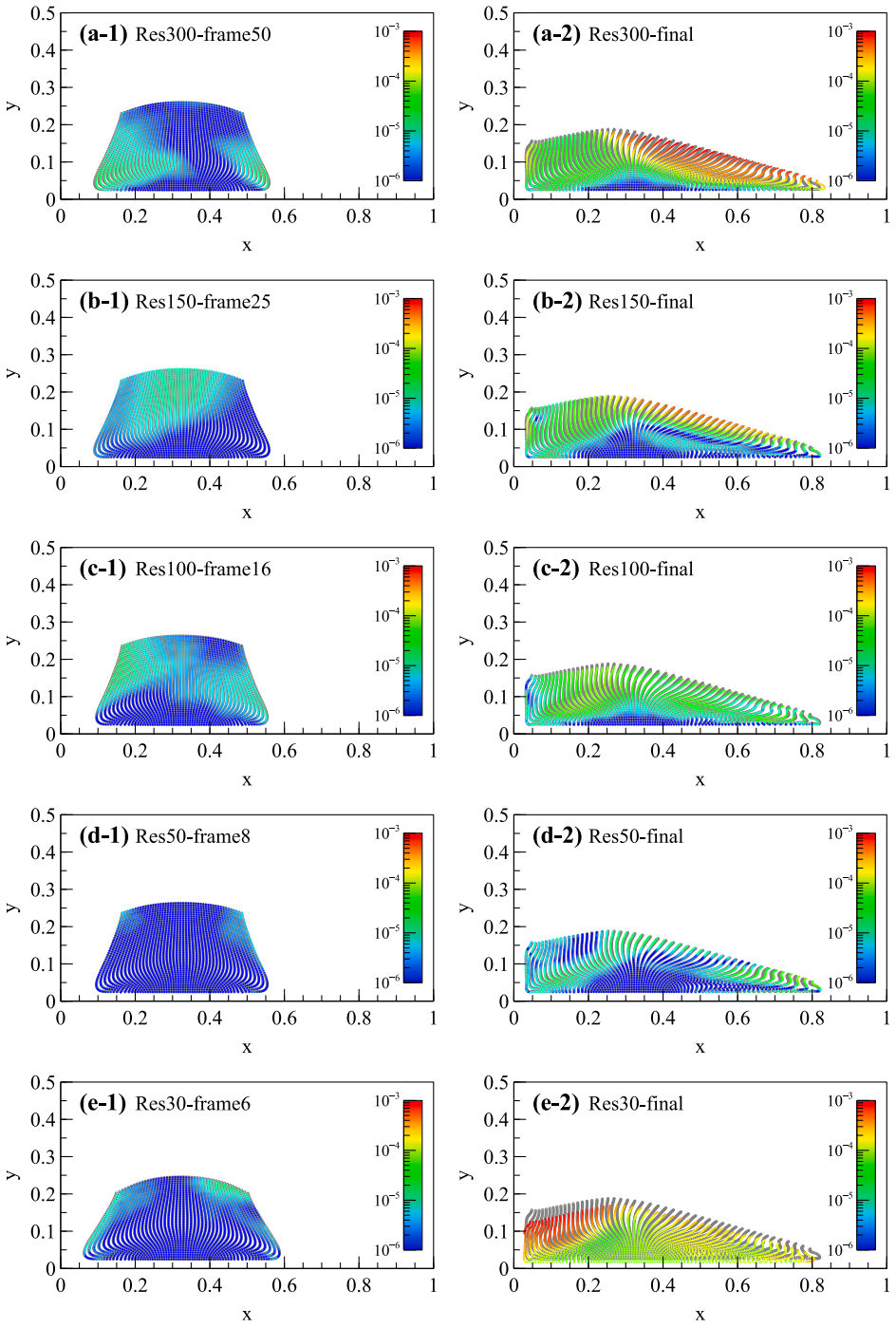

Fig. 16. Comparison of ground truth (gray) and learned (colorful) particle positions for Case 1 with different resolution datasets. The color map represents squared displacement errors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 5
Simulation configurations for the test dataset.

Test	Column size	Particle #		Time steps	Remark
	(w, h) [m]	MPM	DEM	[x1000]	
Col2x2	(2, 2)	1024	409 600	30	–
Col2x2twins	2x(2, 2)	2048	819 200	30	Two columns
Col5x2	(5, 2)	2560	1 024 000	30	–
Col10x2	(10, 2)	5120	2 048 000	30	–
Col5x3	(5, 3)	3840	1 536 000	30	–
Col5x4	(5, 4)	5120	2 048 000	50	–
Col5x5	(5, 5)	6400	2 560 000	50	–
Col3x3barriers	(3, 3)	2304	921 600	30	Two barriers
Penetration	–[a]	5120	2 048 000	50	Rigid plate loading

[a] Penetration is conducted on the stable configuration of collapsed Col10x2.

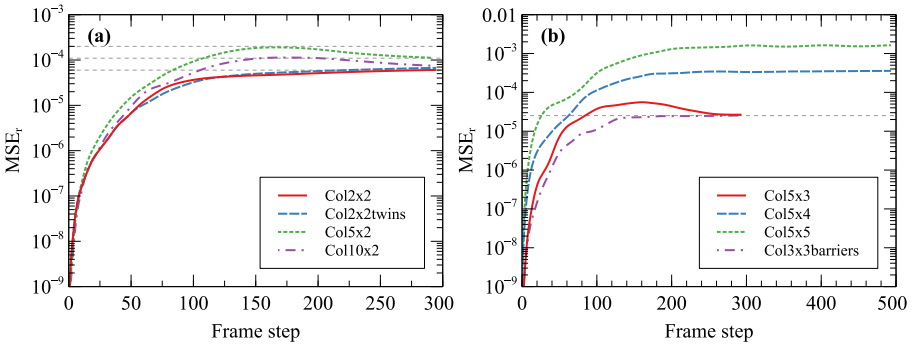

Fig. 17. Rollout MSE$_r$ for test trajectories: (a) column height $h < 3$ m and (b) column height $h \geq 3$ m. Note: the gray dashed lines serve as reference lines for peaks and plateaus of MSE$_r$.

case, Col2x2, as shown in Fig. 18(b). Subsequently, the column twins come into contact with each other, a phenomenon accurately predicted by the TGNNS. Notably, this additional physical process has not been observed in the training data, indicating the excellent inductive performance of the TGNNS.

Fig. 19 illustrates the collapse of the granular columns Col5x2 and Col10x2. It is noteworthy that the ground truth particle positions, depicted in gray, are almost entirely overlapped by the learned particle positions, indicating good model predictions for both cases. It is also can be seen that both columns exhibit similar patterns of granular flow, resulting in contours of squared displacement errors that are closely aligned. Notably, the squared displacement error is the most significant near the left inclined surface, while it is the smallest near the bottom. Moreover, we observe asymmetric contours in squared displacement error, suggesting a biased prediction by the TGNNS for nearly symmetric problems. To address this bias, one potential approach is to enhance the unbiased nature of the training data and employ more unbiased sampling techniques during the training process. However, a comprehensive study on this topic is beyond the scope of our current investigation and will be explored in future research. Furthermore, as the column width increases, more material points near the central part of the column are not influenced by the collapse, leading to a smaller MSE$_r$. This explains the smaller MSE$_r$ of Test Col10x2 depicted in Fig. 17(a).

Fig. 20 shows the snapshots of the granular columns Col5x3, Col5x4, and Col5x5, as they collapse. The predictability of run-out head becomes less accurate as the column height increases, particularly in the case of Test Col5x5. Notably, the model performance exhibits greater sensitivity to changes in column height than that in column width.

Fig. 21 presents the snapshots of a granular column collapsing and impacting two rigid barriers, A and B, referred to as Test Col3x3barriers. The frictional top and bottom faces of both barriers have the same friction coefficient as the bottom of the simulation box used during training. However, the left and right faces of the barriers are frictionless. It is important to note that we incorporate the boundary constraints directly into the model during inference, eliminating the need to learn the specific particle–barrier interaction during model training. The snapshots serve as validation for these hard-coded boundary constraints. This implies that there is no requirement to train the model for the particle–barrier interaction in this particular scenario. However, for more complex boundaries with diverse geometric and physical properties, it may be necessary to train the model to handle the complex particle–boundary interactions. One approach to achieve this is by introducing boundary primitives, such as line segments for 2D or triangular facets for 3D, as previously described in a related work [37].

In our endeavor to deepen our comprehension of intricate boundaries, we have undertaken an examination of the behavior of a classical rigid footing problem in geotechnical engineering using our sophisticated trained model. Building upon the results of the Col10x2 test, we have subjected a granular bed to penetration loading by steadily lowering a plate at a constant velocity of 0.1 m/s. The penetration process commences at Frame 400, marking the point at which the granular column transitions into a state

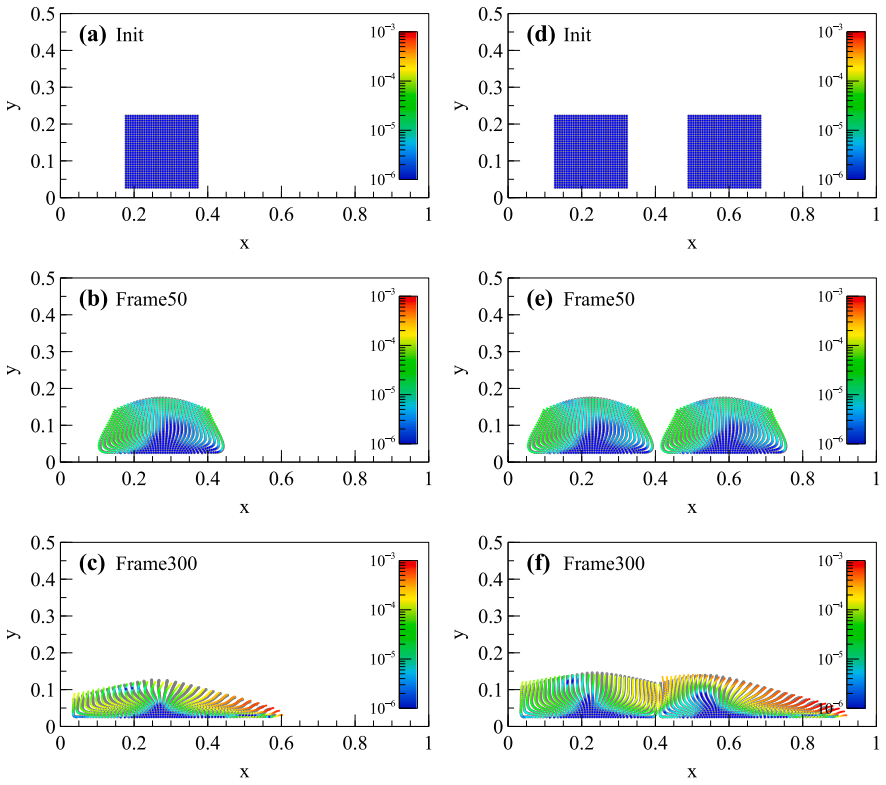

Fig. 18. Comparison of ground truth (gray) and learned (colorful) particle positions during column collapsing for: Col2x2 (*left*) and Col2x2twins (*right*). The color map represents squared displacement errors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of equilibrium. As illustrated in Fig. 22, our TGNNS encounters difficulties in accurately predicting the deformation of the granular bed during penetration. This challenge primarily arises from the insufficient capture of repulsive interactions among the constituent material particles, underscoring a notable limitation in the predictive capabilities of the present trained model. One significant reason for the limitation is the lack of adequate training data to capture the repulsive interactions between material points. The existing training dataset is derived exclusively from column collapse tests, which do not emphasize these repulsive interactions. As a result, the model's understanding of inter-particle interactions is confined to dynamic collapse scenarios, which differ markedly from the penetration problem that is primarily governed by repulsive forces. To enhance the performance of the current TGNNS model, two potential strategies can be employed: (1) incorporating new training data specifically related to penetration problems, and (2) integrating additional features, such as stress, into the model.

Remarks: The TGNNS demonstrates good performance when applied to granular columns with heights that are not greater than the training height, regardless of their widths. However, when the column height exceeds the training height, the model performance significantly deteriorates. The question arises: why does the column height matter? In fact, the unseen column height implies a substantial amount of information that remains unknown to the model. One crucial piece of information is the stress state of a granular column at its initial state. The vertical stress can be approximately proportional to the column height (as illustrated in Fig. 6(a)). The stress state has the potential to influence the porosity within the granular material. Moreover, different column heights can trigger varying magnitudes of velocities during column collapse. Therefore, there are two potential avenues for improving the TGNNS. First, one could introduce physics-based measures, such as stress state and porosity, into the model, thereby enhancing its ability to incorporate physical principles. The second approach involves incorporating larger granular columns into the training data. However, this approach may not completely resolve the inductive issue that arises when predicting the collapse of much larger columns. In our future study, we will explore a comprehensive enhancement of the TGNNS that takes into account these considerations.

5.5. Computational efficiency

We begin by analyzing the training efficiency of different network architectures employed in the ablation study. The training process is carried out on 2 NVIDIA H800 GPUs, utilizing a mini-batch size of 2 without any gradient accumulation. Table 6 presents the parameter counts, which include the total number of learnable parameters (weights and biases), along with the corresponding

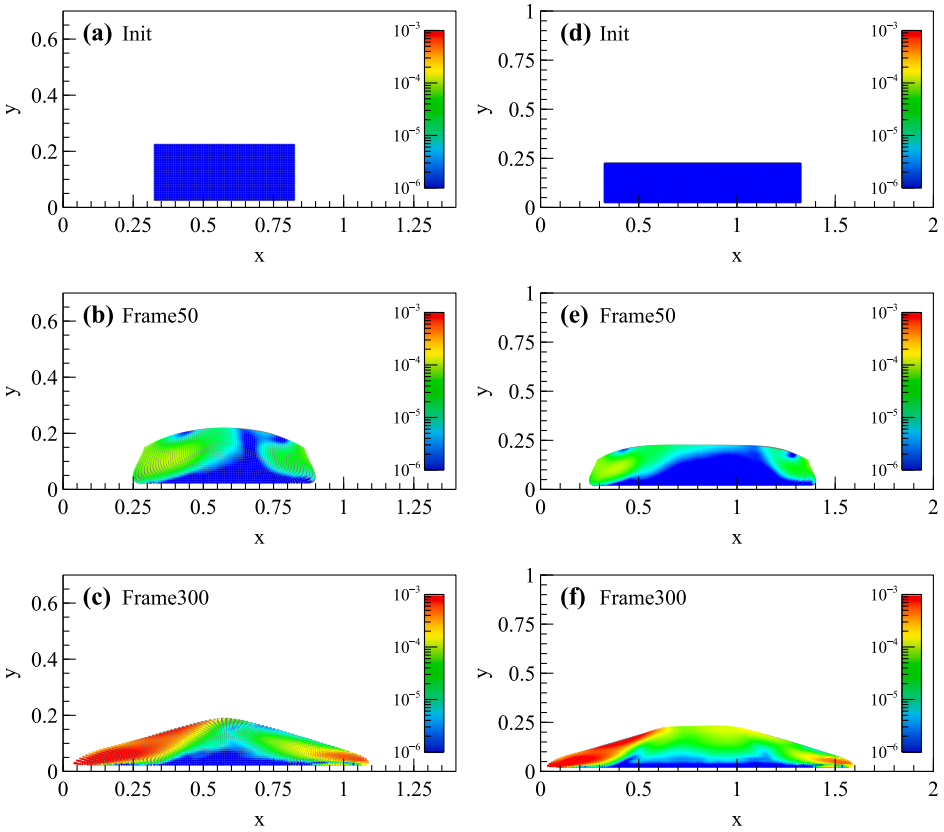

Fig. 19. Comparison of ground truth (gray) and learned (colorful) particle positions during column collapsing for: (a–c) Col5x2 (*left*) and (d–f) Col10x2 (*right*). The color map represents squared displacement errors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 6
Elapsed time for training the TGNNS 1 million steps with various network architectures.

Network	Parameter #	Total elapsed time [h][a]	Speed [ms/step]
Baseline	1 425 282	20.98	70.5
Hidden1	1 062 018	19.43	64.7
Msg1	1 276 546	19.92	66.1
Msg0	1 127 810	18.70	61.9
Msg0-hidden1	830 594	16.45	54.6
Frame4	1 127 810	15.67	51.5
Frame4-msg0-hidden1	599 170	11.42	37.2
Frame4-msg1	979 074	14.01	46.3
Frame4-hidden1	830 594	14.42	47.6
Frame4-msg1-hidden1-edgemsg	748 802	25.75	85.3

[a] It encompasses the time taken for model saving and validation.

total elapsed time. The total elapsed time encompasses both the duration required for model saving and the time spent on validation at intervals of 5000 steps during training. It is important to note that the average speed does not take into account these time intervals.

The computational time for the test datasets is summarized in Table 7. The direct numerical simulations are conducted on an NVIDIA GeForce RTX 4090 GPU using DEMPM, while the TGNNS runs on an NVIDIA H800 GPU or RTX 4090 GPU. The TGNNS predictions are made with the Baseline model. As shown in the table, the TGNNS achieves an average speedup of 130 and 85 on H800 and RTX 4090, respectively. It is evident that H800 is approximately 1.5x faster than RTX 4090 for our specific tests. It is important to note that there is room for improvement in the current implementation of the TGNNS prediction. For instance, the Baseline model requires six graph constructions to predict a single frame, but this can be reduced to just one by caching the previous five graphs. Moreover, it is worth mentioning that the present GPU-accelerated DEMPM is highly efficient. In a previous study [41], we observed that the GPU-accelerated DEMPM running on an older NVIDIA GeForce RTX 2080 Ti GPU (which is only

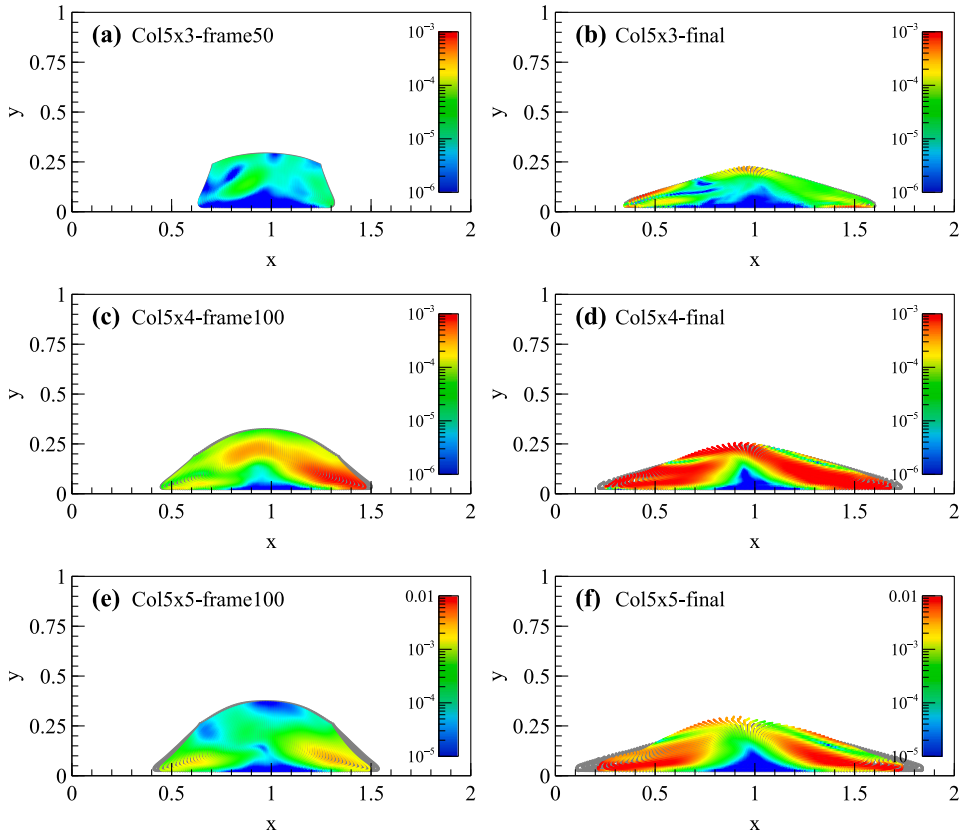

Fig. 20. Comparison of ground truth (gray) and learned (colorful) particle positions during column collapsing for: (a–b) Col5x3 (*top*), (c–d) Col5x4 (*middle*), and (e–f) Col5x5 (*bottom*). The color map represents squared displacement errors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

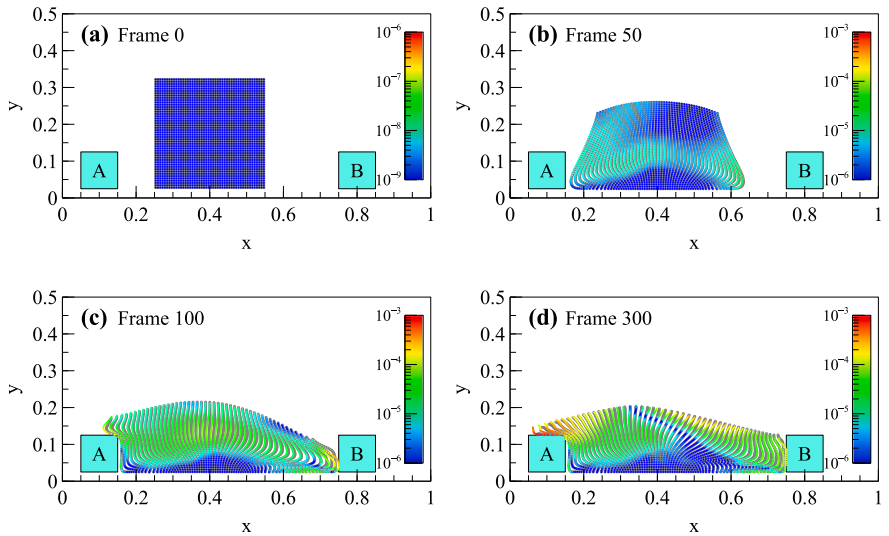

Fig. 21. Comparison of ground truth (gray) and learned (colorful) particle positions during column collapsing against two barriers. The color map represents squared displacement errors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

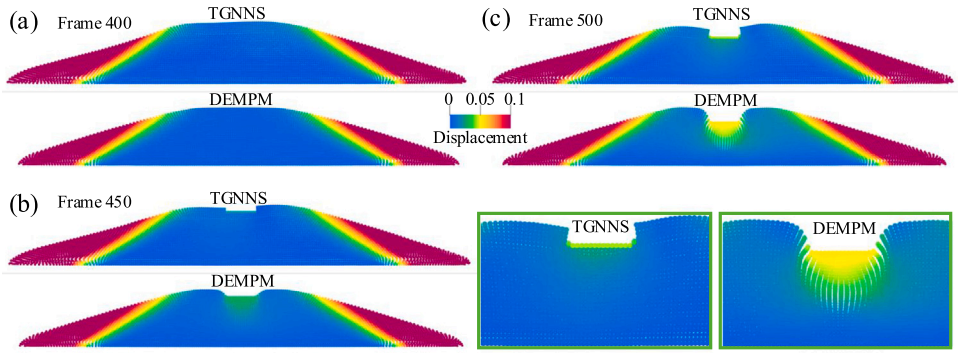

Fig. 22. Snapshots of a rigid footing penetrating into a granular bed.

Table 7
Elapsed time for the direct numerical simulations and TGNNS predictions for test.

Test	DEMPM [s]	TGNNS[a] [s]		Speedup	
	RTX 4090	RTX 4090	H800	RTX 4090	H800
Col2x2	325	4.8	3.2	68	102
Col2x2twins	639	7.6	5.1	84	125
Col5x2	948	9.3	6.0	102	158
Col10x2	1718	16.5	11.0	104	156
Col5x3	1370	13.3	8.4	103	163
Col5x4	1614	28.3	17.9	57	90
Col5x5	2299	35.2	22.2	65	104
Col3x3barriers	962	10.0	6.6	96	146

[a] The Baseline model is used.

half as powerful as the current RTX 4090 GPU) was 90 times faster than its CPU counterparts running on 44 CPU threads. A rough estimation suggests that the TGNNS could be 9000 times faster than the CPU-based DEMPM.

6. Conclusions

We introduce the Temporal Graph Neural Network-based Simulator (TGNNS), a novel deep learning-based simulator designed specifically for modeling granular materials. The TGNNS takes a sequence of frames, each representing the positions of material points, as input. Each frame is represented as a distinct graph with particle properties as nodes. The dynamics of the particles propagate through the sequence of frames graph by graph, resembling the operation of a Lagrangian method where particles carry their dynamics information. As a consequence, the TGNNS offers a more physically grounded architecture for learning granular flows compared to other graph-based neural networks described in the literature. This enhanced interpretability is expected to yield more reasonable model behavior.

We extensively train, validate, and test the TGNNS using simulation data generated from a hierarchical multiscale modeling approach (coupling of MPM and DEM) to granular column collapse. We also conduct a comprehensive examination of ablative neural networks derived from the TGNNS baseline. The results demonstrate that, after training for one million steps, the TGNNS achieves good model performance, with the baseline model outperforming other ablative architectures. Moreover, we train the TGNNS using datasets of varying resolutions in frame. Notably, the TGNNS struggles to train effectively on the lowest-resolution dataset due to a significant loss of physical information between frames. However, this limitation highlights the physically-sound architecture of the TGNNS.

Regarding the inductive performance of the TGNNS, the trained model exhibits strong performance when presented with previously unseen datasets of varying sizes, even in the presence of additional barriers. This suggests that the proposed neural network architecture effectively learns and captures the inter-particle interactions. Nevertheless, it is worth noting that its sensitivity to column height (or the initial stress level) could potentially restrict its applicability in predicting granular flows with high initial stress levels. In order to address this limitation, there are two potential avenues for enhancing the TGNNS in future studies: (i) One could consider incorporating the initial stress level, or related features such as porosity, into the TGNNS framework; and (ii) it could also prove beneficial to expand the training data to encompass taller granular columns. Furthermore, the TGNNS may not be suitable for quasi-static problems, as showcased in the penetration simulation, while a further sophisticated design of node and edge features can be helpful to enhance the performance of the TGNNS in handling quasi-static problems. Overall, in spite of its limitations, the TGNNS has three potential advantages across three key facets: (i) the utilization of dynamic graphs, (ii) the implementation of inter-graph message passing, and (iii) the mitigation of challenges associated with neighbor expansion.

CRediT authorship contribution statement

Shiwei Zhao: Writing – original draft, Visualization, Software, Methodology, Funding acquisition, Conceptualization. **Hao Chen:** Writing – review & editing, Visualization, Data curation. **Jidong Zhao:** Writing – review & editing, Supervision, Resources, Funding acquisition.

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.

Acknowledgments

This work was financially supported by Research Grants Council of Hong Kong (by GRF Projects No. 16206322 and No. 16211221, by CRF Project No. C7082-22G, and by TRS Project No. T22-606/23-R), the Guangdong Basic and Applied Basic Research Foundation (2022A1515010848). The authors acknowledge the SuperPOD resources supported by the Information Technology Services Center (ITSC) at HKUST. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the financial bodies.

Data availability

The simulation data can be reproduced using the DEMPM package from our developed software SudoSim, which is available for free download from the download page at https://www.sudosimlab.com/en/download/.

References

[1] J. He, L. Zhang, T. Xiao, H. Wang, H. Luo, Prompt quantitative risk assessment for rain-induced landslides, J. Geotech. Geoenviron. Eng. 149 (5) (2023) 04023023.
[2] T. Zohdi, A machine-learning digital-twin for rapid large-scale solar-thermal energy system design, Comput. Methods Appl. Mech. Engrg. 412 (2023) 115991.
[3] M. Torzoni, M. Tezzele, S. Mariani, A. Manzoni, K.E. Willcox, A digital twin framework for civil engineering structures, Comput. Methods Appl. Mech. Engrg. 418 (2024) 116584.
[4] D. Sulsky, S.-J. Zhou, H.L. Schreyer, Application of a particle-in-cell method to solid mechanics, Comput. Phys. Comm. 87 (1–2) (1995) 236–252.
[5] R.A. Gingold, J.J. Monaghan, Smoothed particle hydrodynamics: theory and application to non-spherical stars, Mon. Not. R. Astron. Soc. 181 (3) (1977) 375–389.
[6] H.H. Bui, R. Fukagawa, K. Sako, S. Ohno, Lagrangian meshfree particles method (sph) for large deformation and failure flows of geomaterial using elastic–plastic soil constitutive model, Int. J. Numer. Anal. Methods Geomech. 32 (12) (2008) 1537–1570.
[7] K. Soga, E. Alonso, A. Yerro, K. Kumar, S. Bandara, Trends in large-deformation analysis of landslide mass movements with particular emphasis on the material point method, Géotechnique 66 (3) (2016) 248–273.
[8] P.A. Cundall, O.D. Strack, A discrete numerical model for granular assemblies, Geotechnique 29 (1) (1979) 47–65.
[9] R. Kawamoto, E. Andò, G. Viggiani, J.E. Andrade, All you need is shape: Predicting shear banding in sand with ls-dem, J. Mech. Phys. Solids 111 (2018) 375–392.
[10] J. Zhao, S. Zhao, S. Luding, The role of particle shape in computational modelling of granular matter, Nat. Rev. Phys. 5 (9) (2023) 505–525.
[11] S. Zhao, Z. Lai, J. Zhao, Leveraging ray tracing cores for particle-based simulations on gpus, Internat. J. Numer. Methods Engrg. 124 (3) (2023) 696–713.
[12] 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.
[13] 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.
[14] N. Guo, J. Zhao, A coupled fem/dem approach for hierarchical multiscale modelling of granular media, Internat. J. Numer. Methods Engrg. 99 (11) (2014) 789–818.
[15] W. Liang, J. Zhao, Multiscale modeling of large deformation in geomechanics, Int. J. Numer. Anal. Methods Geomech. 43 (5) (2019) 1080–1114.
[16] K. Willcox, B. Segundo, The role of computational science in digital twins, Nat. Comput. Sci. 4 (3) (2024) 147–149.
[17] N.N. Vlassis, W. Sun, Sobolev training of thermodynamic-informed neural networks for interpretable elasto-plasticity models with level set hardening, Comput. Methods Appl. Mech. Engrg. 377 (2021) 113695.
[18] T. Qu, J. Zhao, S. Guan, Y. Feng, Data-driven multiscale modelling of granular materials via knowledge transfer and sharing, Int. J. Plast. 171 (2023) 103786.
[19] J.N. Fuhg, G.A. Padmanabha, N. Bouklas, B. Bahmani, W. Sun, N.N. Vlassis, M. Flaschel, P. Carrara, L. De Lorenzis, A review on data-driven constitutive laws for solids, arXiv preprint arXiv:2405.03658.
[20] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2008) 61–80.
[21] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, et al., Interaction networks for learning about objects, relations and physics, Adv. Neural Inf. Process. Syst. 29.
[22] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., Relational inductive biases, deep learning, and graph networks, arXiv preprint arXiv:1806.01261.
[23] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P. Battaglia, Learning to simulate complex physics with graph networks, in: International Conference on Machine Learning, PMLR, 2020, pp. 8459–8468.
[24] Y. Choi, K. Kumar, Graph neural network-based surrogate model for granular flows, Comput. Geotech. 166 (2024) 106015.
[25] 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.

[26] S. Yang, H. Kim, Y. Hong, K. Yee, R. Maulik, N. Kang, Data-driven physics-informed neural networks: A digital twin perspective, arXiv preprint arXiv:2401.08667.
[27] W.J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, B. Yu, Definitions, methods, and applications in interpretable machine learning, Proc. Natl. Acad. Sci. 116 (44) (2019) 22071–22080.
[28] S.G. Bardenhagen, E.M. Kober, et al., The generalized interpolation material point method, Comput. Model. Eng. Sci. 5 (6) (2004) 477–496.
[29] J.J. Monaghan, Smoothed particle hydrodynamics, Annu. Rev. Astron. Astrophys. 30 (A93-25826 09-90) (1992) 543–574.
[30] S.A. Silling, Reformulation of elasticity theory for discontinuities and long-range forces, J. Mech. Phys. Solids 48 (1) (2000) 175–209.
[31] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, M. Bronstein, Temporal graph networks for deep learning on dynamic graphs, arXiv preprint arXiv:2006.10637.
[32] A. Longa, V. Lachi, G. Santin, M. Bianchini, B. Lepri, P. Lio, F. Scarselli, A. Passerini, Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities, arXiv preprint arXiv:2302.01018.
[33] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, Adv. Neural Inf. Process. Syst. 30.
[34] M. Fey, J.E. Lenssen, F. Weichert, J. Leskovec, Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings, in: International Conference on Machine Learning, PMLR, 2021, pp. 3294–3304.
[35] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, C.-J. Hsieh, Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 257–266.
[36] Y. Rubanova, A. Sanchez-Gonzalez, T. Pfaff, P. Battaglia, Constraint-based graph network simulator, arXiv preprint arXiv:2112.09161.
[37] A. Mayr, S. Lehner, A. Mayrhofer, C. Kloss, S. Hochreiter, J. Brandstetter, Boundary graph neural networks for 3d simulations, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 37, 2023, pp. 9099–9107.
[38] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch.
[39] M. Fey, J.E. Lenssen, Fast graph representation learning with PyTorch Geometric, in: ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.
[40] J.L. Ba, J.R. Kiros, G.E. Hinton, Layer normalization, arXiv preprint arXiv:1607.06450.
[41] S. Zhao, J. Zhao, W. Liang, A thread-block-wise computational framework for large-scale hierarchical continuum-discrete modeling of granular media, Internat. J. Numer. Methods Engrg. 122 (2) (2021) 579–608.
[42] S. Zhao, H. Chen, J. Zhao, Multiscale modeling of freeze-thaw behavior in granular media, Acta Mech. Sin. 39 (1) (2023) 722195.
[43] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.
[44] M. Steininger, K. Kobs, P. Davidson, A. Krause, A. Hotho, Density-based weighting for imbalanced regression, Mach. Learn. 110 (2021) 2187–2211.
[45] R. Ni, X. Zhang, A precise critical time step formula for the explicit material point method, Internat. J. Numer. Methods Engrg. 121 (22) (2020) 4989–5016.